produces random values with a uniform distribution between the required arguments for minimum and maximum. After this edit, TEv, with the command-line argument "a", can be used to view the amplitude for all Textures and confirm the edit.

Example 4-16. Editing a single parameter of all Textures with TEe

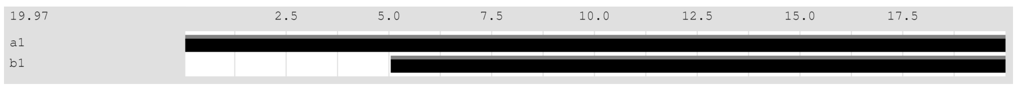
```
pi{auto-muteHiConga}ti{b1} :: tee
edit all TextureInstances
which parameter? (i,t,b,r,p,f,o,a,n,x): a
sample amplitude: randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
new value: ru, .6, 1
TI a1: parameter amplitude updated.
TI b1: parameter amplitude updated.

pi{auto-muteHiConga}ti{b1} :: tev a
compare parameters: amplitude
{name,value,}
a1                randomUniform, (constant, 0.6), (constant, 1)
b1                randomUniform, (constant, 0.6), (constant, 1)
```

Using ELn, the current collection of Textures can be used to create an EventList, and ELh may be used to audition the results. (For more information on using ELn, see Section 2.5.) The random fluctuation of amplitude values should provide a variety of accent patterns to the fixed rhythmic loop.

The collection of Textures can be displayed in a graphical and textual diagram produced by the TEmap command. This command lists each Texture and Clone within the current AthenaObject and provides a proportional representation of their respective start and end times.

Example 4-17. Generating a graphical display of Texture position with TEmap

```
pi{auto-muteHiConga}ti{b1} :: temap
TextureEnsemble Map:
19.97s            |   .   |   .   |   .   |   .   |
a1                _______________________________________________
b1                        _______________________________________
```

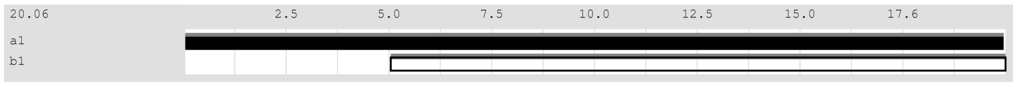


4.7. Muting Textures

Textures can be muted to disable the inclusion of their events in all EventOutputs. Textures and their Clones (see Chapter 6) can be muted independently. The command TImute, if no arguments are given, toggles the current Texture's mute status. The following example demonstrates muting Texture a1, listing all Textures with with TIls, and then displaying the collection of Textures with the TEmap command. Notice that in the TIls display, the "status" of Texture a1 is set to "o", meaning that it is muted.

Example 4-18. Muting a Texture with TImute

```
pi{auto-muteHiConga}ti{b1} :: timute
TI b1 is now muted.

pi{auto-muteHiConga}ti{b1} :: tils
TextureInstances available:
{name,status,TM,PI,instrument,time,TC}
   a1              + LineGroove  auto-lowConga    64  00.0--20.0   0
 + b1              o LineGroove  auto-muteHiConga 62  05.0--20.0   0

pi{auto-muteHiConga}ti{b1} :: temap
TextureEnsemble Map:
19.97s              |      .      |      .      |      .      |      .      |
a1                  ___________________________________________________________
b1                                _____________________________________________
```



By providing the name of one or more Textures as command-line arguments, numerous Texture's mute status can be toggled. In the following example, Texture a1 is given as an argument to the TImute command. The TIls command shows that the Texture is no longer muted.

Example 4-19. Removing mute status with TImute

```
pi{auto-muteHiConga}ti{b1} :: timute a1
TI a1 is now muted.

pi{auto-muteHiConga}ti{b1} :: tils
TextureInstances available:
{name,status,TM,PI,instrument,time,TC}
   a1              o LineGroove  auto-lowConga    64  00.0--20.0   0
 + b1              o LineGroove  auto-muteHiConga 62  05.0--20.0   0

pi{auto-muteHiConga}ti{b1} :: timute a1
TI a1 is no longer muted.

pi{auto-muteHiConga}ti{b1} :: timute b1
TI b1 is no longer muted.
```

4.8. Viewing and Searching ParameterObjects

For each dynamic attribute of a TextureInstance, a ParameterObject can be assigned to produce values over the duration of the Texture. Complete documentation for all ParameterObjects can be found in Appendix C. Texture attributes for bpm, local field, local octave, amplitude, panning, and all auxiliary parameters (if required by the instrument) can have independent ParameterObjects.

ParameterObjects are applied to a Texture attribute with an argument list. athenaCL accepts lists in the same comma-separated format of Python list data structures. A list can consist of elements like strings, numbers, and other lists, each separated by a comma. Within athenaCL, text strings need not be in quotes, and sub-lists can be given with either parenthesis or brackets. Each entry in the ParameterObject argument list corresponds, by ordered-position, to an internal setting within the ParameterObject. The first entry in the argument list is always the name of the ParameterObject. ParameterObject names, as well as all ParameterObject configuration strings, can always be accessed with acronyms.

To display a list if all available ParameterObjects, enter the command TPls, for TextureParameter list:

Example 4-20. Displaying all ParameterObjects with TPls

```
pi{auto-muteHiConga}ti{b1} :: tpls
Generator ParameterObject
{name}
   accumulator
   basketFill
   basketFillSelect
   basketGen
   basketSelect
   breakGraphFlat
   breakGraphHalfCosine
   breakGraphLinear
   breakGraphPower
   breakPointFlat
   breakPointHalfCosine
   breakPointLinear
   breakPointPower
   caList
   caValue
   constant
   constantFile
   cyclicGen
   directorySelect
   envelopeGeneratorAdsr
   envelopeGeneratorTrapezoid
   envelopeGeneratorUnit
   feedbackModelLibrary
   fibonacciSeries
   funnelBinary
   grammarTerminus
   henonBasket
   iterateCross
   iterateGroup
   iterateHold
   iterateSelect
   iterateWindow
   lineSegment
   listPrime
   logisticMap
   lorenzBasket
   markovGeneratorAnalysis
   markovValue
   mask
   maskReject
   maskScale
   noise
   oneOver
   operatorAdd
   operatorCongruence
   operatorDivide
   operatorMultiply
   operatorPower
   operatorSubtract
   pathRead
   quantize
   randomBeta
   randomBilateralExponential
   randomCauchy
   randomExponential
   randomGauss
   randomInverseExponential
   randomInverseLinear
   randomInverseTriangular
   randomLinear
   randomTriangular
   randomUniform
   randomWeibull
   sampleAndHold
   sampleSelect
   sieveFunnel
   sieveList
   staticInst
   staticRange
   typeFormat
   valuePrime
   valueSieve
   waveCosine
   waveHalfPeriodCosine
   waveHalfPeriodPulse
   waveHalfPeriodSine
   waveHalfPeriodTriangle
   wavePowerDown
   wavePowerUp
   wavePulse
   waveSawDown
   waveSawUp
   waveSine
   waveTriangle

Rhythm Generator ParameterObject
{name}
   binaryAccent
   convertSecond
   convertSecondTriple
   gaRhythm
   iterateRhythmGroup
   iterateRhythmHold
   iterateRhythmWindow
   loop
   markovPulse
   markovRhythmAnalysis
   pulseSieve
   pulseTriple
   rhythmSieve

Filter ParameterObject
{name}
   bypass
   filterAdd
   filterDivide
   filterDivideAnchor
   filterFunnelBinary
   filterMultiply
   filterMultiplyAnchor
   filterPower
```

```
    filterQuantize
    maskFilter
    maskScaleFilter
    orderBackward
    orderRotate
    pipeLine
    replace
```

To display detailed documentation for a ParameterObject, enter the command TPv, for Texture Parameter view. In the following example the user views the ParameterObjects "wavePowerDown" and "noise" by providing command-line arguments for the desired ParameterObject name:

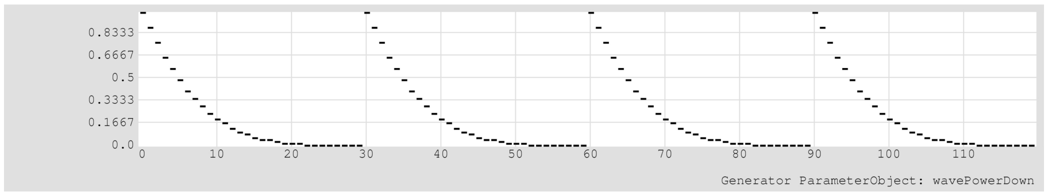
Example 4-21. Viewing ParameterObject reference information

```
pi{auto-muteHiConga}ti{b1} :: tpv wpd
Generator ParameterObject
{name,documentation}
WavePowerDown       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

pi{auto-muteHiConga}ti{b1} :: tpv noise
Generator ParameterObject
{name,documentation}
Noise               noise, resolution, parameterObject, min, max
                    Description: Fractional noise (1/fn) Generator, capable of
                    producing states and transitions between 1/f white, pink,
                    brown, and black noise. Resolution is an integer that
                    describes how many generators are used. The gamma argument
                    determines what type of noise is created. All gamma values
                    are treated as negative. A gamma of 0 is white noise; a
                    gamma of 1 is pink noise; a gamma of 2 is brown noise; and
                    anything greater is black noise. Gamma can be controlled by
                    a dynamic ParameterObject. The value produced by the noise
                    generator is scaled within the unit interval. This
                    normalized value is then scaled within the range designated
                    by min and max; min and max may be specified by
                    ParameterObjects. Arguments: (1) name, (2) resolution, (3)
                    parameterObject {gamma value as string or number}, (4) min,
                    (5) max
```

The command TPmap provides graphical displays of ParameterObject-generated values. (To configure athenaCL graphics output, see Example 1-11.) The user must supply the name of the ParamaterObject library (Generator, Rhythm, or Filter), the number of events to generate, and the ParameterObject argument list.

Example 4-22. ParameterObject Map display with TPmap

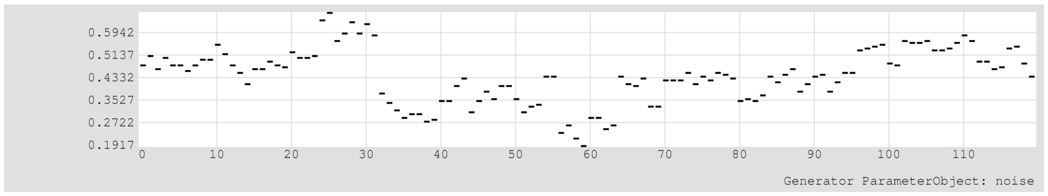
```
pi{auto-muteHiConga}ti{b1} :: tpmap
select a library: Generator, Rhythm, or Filter. (g, r, f): g
number of events: 120
enter a Generator ParameterObject argument: wpd, e, 30, 0, 2

wavePowerDown, event, (constant, 30), 0, 2, (constant, 0), (constant, 1)
TPmap display complete.
```



The TPmap, like other athenaCL commands, can be used with command-line arguments. In the following example, the user produces a TPmap display of the noise ParameterObject, generating "brown" fractional noise:

Example 4-23. ParameterObject Map display with TPmap

```
pi{auto-muteHiConga}ti{b1} :: tpmap 120 n,50,(c,2),0,1

noise, 50, (constant, 2), (constant, 0), (constant, 1)
TPmap display complete.
```



4.9. Editing ParameterObjects

To edit an attribute of Texture, a user enters a new ParameterObject argument list. The command TIe, as before, first prompts the user to select which attribute to edit. Next, the current value of the attribute is displayed, followed by a prompt for the new value. TIv can be used to confirm the changed value. In the following example, the panning of Texture "a1" is assigned a fractional noise (1/f) ParameterObject:

Example 4-24. Editing the panning of a TextureInstance

```
pi{auto-muteHiConga}ti{b1} :: tie
edit TI b1
which parameter? (i,t,b,r,p,f,o,a,n,x,s,d): n
current panning: constant, 0.5
```

```
new value: n, 50, (cg, ud, 1, 3, .2), .5, 1
TI b1: parameter panning updated.

pi{auto-muteHiConga}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: o, duration: 005.0--19.97
(i)nstrument        62 (generalMidiPercussion: muteHiConga)
(t)ime range        05.0--20.0
(b)pm               constant, 120
(r)hythm            pulseTriple, (constant, 4), (basketGen, randomPermutate,
                    (1,1,2,3)), (constant, 1), (constant, 0.75)
(p)ath              auto-muteHiConga
                    (D4)
                    15.00(s)
local (f)ield       constant, 0
local (o)ctave      constant, 0
(a)mplitude         randomUniform, (constant, 0.6), (constant, 1)
pan(n)ing           noise, 50, (cyclicGen, upDown, 1, 3, 0.2), (constant, 0.5),
                    (constant, 1)
au(x)iliary         none
texture (s)tatic
      s0            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

The noise ParameterObject has been given an embedded ParameterObject to control the gamma argument. Notice that instead of entering "noise", "cyclicGen" or "upDown" the user can enter the acronyms "n", "cg", and "ud". All ParameterObjects support automatic acronym expansion of argument strings. This is an important and time-saving shortcut.

The previous example edited the panning of Texture "a1" such that it produces values within the range of .5 to 1. This limits the spatial location of the sound to the upper half of the range (the middle to right stereo position). To limit the spatial location of "b1" in a complementary fashion, the Texture is edited to produce values within the range 0 to .5, corresponding to the lower half of the range (the middle to left stereo position). In the example below, TIo is used to select "b1" before entering the TIe command. TEv is then used to compare panning values for all Textures.

Example 4-25. Editing the panning of a TextureInstance

```
pi{auto-muteHiConga}ti{b1} :: tio b1
TI b1 now active.

pi{auto-muteHiConga}ti{b1} :: tie n wpd,e,15,.25,2.5,0,.5
TI b1: parameter panning updated.

pi{auto-muteHiConga}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: o, duration: 005.0--20.06
(i)nstrument        62 (generalMidiPercussion: muteHiConga)
(t)ime range        05.0--20.0
(b)pm               constant, 120
(r)hythm            pulseTriple, (constant, 4), (basketGen, randomPermutate,
                    (1,1,2,3)), (constant, 1), (constant, 0.75)
(p)ath              auto-muteHiConga
                    (D4)
                    15.00(s)
local (f)ield       constant, 0
local (o)ctave      constant, 0
(a)mplitude         randomUniform, (constant, 0.6), (constant, 1)
pan(n)ing           wavePowerDown, event, (constant, 15), 0.25, 2.5, (constant,
                    0), (constant, 0.5)
au(x)iliary         none
texture (s)tatic
      s0            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none

pi{auto-muteHiConga}ti{b1} :: tev n
compare parameters: panning
{name,value,}
a1                  constant, 0.5
b1                  wavePowerDown, event, (constant, 15), 0.25, 2.5, (constant,
                    0), (constant, 0.5)
```

Notice that, in the above example, the user provided complete command-line arguments for the TIe command. When entering a ParameterObject from the command-line, no spaces, and only commas, can be used between ParameterObject arguments. As command-line arguments are space delimited (and ParameterObject arguments are comma delimited), a ParameterObject on the command line must be given without spaces between arguments. When providing a ParameterObject to a TIe prompt, however, spaces may be provided.

4.10. Editing Rhythm ParameterObjects

Rhythm ParameterObjects are ParameterObjects specialized for generating time and rhythm information. Many Rhythm ParameterObjects use Pulse object notations to define proportional rhythms and reference a Texture's dynamic bpm attribute. Other ParameterObjects are independent of bpm and can use raw timing information provided by one or more Generator ParameterObjects.

When using proportional rhythms, athenaCL uses Pulse objects. Pulses represent a ratio of duration in relation to the duration of beat (specified in BPM and obtained from the Texture). For details on Pulse notation, enter "help pulse":

Example 4-26. View Pulse and Rhythm help

```
pi{auto-muteHiConga}ti{b1} :: help pulse
{topic,documentation}
Pulse and Rhythm    Pulses represent a duration value derived from ratio and a
                    beat-duration. Beat duration is always obtained from a
                    Texture. Pulses are noted as a list of three values: a
                    divisor, a multiplier, and an accent. The divisor and
                    multiplier must be positive integers greater than zero.
                    Accent values must be between 0 and 1, where 0 is a measured
                    silence and 1 is a fully sounding event. Accent values my
                    alternatively be notated as + (for 1) and o (for 0). If a
```

```
beat of a given duration is equal to a quarter note, a Pulse
of (1,1,1) is a quarter note, equal in duration to a beat. A
Pulse of (2,1,0) is an eighth-note rest: the beat is divided
by two and then multiplied by one; the final zero designates
a rest. A Pulse of (4,3,1) is a dotted eight note: the beat
is divided by four (a sixteenth note) and then multiplied by
three; the final one designates a sounding event. A Rhythm
is designated as list of Pulses. For example: ((4,2,1),
(4,2,1), (4,3,1)).
```

To edit the rhythms used by Texture b1, enter TIe followed by an "r" to access the rhythm attribute. As before, the user is presented with the current value, then prompted for a new value. In the following example, the ParameterObject "loop" is examined first with the TPv, then the active Texture is edited by providing an random walk over an expanded rhythm. Finally, the rhythm attribute of all Textures is viewed with TEv.

Example 4-27. Editing Rhythm ParameterObjects with TIe

```
pi{auto-muteHiConga}ti{b1} :: tpv loop
Rhythm Generator ParameterObject
{name,documentation}
Loop                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'}

pi{auto-muteHiConga}ti{b1} :: tie
command.py: raw args
edit TI b1
which parameter? (i,t,b,r,p,f,o,a,n,x,s,d): r
current rhythm: pulseTriple, (constant, 4), (basketGen, randomPermutate,
(1,1,2,3)), (constant, 1), (constant, 0.75)
new value: l, ((4,1,1),(4,1,1),(4,2,1),(4,3,1),(4,5,1),(4,3,1)), rw
TI b1: parameter rhythm updated.

pi{auto-muteHiConga}ti{b1} :: tev r
compare parameters: rhythm
{name,value,}
a1                  pulseTriple, (constant, 4), (basketGen, randomPermutate,
                    (1,1,2,3)), (constant, 1), (constant, 0.75)
b1                  loop, ((4,1,+),(4,1,+),(4,2,+),(4,3,+),(4,5,+),(4,3,+)),
                    randomWalk
```

Notice that, as with all ParameterObjects, abbreviations can be used for argument strings. The user need only enter the string "l" to select the "loop" RhythmObject, and "rw" to select the randomWalk selection method.

To edit Texture a1, the user must first make a1 the active texture with TIo. In the following example, the user applies a zero-order Markov chain to generate pulses. The user fist consults the documentation for ParameterObject markovPulse. For more information about Markov transition strings (Ariza 2006), enter "help markov". After selecting and editing the Texture, the Rhythms are compared with TEv:

Example 4-28. Editing Rhythm ParameterObjects with TIe

```
pi{auto-muteHiConga}ti{b1} :: tpv markovp
Rhythm Generator ParameterObject
{name,documentation}
markovPulse         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}

pi{auto-muteHiConga}ti{b1} :: tio a1
TI a1 now active.

pi{auto-muteHiConga}ti{a1} :: tie
edit TI a1
which parameter? (i,t,b,r,p,f,o,a,n,x,s,d): r
current rhythm: pulseTriple, (constant, 4), (basketGen, randomPermutate,
(1,1,2,3)), (constant, 1), (constant, 0.75)
new value: mp, a{8,1,1}b{4,3,1}c{4,2,1}d{4,5,1}:{a=1|b=3|c=4|d=7}, (c,0)
TI a1: parameter rhythm updated.

pi{auto-muteHiConga}ti{a1} :: tev r
compare parameters: rhythm
{name,value,}
a1                  markovPulse,
                    a{8,1,1}b{4,3,1}c{4,2,1}d{4,5,1}:{a=1|b=3|c=4|d=7},
                    (constant, 0)
b1                  loop, ((4,1,+),(4,1,+),(4,2,+),(4,3,+),(4,5,+),(4,3,+)),
                    randomWalk
```

In the previous example, the user supplies four Pulses; each pulses is weighted such that the shortest, (8,1,1), is the least frequent (weight of 1), and the longest, (4,5,1), is the most frequent (weight of 7).

Using ELn, the current collection of Textures can be used to create an EventList, and ELh may be used to audition the results. (For more information on using ELn, see Section 2.5.) Each time an EventList is created, different sequences of rhythms will be generated: for Texture a1, these rhythms will be the result of a zero-order Markov chain; for Texture b1, these rhythms will be the result of a random walk on an ordered list of Pulses.

A final alternation can be made to the metric performance of these Textures. Using the TEe command, both Texture's bpm attribute can be altered to cause a gradual accelerando from 120

BPM to 300 BPM. In the following example, the user applies a wavePowerUp ParameterObject to the bpm attribute of both Textures by using the TEe command with complete command-line arguments:

Example 4-29. Editing BPM with TEe

```
pi{auto-muteHiConga}ti{a1} :: tee b wpu,t,20,0,2,120,300
TI a1: parameter bpm updated.
TI b1: parameter bpm updated.
```

4.11. Editing Instruments and Altering EventMode

A Texture's instrument can be edited like other Texture attributes. The instruments available for editing, just as when creating a Texture, are dependent on the active EventMode. To use instruments from another EventMode, the active EventMode must first be changed, and then the Texture may be assigned an instrument.

In the following example, the user changes the EventMode to csoundNative with EMo, examines the available instruments with EMi, and then assigns each Texture instrument 80:

Example 4-30. Changing EventMode and editing Texture instrument

```
pi{auto-muteHiConga}ti{a1} :: emo cn
EventMode mode set to: csoundNative.

pi{auto-muteHiConga}ti{a1} :: emi
csoundNative instruments:
{number,name}
   3      sineDrone
   4      sineUnitEnvelope
   5      sawDrone
   6      sawUnitEnvelope
   11     noiseWhite
   12     noisePitched
   13     noiseUnitEnvelope
   14     noiseTambourine
   15     noiseUnitEnvelopeBandpass
   16     noiseSahNoiseUnitEnvelope
   17     noiseSahNoiseUnitEnvelopeDistort
   20     fmBasic
   21     fmClarinet
   22     fmWoodDrum
   23     fmString
   30     samplerReverb
   31     samplerRaw
   32     samplerUnitEnvelope
   33     samplerUnitEnvelopeBandpass
   34     samplerUnitEnvelopeDistort
   35     samplerUnitEnvelopeParametric
   36     samplerSahNoiseUnitEnvelope
   40     vocodeNoiseSingle
   41     vocodeNoiseSingleGlissando
   42     vocodeNoiseQuadRemap
   43     vocodeNoiseQuadScale
   44     vocodeNoiseQuadScaleRemap
   45     vocodeNoiseOctScale
   46     vocodeNoiseOctScaleRemap
   47     vocodeNoiseBiOctScale
   48     vocodeNoiseTriOctScale
   50     guitarNylonNormal
   51     guitarNylonLegato
   52     guitarNylonHarmonic
   60     additiveBellBright
   61     additiveBellDark
   62     additiveBellClear
   70     synthRezzy
   71     synthWaveformVibrato
   72     synthVcoAudioEnvelopeSineQuad
   73     synthVcoAudioEnvelopeSquareQuad
   74     synthVcoDistort
   80     pluckTamHats
   81     pluckFormant
   82     pluckUnitEnvelope
   110    noiseAudioEnvelopeSineQuad
   111    noiseAudioEnvelopeSquareQuad
   130    samplerAudioEnvelopeSineQuad
   131    samplerAudioEnvelopeSquareQuad
   132    samplerAudioFileEnvelopeFilter
   133    samplerAudioFileEnvelopeFollow
   140    vocodeSineOctScale
   141    vocodeSineOctScaleRemap
   142    vocodeSineBiOctScale
   143    vocodeSineTriOctScale
   144    vocodeSineQuadOctScale
   145    vocodeSinePentOctScale
   146    vocodeSineHexOctScale
   230    samplerVarispeed
   231    samplerVarispeedAudioSine
   232    samplerVarispeedReverb
   233    samplerVarispeedDistort
   234    samplerVarispeedSahNoiseDistort
   240    vocodeVcoOctScale
   241    vocodeVcoOctScaleRemap

pi{auto-muteHiConga}ti{a1} :: tie i 80
baseTexture.py: WARNING: new Texture auxiliary value 2
TI a1: parameter instrument updated.

pi{auto-muteHiConga}ti{a1} :: tio b1
TI b1 now active.

pi{auto-muteHiConga}ti{b1} :: tie i 80
baseTexture.py: WARNING: new Texture auxiliary value 2
TI b1: parameter instrument updated.

pi{auto-muteHiConga}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: o, duration: 005.0--20.12
(i)nstrument        80 (csoundNative: pluckTamHats)
(t)ime range        05.0--20.0
(b)pm               wavePowerUp, time, (constant, 20), 0, 2, (constant, 120),
                    (constant, 300)
(r)hythm            loop, ((4,1,+),(4,1,+),(4,2,+),(4,3,+),(4,5,+),(4,3,+)),
                    randomWalk
(p)ath              auto-muteHiConga
                    (D4)
                    15.00(s)
local (f)ield       constant, 0
local (o)ctave      constant, 0
(a)mplitude         randomUniform, (constant, 0.6), (constant, 1)
pan(n)ing           wavePowerDown, event, (constant, 15), 0.25, 2.5, (constant,
                    0), (constant, 0.5)
au(x)iliary
      x0            cyclicGen, up, 0.1, 0.9, 0.1
      x1            cyclicGen, down, 800, 16000, 200
texture (s)tatic
      s0            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

Notice that, after editing the Texture, a warning is issued. This warning tells the user that additional auxiliary ParameterObjects have been added. As a Csound-based instrument, each event of instrument 80 can accept two additional synthesis parameters. When viewing a Texture with this instrument, as shown above, the auxiliary display shows two additional ParameterObjects, x0 and x1. To learn what these auxiliary ParameterObjects control, the command TIdoc ma be used:

Example 4-31. Examining Texture documentation with TIdoc

```
pi{auto-muteHiConga}ti{b1} :: tidoc
TI: b1, TM: LineGroove
(i)nstrument        80 (csoundNative: pluckTamHats)
(b)pm               (1) name, (2) stepString {'event', 'time'}, (3)
                    parameterObject {secPerCycle}, (4) phase, (5) exponent, (6)
                    min, (7) max
(r)hythm            (1) name, (2) pulseList {a list of Pulse notations}, (3)
                    selectionString {'randomChoice', 'randomWalk',
                    'randomPermutate', 'orderedCyclic',
                    'orderedCyclicRetrograde', 'orderedOscillate'}
local (f)ield       (1) name, (2) value
local (o)ctave      (1) name, (2) value
(a)mplitude         (1) name, (2) min, (3) max
pan(n)ing           (1) name, (2) stepString {'event', 'time'}, (3)
                    parameterObject {secPerCycle}, (4) phase, (5) exponent, (6)
                    min, (7) max
au(x)iliary
      x0            iparm (0-1)
                    (1) name, (2) directionString {'upDown', 'downUp', 'up',
                    'down'}, (3) min, (4) max, (5) increment
      x1            low-pass filter frequency
                    (1) name, (2) directionString {'upDown', 'downUp', 'up',
                    'down'}, (3) min, (4) max, (5) increment
texture (s)tatic
      s0            (1) name, (2) transpositionList, (3) timeDelay
      s1            (1) name, (2) selectionString {'randomChoice', 'randomWalk',
                    'randomPermutate', 'orderedCyclic',
                    'orderedCyclicRetrograde', 'orderedOscillate'}
      s2            (1) name, (2) level {'set', 'event'}
      s3            (1) name, (2) level {'set', 'event'}
texture (d)ynamic   none
```

Assuming that Csound is properly configured, a new set of EventLists can be created. As the user is now in EventMode csoundNative and has csoundNative textures, both a Csound score and a MIDI file are created. (See Section 2.6 for more information on working with Csound in athenaCL.) The user may render the Csound score with ELr, and then audition the results with the ELh command.

Example 4-32. Creating a new EventList with ELn

```
pi{auto-muteHiConga}ti{b1} :: eln
      EventList ath2010.07.03.18.40.11 complete:
/Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.bat
/Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.csd
/Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.mid
/Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.xml
audio rendering initiated: /Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.bat
EventList hear initiated: /Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.aif
EventList hear initiated: /Volumes/xdisc/_scratch/ath2010.07.03.18.40.11.mid
```

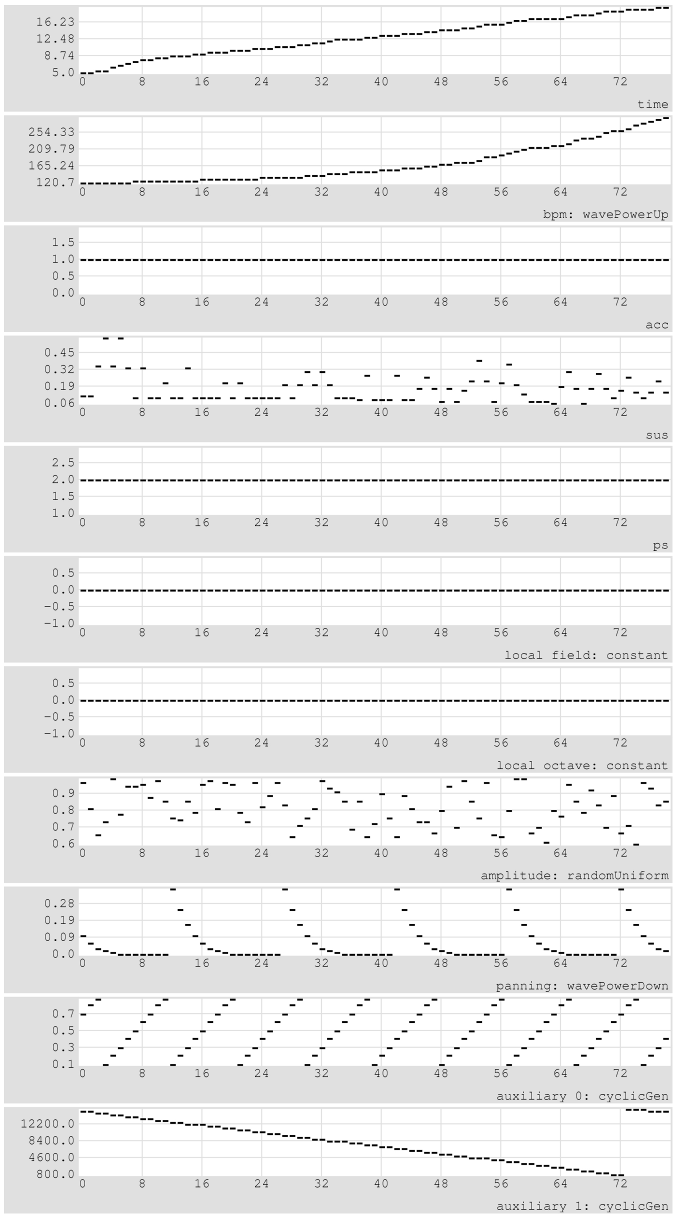
4.12. Displaying Texture Parameter Values

It is often useful to view the values produced by a Texture with a graphical diagram. The command TImap provides a multi-parameter display of all raw values input from ParameterObjects into the Texture. The values displayed by TImap are pre-TextureModule, meaning that they are the raw values produced by the ParameterObjects; the final parametric event values may be altered or completely changed by the Texture's internal processing (its TextureModule) to produce different arrangements of events. The TImap command thus only provides a partial representation of what a Texture produces.

To view a TImap display, the user's graphic preferences must be properly set (see Example 1-11 for more information). The command TImap displays the active Texture:

Example 4-33. Viewing a Texture with TImap

```
pi{auto-muteHiConga}ti{b1} :: timap
TImap (event-base, pre-TM) display complete.
```


16.23
12.48
8.74
5.0
0 8 16 24 32 40 48 56 64 72
time
254.33
209.79
165.24
120.7
bpm: wavePowerUp
1.5
1.0
0.5
0.0
acc
0.45
0.32
0.19
0.06
sus
2.5
2.0
1.5
1.0
ps
0.5
0.0
-0.5
-1.0
local field: constant
local octave: constant
0.9
0.8
0.7
0.6
amplitude: randomUniform
0.28
0.19
0.09
0.0
panning: wavePowerDown
0.7
0.5
0.3
0.1
auxiliary 0: cyclicGen
12200.0
8400.0
4600.0
800.0
auxiliary 1: cyclicGen

Chapter 5. Tutorial 5: Textures and Paths

This tutorial demonstrates basic use of Paths within Textures. This chapter is essential for understanding the use of Paths in algorithmic music production.

5.1. Path Linking and Pitch Formation Redundancy

Textures link to Paths. Said another way, a Texture contains a reference to a Path object stored in the AthenaObject. Numerous Textures can thus share the same Path; further, if a change is made to this Path, all Textures will reference the newly updated version of the Path.

Events generated by a Texture can derive pitch values from a sequence of many transformations. These transformations allow the user to work with Pitch materials in a wide variety of orientations and parametric specifications. One or more Textures may share a single Path to derive pitch class or pitch space pitch values. Each Texture has independent ParameterObject control of a local transposition (local field) and a local register position (local octave), and with most TextureModules this control can be configured to be applied once per event or once per Path set. Finally, each Texture has a modular Temperament object to provide microtonal and dynamic final pitch tuning. Ultimately, the TextureModule is responsible for interpreting this final pitch value into a linear, horizontal, or other event arrangement.

For example, a Texture may be linked to simple Path consisting of a single pitch. This pitch will serve as a referential pitch value for all pitch generation and transformation within the Texture. The Texture's local field and local octave controls could then be used to produce a diverse collection of Pitch values. Changing the single pitch of the Path would then provide global transposition of Texture-based pitch processes. Alternatively, a Path may specify a complex sequence of chord changes. Numerous Textures, linked to this single Path, could each apply different local octave settings to distinguish register, and could each apply different microtonal tunings with local field and Temperament settings.

5.2. Creating a Path with a Duration Fraction

First, the user creates a path consisting of three Multisets. As demonstrated in Chapter 3, there are many ways to create and edit a Path. In the following example, the user creates a new path named q1 by simply providing pitch space values using conventional note names. The path is the then viewed with the PIv command, and auditioned with the PIh command.

Example 5-1. Creating a Path with PIn

```
pi{}ti{} :: pin
name this PathInstance: q1
enter a pitch set, sieve, spectrum, or set-class: D2,G#3,A3,D3,E2,B2,A2
      SC 5-29A as (D2,G#3,A3,D3,E2,B2,A2)? (y, n, or cancel): y
      add another set? (y, n, or cancel): y
enter a pitch set, sieve, spectrum, or set-class: C4,C#4,F#3,G4,A3
      SC 5-19A as (C4,C#4,F#3,G4,A3)? (y, n, or cancel): y
      add another set? (y, n, or cancel): y
```

```
enter a pitch set, sieve, spectrum, or set-class: G#5,A4,D#4,E5
      SC 4-8 as (G#5,A4,D#4,E5)? (y, n, or cancel): y
      add another set? (y, n, or cancel): n
PI q1 added to PathInstances.

pi{q1}ti{} :: piv
PI: q1
psPath              -22,-4,-3,-10,-20,-13,-15  0,1,-6,7,-3       20,9,3,16
                    D2,G#3,A3,D3,E2,B2,A2      C4,C#4,F#3,G4,A3  G#5,A4,D#4,E5
pcsPath             2,8,9,2,4,11,9             0,1,6,7,9         8,9,3,4
scPath              5-29A                      5-19A             4-8
durFraction         1(33%)                     1(33%)            1(33%)
TI References: none.

pi{q1}ti{} :: pih
PI q1 hear with TM LineGroove complete.
(/Volumes/xdisc/_scratch/ath2010.07.03.19.05.21.mid)
```

As should be clear from the psPath display or the auditioned MIDI file, Path q1 covers a wide pitch range, from E2 to G#5. Notice also that the "durFraction" specifies that each Multiset in the Path has an equal duration weighting (1, or 33%). The durFraction of a Path is a means of providing a proportional temporal weighting to each Multiset in the Path. When a Texture interprets a Path, it partitions its duration into as many segments as there are Path Multisets, and each segment is given a duration proportional to the Path durFraction. The command PIdf can be used to alter a Path's duration weighting. The user must supply a list of values, either as percentages (floating point or integer) or simply as numeric weightings. In the following example, after calling PIdf, the command PIh is used to audition the results of an altered durFraction:

Example 5-2. Altering a Path's durFraction with PIdf

```
pi{q1}ti{} :: pidf
edit PI q1
enter a list of duration fractions: 8,5,3
PI q1 edited.

pi{q1}ti{} :: piv
PI: q1
psPath              -22,-4,-3,-10,-20,-13,-15  0,1,-6,7,-3       20,9,3,16
                    D2,G#3,A3,D3,E2,B2,A2      C4,C#4,F#3,G4,A3  G#5,A4,D#4,E5
pcsPath             2,8,9,2,4,11,9             0,1,6,7,9         8,9,3,4
scPath              5-29A                      5-19A             4-8
durFraction         8(50%)                     5(31%)            3(19%)
TI References: none.

pi{q1}ti{} :: pih
PI q1 hear with TM LineGroove complete.
(/Volumes/xdisc/_scratch/ath2010.07.03.19.08.09.mid)
```

The PIv display shows that the Multisets are weighted such that the first is given 50%, the second 31%, and the last 19%. The MIDI file created with PIh should confirm this distribution.

5.3. Setting EventMode and Creating a Texture

As explained in Chapter 4, the athenaCL EventMode determines what instruments are available for Texture creation. In the following example, the EventMode is set to midi, the TextureModule LiteralVertical is selected, a new Texture is created with instrument 0, and the Texture is displayed with TIv.

Example 5-3. Creating a Texture with TM LiteralVertical

```
pi{q1}ti{} :: emo midi
EventMode mode set to: midi.

pi{q1}ti{} :: tmls
TextureModules available:
{name,TIreferences}
   DroneArticulate    0
   DroneSustain       0
   HarmonicAssembly   0
   HarmonicShuffle    0
   InterpolateFill    0
   InterpolateLine    0
   IntervalExpansion  0
   LineCluster        0
 + LineGroove         0
   LiteralHorizontal  0
   LiteralVertical    0
   MonophonicOrnament 0
   TimeFill           0
   TimeSegment        0

pi{q1}ti{} :: tmo lv
TextureModule LiteralVertical now active.

pi{q1}ti{} :: tin a1 0
TI a1 created.

pi{q1}ti{a1} :: tiv
TI: a1, TM: LiteralVertical, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, duration: 000.0--19.94
(i)nstrument        0 (generalMidi: piano1)
(t)ime range        00.0--20.0
(b)pm               constant, 120
(r)hythm            pulseTriple, (constant, 4), (basketGen, randomPermutate,
                    (1,1,2,3)), (constant, 1), (constant, 0.75)
(p)ath              q1
                    (D2,G#3,A3,D3,E2,B2,A2),(C4,C#4,F#3,G4,A3),(G#5,A4,D#4,E5)
                    10.00(s), 6.25(s), 3.75(s)
local (f)ield       constant, 0
local (o)ctave      constant, 0
(a)mplitude         randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
pan(n)ing           constant, 0.5
au(x)iliary         none
texture (s)tatic
      s0            loopWithinSet, on
      s1            maxTimeOffset, 0.03
      s2            levelFieldPolyphonic, event
      s3            levelOctavePolyphonic, event
      s4            pathDurationFraction, on
texture (d)ynamic   none
```

Notice that the Texture's Path attribute is set to q1. In all cases, a Texture, when created, links to the active Path. The Path a Texture links to can be edited later if necessary. Notice also that the Path listing in the TIv display shows the pitches of the Path, as well as timings for each set: 10, 6.25, and 3.74 seconds respectively. These times are the result of the duration fraction applied to the Texture's duration.

To hear this Texture, create an EventList as explained in Section 2.5. After using the ELn command, the ELh command can be used to open the MIDI file. Notice that each chord lasts the appropriate duration fraction of the total twenty-second duration of the Texture.

A few small edits to this Texture will make it more interesting. In the following example, both the rhythm and amplitude are edited: the rhythm is given more Pulses and made to oscillate back and forth along the specified series; the amplitude randomly selects from a list of four amplitudes.

Example 5-4. Editing a Texture

```
pi{q1}ti{a1} :: tie r l, ((4,1,1), (4,1,1), (4,3,0), (2,3,1), (3,2,1), (3,2,1)), oo
TI a1: parameter rhythm updated.

pi{q1}ti{a1} :: tie a bg, rc, (.5,.6,.8,1)
TI a1: parameter amplitude updated.
```

Again, ELn and ELh can be used to create and audition the resulting musical structure.

5.4. PitchMode

PitchMode is a Texture attribute that controls the interpretation of the Path from inside a TextureInstance.

PitchMode determines if a Path is represented to the Texture as a pitch space set, a pitch class set, or set class. As a pitch space set, a Texture performs register information included in a Path. The set (1,11,24), performed as a pitch space set, would consist of a C-sharp, a B-natural a minor seventh above the lowest pitch, and C-natural an octave and major-seventh above the lowest pitch. The set (1,11,24) performed as a pitch-class set, would be interpreted as the set (1,11,0): register information is removed, while pitch class is retained. The set (1,11,24), performed as a set-class, would be interpreted as the set (0,1,2): register and pitch-class are removed, while the normal-form of the set-class is retained.

In the following example, a new Texture is created from TextureModule LineGroove. First, the TM must be selected with the TMo command. Next, a new Texture named b1 is created with the TIn command. The TImode command can be used to edit many Texture options. In this example, pitchMode is selected and "pcs," for pitch class space, is selected. Finally, the Texture is given a more interesting rhythm, by use of the Rhythm ParameterObject markovPulse, and is panned to the right with a constant value:

Example 5-5. Editing PitchMode of a TextureInstance

```
pi{q1}ti{a1} :: tmo lg
TextureModule LineGroove now active.

pi{q1}ti{a1} :: tin b1 0
TI b1 created.

pi{q1}ti{b1} :: timode
edit TI b1: Pitch, Polyphony, Silence, or PostMap Mode? (p, y, s, m): p
      current Pitch Mode: pitchSpace. enter new mode (sc, pcs, ps): pcs
Pitch Mode changed to pitchClass

pi{q1}ti{b1} :: tie
edit TI b1
which parameter? (i,t,b,r,p,f,o,a,n,x,s,d): r
current rhythm: pulseTriple, (constant, 4), (basketGen, randomPermutate,
(1,1,2,3)), (constant, 1), (constant, 0.75)
new value: mp, a{6,1,1}b{3,2,0}c{3,5,1}:{a=5|b=3|c=2}, (c,0)
TI b1: parameter rhythm updated.

pi{q1}ti{b1} :: tie n c,.9
TI b1: parameter panning updated.

pi{q1}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchClass, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, duration: 000.0--19.83
(i)nstrument        0 (generalMidi: piano1)
(t)ime range        00.0--20.0
(b)pm               constant, 120
(r)hythm            markovPulse, a{6,1,1}b{3,2,0}c{3,5,1}:{a=5|b=3|c=2},
                    (constant, 0)
(p)ath              q1
                    (D2,G#3,A3,D3,E2,B2,A2),(C4,C#4,F#3,G4,A3),(G#5,A4,D#4,E5)
                    10.00(s), 6.25(s), 3.75(s)
local (f)ield       constant, 0
local (o)ctave      constant, 0
(a)mplitude         randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
pan(n)ing           constant, 0.9
au(x)iliary         none
texture (s)tatic
      s0            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

Because Path q1 is still active, this new Texture is assigned the same Path as Texture a1. After setting the Texture's pitchMode to pitchClassSpace, however, Texture b1 will receive only pitch class values from Path q1: all register information, as performed in Texture a1, is stripped. By creating a new EventList with ELn and auditioning the results, it should be clear that both Textures share the same pitch information and duration weighting. Notice that the faster-moving single-note line Texture b1, however, stays within a single register. When a Texture is in pitchClassSpace Pitch mode, all pitches from a Path are interpreted within the octave from C4 to C5.

5.5. Editing Local Octave

With a Texture's local field and local octave controls, ParameterObjects can be used to alter the pitches derived from a Path. In most TextureModules, the transformation offered by field and octave control can be applied either once per Multiset, or once per event. This is set by editing the TextureStatic options levelField and levelOctave.

In the following example, the local octave attribute of Texture b1 is edited such that octaves are chosen in order from a list of possibilities, creating a sequence of octave transpositions. An octave value of 0 means no transposition; an octave of -2 means a transposition two octaves down.

Example 5-6. Editing Local Octave

```
pi{q1}ti{b1} :: tie
command.py: raw args
edit TI b1
which parameter? (i,t,b,r,p,f,o,a,n,x,s,d): o
current local octave: constant, 0
new value: bg, oc, [-3,-2,2,-1]
TI b1: parameter local octave updated.

pi{q1}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchClass, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, duration: 000.0--19.83
(i)nstrument        0 (generalMidi: piano1)
(t)ime range        00.0--20.0
(b)pm               constant, 120
(r)hythm            markovPulse, a{6,1,1}b{3,2,0}c{3,5,1}:{a=5|b=3|c=2},
                    (constant, 0)
(p)ath              q1
                    (D2,G#3,A3,D3,E2,B2,A2),(C4,C#4,F#3,G4,A3),(G#5,A4,D#4,E5)
                    10.00(s), 6.25(s), 3.75(s)
local (f)ield       constant, 0
local (o)ctave      basketGen, orderedCyclic, (-3,-2,2,-1)
(a)mplitude         randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
pan(n)ing           constant, 0.9
au(x)iliary         none
texture (s)tatic
      s0            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

Listening to the results of the previous edit (with ELn and ELh), it should be clear that a new octave is applied to each event of Texture b1, creating an regular oscillation of register independent of Path Multiset.

Alternatively, the user may desire local octave and field controls to only be applied once per Multiset. This option can be set for TextureModule LineGroove by editing the TextureStatic parameter "levelOctaveMonophonic." In the following example, the user examines the documentation of ParameterObject levelOctaveMonophonic, and a copy of Texture b1 is created named b2. Next, this Texture's panning is edited, and then the TextureStatic option levelOctaveMonophonic is changed from "event" to "set":

Example 5-7. Editing TextureStatic

```
pi{q1}ti{b1} :: tpv leveloctave
Texture Static ParameterObject
{name,documentation}
levelOctaveMonophonic 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'}
levelOctavePolyphonic 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'}

pi{q1}ti{b1} :: ticp b1 b2
TextureInstance b2 created.

pi{q1}ti{b2} :: tie
edit TI b2
which parameter? (i,t,b,r,p,f,o,a,n,x,s,d): s
select a texture static parameter to edit, from s0 to s3: 3
current value for s3: event
new value: set
TI b2: parameter texture static updated.

pi{q1}ti{b2} :: tie n c,.1
TI b2: parameter panning updated.
```

Listening to a new EventList created with these three Textures (with ELn and ELh), it should be clear that all pitch information is synchronized by use of a common Path. In the case of Texture a1, the pitches are taken directly from the Path with register. In the case of Texture b1 (right channel), the Path pitches, without register, are transposed into various registers for each event. In the case of Texture b2 (left channel), the Path pitches, also without register, are transposed into various registers only once per Multiset.

5.6. Editing Local Field and Temperament

Within athenaCL, any pitch can be tuned to microtonal specifications, allowing the user to apply non-equal tempered frequencies to each pitch in either a fixed relationship or a dynamic, algorithmically generated manner. Within athenaCL Textures there are two ways to provide microtonal tunings. First, pitches can be transposed and tuned with any ParameterObject by using the Texture local field attribute. Each integer represents a half-step transposition, and floating point values can provide any detail of microtonal specification. Second, each Texture can have a different Temperament, or tuning system based on either pitch class, pitch space, or algorithmic specification. The command TTls allows the user to list the available TextureTemperaments.

Example 5-8. Listing all TextureTemperaments

```
pi{q1}ti{b2} :: ttls
TextureTemperaments available for TI b2:
{name,tunning}
 + TwelveEqual
   Pythagorean
   Just
   MeanTone
   Split24Upper
   Split24Lower
   Interleave24Even
   Interleave24Odd
   NoiseLight
   NoiseMedium
   NoiseHeavy
```

The temperament "TwelveEqual" is the active Temperament for current Texture b2. This temperament produces equal-tempered frequencies. To select a different temperament for the active Texture, enter the command TTo. In the example below the user selects the temperament NoiseLight for Textures b2 and Texture b1, and then selects the temperament NoiseMedium for Texture a1. In the last case, two command are given on the some command line. As is the UNIX convention, the commands and arguments are separated by a semicolon:

Example 5-9. Selecting Texture Temperament with TTo

```
pi{q1}ti{b2} :: tto
select a TextureTemperament for TI b2: (name or number 1-11): nl
TT NoiseLight now active for TI b2.

pi{q1}ti{b2} :: tio b1
TI b1 now active.

pi{q1}ti{b1} :: tto nl
TT NoiseLight now active for TI b1.

pi{q1}ti{b1} :: tio a1; tto nm
TI a1 now active.

TT NoiseMedium now active for TI a1.
```

Not all EventOutputs can perform microtones. MIDI files, for example, cannot store microtonal specifications of pitch. Though such pitches will be generated within athenaCL, they will be rounded when written to the MIDI file. EventOutputs for Csound, however, can handle microtones.

Chapter 6. Tutorial 6: Textures and Clones

This tutorial demonstrates basic Clone creation, configuration, and deployment in musical structures. Clones provide an additional layer of algorithmic music production, processing the literal output of Textures.

6.1. Introduction to Clones

A TextureClone (or a Clone or TC) is a musical part made from transformations of the exact events produced by a single Texture. Said another way, a Clone is not a copy of a Texture, but a transformed copy of the events produced by a Texture. Textures are not static entities, but algorithmic instructions that are "performed" each time an EventList is created. In order to capture and process the events of a single Texture, one or more Clones can be created in association with a single Texture.

Clones use Filter ParameterObjects to parametrically modify events produced from the parent Texture. Clones can be used to achieve a variety of musical structures. An echo is a simple example: by shifting the start time of events, a Clone can be used to create a time-shifted duplicate of a Texture's events. Clones can be used with a Texture to produce transformed motivic quotations of events, or can be used to thicken or harmonize a Texture with itself, for instance by filtering event pitch values.

Clones are also capable of non-parametric transformations that use CloneStatic ParameterObjects. For example a Clone, using a retrograde transformation, can reverse the events of a Texture.

6.2. Creating and Editing Clones

First, using EventMode midi and instrument 0, a Texture with a descending melodic arc will be created. The Texture's time range is set from 0 to 6. The Texture's rhythm employs the ParameterObject convertSecond and uses a standard Generator ParameterObject to create raw duration values in seconds. Finally, This Texture, using a Path only as a reference pitch, employs the Texture's local field to provide harmonic shape.

Example 6-1. Creating a Texture

```
pi{}ti{} :: emo m
EventMode mode set to: midi.

pi{}ti{} :: tin a1 0
TI a1 created.

pi{auto}ti{a1} :: tie t 0,6
TI a1: parameter time range updated.

pi{auto}ti{a1} :: tie r cs,(wpd,e,16,2,0,.6,.02)
TI a1: parameter rhythm updated.

pi{auto}ti{a1} :: tie f wpd,e,16,2,0,12,-24
TI a1: parameter local field updated.
```

```
pi{auto}ti{a1} :: tiv
TI: a1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, duration: 00.0--6.41
(i)nstrument        0 (generalMidi: piano1)
(t)ime range        0.0--6.0
(b)pm               constant, 120
(r)hythm            convertSecond, (wavePowerDown, event, (constant, 16), 2, 0,
                    (constant, 0.6), (constant, 0.02))
(p)ath              auto
                    (C4)
                    6.00(s)
local (f)ield       wavePowerDown, event, (constant, 16), 2, 0, (constant, 12),
                    (constant, -24)
local (o)ctave      constant, 0
(a)mplitude         randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
pan(n)ing           constant, 0.5
au(x)iliary         none
texture (s)tatic
      s0            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

After creating a Texture, a Clone can be created with the command TCn, for TextureClone New. The user is prompted to enter the name of the new Clone. By default, the Filter ParameterObject filterAdd is applied to the start time of all events with a duration equal to one Pulse. A Clone can be displayed with the TCv command. After displaying the Clone, the user examines the documentation for ParameterObject filterAdd:

Example 6-2. Creating and Viewing a Clone with TCn and TCv

```
pi{auto}ti{a1} :: tcn
name this TextureClone: w1
TC w1 created.

pi{auto}ti{a1} :: tcv
TC: w1, TI: a1
      status: +, duration: 00.5--6.91
(t)ime              filterAdd, (loop, ((1,1,+)), orderedCyclic)
s(u)stain           bypass
a(c)cent            bypass
local (f)ield       bypass
local (o)ctave      bypass
(a)mplitude         bypass
pan(n)ing           bypass
au(x)iliary         none
clone (s)tatic
      s0            timeReferenceSource, textureTime
      s1            retrogradeMethodToggle, off
```

The Filter ParameterObject bypass is the default for most Clone attributes. This ParameterObject simply passes values through to the Clone unaltered.

Upon creating a new EventList and auditioning the results (with ELn and ELh, see Section 2.5 for more information), the descending melodic line of a1 can be heard echoed by Clone w1. In the following example, another Clone is created called w2. This Clone is then edited to have a time value that, rather than shifted by a constant, is scaled by a value that oscillates between 1 and 2. The Clone's local field filter is also set to transpose the Texture's pitches seven half-steps down. The procedure for editing Clone ParameterObjects is similar to that for editing Textures, except for that only Filter ParameterObjects can be provided.

Example 6-3. Editing a Clone with TCe

```
pi{auto}ti{a1} :: tcn w2
TC w2 created.

pi{auto}ti{a1} :: tpv fma
Filter ParameterObject
{name,documentation}
FilterMultiplyAnchor 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}

pi{auto}ti{a1} :: tce t fma, l, (ws, e, 8, 0, 1, 2)
TC w2: parameter time updated.

pi{auto}ti{a1} :: tce f fa,(c,-7)
TC w2: parameter local field updated.

pi{auto}ti{a1} :: tcv
TC: w2, TI: a1
      status: +, duration: 000.0--11.41
(t)ime              filterMultiplyAnchor, lower, (waveSine, event, (constant,
                    8), 0, (constant, 1), (constant, 2))
s(u)stain           bypass
a(c)cent            bypass
local (f)ield       filterAdd, (constant, -7)
local (o)ctave      bypass
(a)mplitude         bypass
pan(n)ing           bypass
au(x)iliary         none
clone (s)tatic
      s0            timeReferenceSource, textureTime
      s1            retrogradeMethodToggle, off
```

As with Textures and other objects in athenaCL, Clones can be listed with the TCls command, and the active Clone can be selected with the TCo command. Further, upon examining the parent Texture with TIls, notice that two Clones are now displayed under the TC heading:

Example 6-4. Listing and Selecting Clones with TCls and TCo

```
pi{auto}ti{a1} :: tcls
TextureClones of TI a1
```

```
{name,status,duration}
   w1              + 00.5--6.91
 + w2              + 000.0--11.41

pi{auto}ti{a1} :: tco w1
TC w1 of TI a1 now active.

pi{auto}ti{a1} :: tils
TextureInstances available:
{name,status,TM,PI,instrument,time,TC}
 + a1              + LineGroove  auto        0   0.0--6.0   2
```

Clones features special transformations selected by CloneStatic ParameterObjects. In the following example, a new Clone is created named w3. The CloneStatic ParameterObject retrogradeMethodToggle is set to timeInverse, causing the Clone to create a retrograde presentation of the Texture's events. Additionally, the Clone's time attribute is set with a filterMultuplyAnchor ParameterObject and the Clone's field attributes is set with a filterAdd ParameterObject:

Example 6-5. Creating and Editing Clones

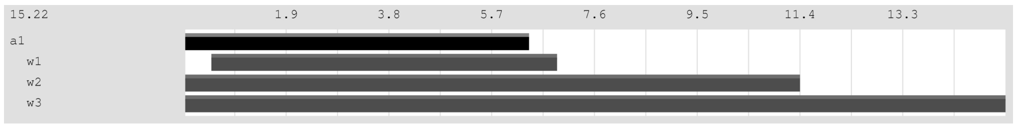
```
pi{auto}ti{a1} :: tpv retrograde
Clone Static ParameterObject
{name,documentation}
retrogradeMethodToggle retrogradeMethodToggle, name
                       Description: Selects type of retrograde transformation
                       applied to Texture events. Arguments: (1) name, (2) name
                       {'timeInverse', 'eventInverse', 'off'}

pi{auto}ti{a1} :: tcn w3
TC w3 created.

pi{auto}ti{a1} :: tce
edit TC a1
which parameter? (t,u,c,f,o,a,n,x,s): s
select a clone static parameter to edit, from s0 to s1: 1
current value for c1: off
new value: timeinverse
TC w3: parameter clone static updated.

pi{auto}ti{a1} :: tce t fma,l,(c,2.5)
TC w3: parameter time updated.

pi{auto}ti{a1} :: tce f fa,(c,7)
TC w3: parameter local field updated.
```

The TEmap command displays all Textures as well as all Texture Clones. Texture Clones appear under their parent Texture. Textures and Clones, further, can be muted independently.

Example 6-6. Viewing Textures and Clones with TEmap

```
pi{auto}ti{a1} :: temap
TextureEnsemble Map:
15.22s             |      .      |      .      |      .      |      .      |
a1                 _________________________
      w3           ............................................................
```

```
w2                .............................................
w1                  .........................
```



Chapter 7. Tutorial 7: Scripting athenaCL in Python

This tutorial demonstrates some of the ways the athenaCL system can be automated, scripted, and used from within the Python programming language.

7.1. Creating an athenaCL Interpreter within Python

Within a Python interpreter or a Python script on any platform, one or more instances of the athenaCL Interpreter can be created and programmatically controlled. Programming a sequence of athenaCL commands via a Python script provides maximal control and flexibility in using athenaCL. Loops, external procedures, and a variety of programming designs can be combined with the high-level syntax of the athenaCL command line. Furthermore, command sequences can be stored, edited, and developed.

The cmd() method of the athenaCL Interpreter can be used to be pass strings or Python data structures. The cmd() method will raise an exception on error. The following example creates an athenaCL Interpter instance named ath and sends it a number of commands to generate a drum beat.

Example 7-1. An athenaCL Interpreter in Python

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
ath.cmd('emo mp')
ath.cmd('tmo lg')

ath.cmd('tin a1 36')
ath.cmd('tie r l,((4,3,1),(4,3,0),(4,2,1)),rc')

ath.cmd('tin b1 37')
ath.cmd('tie r l,((4,6,1),(4,1,1),(4,3,1)),rc')

ath.cmd('tin c1 53')
ath.cmd('tie r l,((4,1,1),(4,1,1),(4,6,0)),rw')

ath.cmd('tee a bg,rc,(.5,.7,.75,.8,1)')
ath.cmd('tee b ws,t,4,0,122,118')

ath.cmd('eln; elh')
```

For advanced and/or extended work with athenaCL, automating command string execution is highly recommended. Included with the athenaCL distribution is over 30 Python files demonstrating fundamental concepts of working with the system. These files can be found in the demo directory and also in Appendix F.

7.2. Creating athenaCL Generator ParameterObjects within Python

Components of the athenaCL system can be used in isolation as resources within Python. Generator ParameterObjects offer particularly useful resources for a range of generative activities.

To create a Generator ParameterObject, a Python list of ParameterObject arguments must be passed to the factory() function of the parameter module. This list of arguments must provide proper data objects for each argument.

The returned ParameterObject instance has many useful attributes and methods. The doc attribute provides the ParameterObject documentation string. The __str__ method, accessed with the built-in str() function, returns the complete formatted argument string. The __call__ method, accessed by calling the instance name, takes a single argument and returns the next value, or the value at the specified argument time value.

Example 7-2. Creating a Generator ParameterObject

```
>>> from athenaCL.libATH.libPmtr import parameter
>>>  po = parameter.factory(['ws','t',6,0,-1,1])
>>> str(po)
'waveSine, time, (constant, 6), 0, (constant, -1), (constant, 1)'
>>> po.doc
'Provides sinusoid oscillation between 0 and 1 at a rate given in either time or
events per period. This value is scaled within the range designated by min and max;
min and max may be specified with ParameterObjects. Depending on the stepString
argument, the period rate (frequency) may be specified in spc (seconds per cycle) or
eps (events per cycle). 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.'
>>> po(1)
0.8660254037844386
>>> po(5)
-0.86602540378443904
```

7.3. Creating athenaCL Generator ParameterObjects within Csound

Within Csound 5.0 orchestras, Python scripts can be written and objects from Python libraries can be instantiated and processed. Numerous Generator ParameterObjects and other athenaCL objects can be created, modified, and called from within Csound instruments. For complete examples, see the detailed articles on the topic (Ariza 2008, 2009b).

Appendix A. Installation Instructions (readme.txt)

```
athenaCL Copyright (c) 2000-2010 Christopher Ariza and others.
athenaCL is free software, distributed under the GNU General Public License.

www.athenacl.org

====================================================================
athenaCL 2.0.0a15
7 July 2010
This document contains the following information:

I. Platform Dependencies
II. Software Dependencies
IIIa. Quick Start Distributions
IIIb. Quick Start Installers
IVa. Installation
IVb. athenaCL via Command Line Interface
IVc. athenaCL via IDLE
IVd. athenaCL via Python Prompt
V. Documentation
VI. Contact Information
VII. Credits and Acknowledgments

====================================================================
I. PLATFORM DEPENDENCIES:

athenaCL is distributed as executable cross-platform source-code. Platform-
specific distributions and installers are provided for convenience. Make sure
you have downloaded the correct archive for your platform:

    Distributions:

Python EGG
http://www.flexatone.net/athenaCL/athenaCL.egg

unix (GNU/Linux, BSD), Macintosh MacOS X
http://www.flexatone.net/athenaCL/athenaCL.tar.gz

Windows (any)
http://www.flexatone.net/athenaCL/athenaCL.exe


====================================================================
II. SOFTWARE DEPENDENCIES:

athenaCL requires Python 2.5 to 2.6. Python 3.0 and better is not yet supported.
There is no athenaCL binary: athenaCL interactive sessions run inside a Python
interpreter. Python is free and runs on every platform. No additional software
is required for basic athenaCL operation. Download Python here:
http://www.python.org/download

athenaCL produces both Csound and MIDI scores. Csound 5 is recommended; Csound
4.16 or better is required to render Csound scores. Csound is free and runs on
every platform. Download Csound here:
http://www.csounds.com

athenaCL produces images with various Python-based graphic output systems. These
output systems include the Python TkInter GUI library and the Python Image
Library (PIL), and may require additional Python software. Most Python
distributions include TkInter (MacOS systems may require additional
configuration); PIL is easily added to Python. Download PIL here:
```

```
http://www.pythonware.com/products/pil/

=====================================================================
IIIa. QUICK-START DISTRIBUTIONS:

All Platforms
    1. install Python 2.6
    2. decompress athenaCL distribution and place wherever desired

UNIX, Command Line Environments, Macintosh MacOS X:
    3. % python setup.py
    4. % python athenacl.py

For more information and additional installation options, see below.

=====================================================================
IIIb. QUICK-START INSTALLERS:

Python Prompt
    1. double click the installer and follow the instructions
    2. start Python
    3. >>> import athenaCL.athenacl

Windows Installer (exe)
    1. double click the .exe file and follow the instructions
    2. start python.exe
    3. >>> import athenaCL.athenacl

For more information and additional installation options, see below.

=====================================================================
IVa. INSTALLATION:

Two installation methods are available: (1) placing the athenaCL directory
wherever desired, or (2) installing the athenaCL source into the Python library
with the Python Distribution Utilities (distutils). Both permit using athenaCL
as an interactive application and as a library imported in Python.

Installing athenaCL consist of running the file "setup.py", a script that
performs installation procedures.

The setup.py script can take arguments to perform optional installation
procedures. (1) the "tool" argument, on UNIX and MacOS X systems, will install a
command-line utility launcher, "athenacl," as well as a corresponding man page.
(2) the "install" argument, on all platforms, will perform a Python distutils
installation into the Python site-packages directory. (3) the "uninstall" option
will remove all athenaCL installation files and directories.

=====================================================================
IVb. athenaCL VIA COMMAND LINE INTERFACE (CLI):

installing:
    1. decompress athenaCL
    2. place athenaCL directory wherever you like
    3. enter the athenaCL directory
    4. % python setup.py

or, to install the "athenacl" launcher and the athenaCL man page:
    4. % python setup.py tool

or, to perform a distutils installation
    4. % python setup.py install

launching from the command line interface:
    5. % python athenacl.py
```

```
launching with the athenaCL tool:
    5. % /usr/local/bin/athenacl

launching with the athenaCL tool and /usr/local/bin in PATH:
    5. % athenacl

====================================================================
IVd. athenaCL VIA IDLE:

installing:
    1. decompress athenaCL
    2. place athenaCL directory wherever you like
    3. enter the athenaCL directory
    4. double-click "setup.py"

launching on Windows:
    5. double-click "athenacl.py"
    6. enter "y" when asked to start athenaCL in IDLE

launching from the command line interface:
    5. % python athenacl.py -s idle

====================================================================
IVd. athenaCL VIA PYTHON PROMPT

If the athenaCL setup.py script has been successfully completed, Python should
already by aware of the location of the current athenaCL installation. If the
athenaCL setup.py script has not been properly run, the directory containing
athenaCL must be manually added to the Python sys.path:
(if the athenaCL directory is located in the directory "/src")
    1. >>> import sys
    2. >>> sys.path.append('/src')

launching:
    3. >>> import athenaCL.athenacl

====================================================================
V. DOCUMENTATION:

For complete documentation, tutorials, and reference, see the athenaCL Tutorial
Manual:
www.flexatone.net/athenaDocs/

====================================================================
VI. CONTACT INFORMATION:

Send questions, comments, and bug reports to:
athenacl@googlegroups.com
athenaCL development is hosted at GoogleCode:
http://code.google.com/p/athenacl/

====================================================================
VII. CREDITS and ACKNOWLEDGMENTS:

athenaCL was created and is maintained by Christopher Ariza. Numerous generator
ParameterObjects based in part on the Object-oriented Music Definition
Environment (OMDE/pmask), Copyright 2000-2001 Maurizio Umberto Puxemdu; Cmask
was created by Andre Bartetzki. The Command Line Interpreter is based in part on
cmd.py; the module textwrap.py is by Greg Ward; both are distributed with
Python, Copyright 2001-2003 Python Software Foundation. The fractional noise
implementation in dice.py, Audacity spectrum importing, and dynamic
ParameterObject boundaries are based in part on implementations by Paul Berg.
The module genetic.py is based in part on code by Robert Rowe. The module
midiTools.py is based in part on code by Bob van der Poel. The module chaos.py
```

is based in part on code by Hans Mikelson. The module permutate.py is based in part on code by Ulrich Hoffman. Pitch class set names provided in part by Larry Solomon. The Rabin-Miller Primality Test is based in part on an implementation by Stephen Krenzel. The mpkg installer is generated with py2app (bdist_mpkg) by Bob Ippolito. Python language testing done with PyChecker (by Neal Norwitz Copyright 2000-2001 MetaSlash Inc.) and pyflakes (by Phil Frost Copyright 2005 Divmod Inc.). Thanks to the following people for suggestions and feedback: Paul Berg, Per Bergqvist, Marc Demers, Ryan Dorin, Elizabeth Hoffman, Anthony Kozar, Paula Matthusen, Robert Rowe, Jonathan Saggau, and Jesse Sklar. Thanks also to the many users who have submitted anonymous bug-reports.

Appendix B. Command Reference

B.1. AthenaHistory Commands

B.1.1. AH

AthenaHistory: Commands: Displays a list of all AthenaHistory commands.

B.1.2. AHexe

AHexe: AthenaHistory: Execute: Execute a command or a command range within the current history.

B.1.3. AHls

AHls: AthenaHistory: List: Displays a listing of the current history.

B.1.4. AHrm

AHrm: AthenaHistory: Remove: Deletes the stored command history.

B.2. AthenaObject Commands

B.2.1. AO

AthenaObject: Commands: Displays a list of all AthenaObject commands.

B.2.2. AOals

AOals: AthenaObject: Attribute List: Displays raw attributes of the current AthenaObject.

B.2.3. AOl

AOl: AthenaObject: Load: Load an athenaCL XML AthenaObject. Loading an AthenaObject will overwrite any objects in the current AthenaObject.

B.2.4. AOmg

AOmg: AthenaObject: Merge: Merges a selected XML AthenaObject with the current AthenaObject.

B.2.5. AOrm

AOrm: AthenaObject: Remove: Reinitialize the AthenaObject, destroying all Paths, Textures, and Clones.

B.2.6. AOw

AOw: AthenaObject: Save: Saves an AthenaObject file, containing all Paths, Textures, Clones, and environment settings.

B.3. AthenaPreferences Commands

B.3.1. AP

AthenaPreferences: Commands: Displays a list of all AthenaPreferences commands.

B.3.2. APa

APa: AthenaPreferences: Audio: Set audio preferences.

B.3.3. APcurs

APcurs: AthenaPreferences: Cursor: Toggle between showing or hiding the cursor prompt tool.

B.3.4. APdir

APdir: AthenaPreferences: Directories: Lets the user select or enter directories necessary for writing and searching files. Directories that can be entered are the "scratch" directory and the "user audio". The scratch directory is used for writing temporary files with automatically-generated file names. Commands such as SCh, PIh, and those that produce graphics (depending on format settings specified with APgfx) use this directory. The user audio directory is used within ParameterObjects that search for files. With such ParameterObjects, the user can specify any file within the specified directory simply by name. To find the the file's complete file path, all directories are recursively searched in both the user audio and the athenaCL/audio directories. Directories named "_exclude" will not be searched. If files in different nested directories do not have unique file names, correct file paths may not be found.

B.3.5. APdlg

APdlg: AthenaPreferences: Dialogs: Toggle between different dialog modes. Not all modes are available on every platform or Python installation. The "text" dialog mode works without a GUI, and is thus available on all platforms and Python installations.

B.3.6. APea

APea: AthenaPreferences: External Applications: Set the file path to external utility applications used by athenaCL. External applications can be set for Csound (csoundCommand) and for handling various media files: midi (midiPlayer), audio (audioPlayer), text (textReader), image (imageViewer), and postscript (psViewer).

B.3.7. APgfx

APgfx: AthenaPreferences: Graphics: Toggle between different graphic output formats. All modes may not be available on every platform or Python installation. This command uses the active graphic output format; this can be selected with the "APgfx" command. Output in "tk" requires the Python Tkinter GUI installation; output in "png" and "jpg" requires the Python Imaging Library (PIL) library installation; output in "eps" and "text" do not require any additional software or configuration.

B.3.8. APr

APr: AthenaPreferences: Refresh: When refresh mode is active, every time a Texture or Clone is edited, a new event list is calculated in order to test ParameterObject compatibility and to find absolute time range. When refresh mode is inactive, editing Textures and Clones does not test event list production, and is thus significantly faster.

B.3.9. APwid

APwid: AthenaPreferences: Width: Manually set the number of characters displayed per line during an athenaCL session. Use of this preference is only necessary on platforms that do not provide a full-featured terminal envrionment.

B.4. AthenaUtility Commands

B.4.1. AU

AthenaUtility: Commands: Displays a list of all AthenaUtility commands.

B.4.2. AUbeat

AUbeat: AthenaUtility: Beat: Simple tool to calculate the duration of a beat in BPM.

B.4.3. AUbug

AUbug: AthenaUtility: Bug: causes a bug to test the error reporting system.

B.4.4. AUca

AUca: AthenaUtility: Cellular Automata: Utility for producing visual representations of values generated by various one-dimensional cellular automata.

B.4.5. AUdoc

AUdoc: AthenaUtility: Documentation: Opens the athenaCL documentation in a web browser. Attempts to load documentation from a local copy; if this fails, the on-line version is loaded.

B.4.6. AUlog

AUlog: AthenaUtility: Log: If available, opens the athenacl-log file used to store error messages.

B.4.7. AUma

AUma: AthenaUtility: Markov Analysis: Given a desired maximum order, this command analyzes the the provided sequence of any space delimited values and returns a Markov transition string.

B.4.8. AUmg

AUmg: AthenaUtility: Markov Generator: Given a properly formated Markov transition string, this command generates a number of values as specified by the count argument. Markov transition strings are entered using symbolic definitions and incomplete n-order weight specifications. The complete transition string consists of two parts: symbol definition and weights. Symbols are defined with alphabetic variable names, such as "a" or "b"; symbols may be numbers, strings, or other objects. Key and value pairs are notated as such: name{symbol}. Weights may be give in integers or floating point values. All transitions not specified are assumed to have equal weights. Weights are specified with key and value pairs notated as such: transition{name=weight | name=weight}. The ":" character is used as the zero-order weight key. Higher order weight keys are specified using the defined variable names separated by ":" characters. Weight values are given with the variable name followed by an "=" and the desired weight. Multiple weights are separated by the "|" character. All weights not specified, within a defined transition, are assumed to be zero. For example, the following string defines three variable names for the values .2, 5, and 8 and provides a zero order weight for b at 50%, a at 25%, and c at 25%: a{.2}b{5}c{8} :{a=1|b=2|c=1}. N-order weights can be included in a transition string. Thus, the following string adds first and second order weights to the same symbol definitions: a{.2}b{5}c{8} :{a=1|b=2|c=1} a:{c=2|a=1} c:{b=1} a:a:{a=3|b=9} c:b:{a=2|b=7|c=4}. For greater generality, weight keys may employ limited single-operator regular expressions within transitions. Operators permitted are "*" (to match all names), "-" (to not match a single name), and "|" (to match any number of names). For example, a:*:{a=3|b=9} will match "a" followed by any name; a:-b:{a=3|b=9} will match "a" followed by any name that is not "b"; a:b|c:{a=3|b=9} will match "a" followed by either "b" or "c".

B.4.9. AUpc

AUpc: AthenaUtility: Pitch Converter: Enter a pitch, pitch name, or frequency value to display the pitch converted to all formats. Pitches may be specified by letter name (psName), pitch space (psReal), pitch class, MIDI note number, or frequency. Pitch letter names may be specified as follows: a sharp is represented as "#"; a flat is represented as "$"; a quarter sharp is represented as "~"; multiple sharps, quarter sharps, and flats are valid. Octave numbers (where middle-C is C4) can be used with pitch letter names to provide register. Pitch space values (as well as pitch class) place C4 at 0.0. MIDI note numbers place C4 at 60. Numerical representations may encode microtones with additional decimal places. MIDI note-numbers and frequency values must contain the appropriate unit as a string ("m" or "hz").

B.4.10. AUsys

AUsys: AthenaUtility: System: Displays a list of all athenaCL properties and their current status.

B.4.11. AUup

AUup: AthenaUtility: Update: Checks on-line to see if a new version of athenaCL is available; if so, the athenaCL download page will be opened in a web browser.

B.5. EventList Commands

B.5.1. EL

EventList: Commands: Displays a list of all EventList commands.

B.5.2. ELauto

ELauto: EventList: Auto Render Control: Turn on or off auto rendering, causing athenaCL to automatically render (ELr) and hear (ELh) whenever an event list is created with ELn.

B.5.3. ELh

ELh: EventList: Hear: If possible, opens and presents to the user the last audible EventList output (audio file, MIDI file) created in the current session.

B.5.4. ELn

ELn: EventList: New: Create a new event list, in whatever formats are specified within the active EventMode and EventOutput. Generates new events for all Textures and Clones that are not muted. Specific output formats are determined by the active EventMode (EMo) and selected output formats (EOo).

B.5.5. ELr

ELr: EventList: Render: Renders the last event list created in the current session with the Csound application specified by APea.

B.5.6. ELv

ELv: EventList: View: Opens the last event list created in the current session as a text document.

B.5.7. ELw

ELw: EventList: Save: Write event lists stored in Textures and Clones, in whatever formats specified within the active EventMode and EventOutput; new event lists are not generated, and output will always be identical.

B.6. EventMode Commands

B.6.1. EM

EventMode: Commands: Displays a list of all EventMode commands.

B.6.2. EMi

EMi: EventMode: Instruments: Displays a list of all instruments available as defined within the active EventMode. The instrument assigned to a Texture determines the number of auxiliary parameters and the default values of these parameters.

B.6.3. EMls

EMls: EventMode: List: Displays a list of available EventModes.

B.6.4. EMo

EMo: EventMode: Select: Select an EventMode. EventModes determine what instruments are available for Textures, default auxiliary parameters for Textures, and the final output format of created event lists.

B.6.5. EMv

EMv: EventMode: View: Displays documentation for the active EventMode. Based on EventMode and selected EventOutputs, documentation for each active OutputEngine used to process events is displayed.

B.7. EventOutput Commands

B.7.1. EO

EventOutput: Commands: Displays a list of all EventOutput commands.

B.7.2. EOls

EOls: EventOutput: List: List all available EventOutput formats.

B.7.3. EOo

EOo: EventOutput: Select: Adds a possible output format to be produced when an event list is created. Possible formats are listed with EOls.

B.7.4. EOrm

EOrm: EventOutput: Remove: Removes a possible output format to be produced when an event list is created. Possible formats can be seen with EOls.

B.8. PathInstance Commands

B.8.1. PI

PathInstance: Commands: Displays a list of all PathInstance commands.

B.8.2. PIals

PIals: PathInstance: Attribute List: Displays a listing of raw attributes of the selected Path.

B.8.3. PIcp

PIcp: PathInstance: Copy: Create a copy of a selected Path.

B.8.4. PIdf

PIdf: PathInstance: Duration Fraction: Provide a new list of duration fractions for each pitch group of the active Path. Duration fractions are proportional weightings that scale a total duration provided by a Texture. When used within a Texture, each pitch group of the Path will be sustained for this proportional duration. Values must be given in a comma-separated list, and can be percentages or real values.

B.8.5. PIe

PIe: PathInstance: Edit: Edit a single Multiset in the active Path.

B.8.6. PIh

PIh: PathInstance: Hear: Creates a temporary Texture with the active Path and the active TextureModule, and uses this Texture to write a short sample EventList as a temporary MIDI file. This file is written in the scratch directory specified by APdir command. If possible, this file is opened and presented to the user.

B.8.7. PIls

PIls: PathInstance: List: Displays a list of all Paths.

B.8.8. PImv

PImv: PathInstance: Move: Rename a Path, and all Texture references to that Path.

B.8.9. PIn

PIn: PathInstance: New: Create a new Path from user-specified pitch groups. Users may specify pitch groups in a variety of formats. A Forte set class number (6-23A), a pitch-class set (4,3,9), a pitch-space set (-3, 23.2, 14), standard pitch letter names (A, C##, E~, G#), MIDI note numbers (58m, 62m), frequency values (222hz, 1403hz), a Xenakis sieve (5&3|11), or an Audacity frequency-analysis file (import) all may be provided. Pitches may be specified by letter name (psName), pitch space (psReal), pitch class, MIDI note number, or frequency. Pitch letter names may be specified as follows: a sharp is represented as "#"; a flat is represented as "$"; a quarter sharp is represented as "~"; multiple sharps, quarter sharps, and flats are valid. Octave numbers (where middle-C is C4) can be used with pitch letter names to provide register. Pitch space values (as well as pitch class) place C4 at 0.0. MIDI note numbers place C4 at 60. Numerical representations may encode microtones with additional decimal places. MIDI note-numbers and frequency values must contain the appropriate unit as a string ("m" or "hz"). Xenakis sieves are entered using logic constructions of residual classes. Residual classes are specified by a modulus and shift, where modulus 3 at shift 1 is notated 3@1. Logical operations are notated with "&" (and), "|" (or), "^" (symmetric difference), and "-" (complementation). Residual classes and logical operators may be nested and grouped by use of braces ({}). Complementation can be applied to a single residual class or a group of residual classes. For example: -{7@0|{-5@2&-4@3}}. When entering a sieve as a pitch set, the logic string may be followed by two comma-separated pitch notations for register bounds. For example "3@2|4, c1, c4" will take the sieve between c1 and c4. Audacity frequency-analysis files can be produced with the cross-platform open-source audio editor Audacity. In Audacity, under menu View, select Plot Spectrum, configure, and export. The file must have a .txt extension. To use the file-browser, enter "import"; to select the file from the prompt, enter the complete file path, optionally followed by a comma and the number of ranked pitches to read.

B.8.10. PIo

PIo: PathInstance: Select: Select the active Path. Used for "PIret", "PIrot", "PIslc", and "TIn".

B.8.11. PIret

PIret: PathInstance: Retrograde: Creates a new Path from the retrograde of the active Path. All PathVoices are preserved in the new Path.

B.8.12. PIrm

PIrm (name): PathInstance: Remove: Delete a selected Path.

B.8.13. PIrot

PIrot: PathInstance: Rotation: Creates a new Path from the rotation of the active Path. Note: since a rotation creates a map not previously defined, PathVoices are not preserved in the new Path.

B.8.14. PIslc

PIslc: PathInstance: Slice: Creates a new Path from a slice of the active Path. All PathVoices are preserved in the new Path.

B.8.15. PIv

PIv: PathInstance: View: Displays all properties of the active Path.

B.9. TextureClone Commands

B.9.1. TC

TextureClone: Commands: Displays a list of all TextureClone commands.

B.9.2. TCals

TCals: TextureClone: Attribute List: Displays raw attributes of the active Clone.

B.9.3. TCcp

TCcp: TextureClone: Copy: Duplicates a user-selected Clone associated with the active Texture.

B.9.4. TCdoc

TCdoc: TextureClone: Documentation: Displays documentation for each auxiliary parameter field from the associated Texture, as well as argument formats for static Clone options.

B.9.5. TCe

TCe: TextureClone: Edit: Edit attributes of the active Clone.

B.9.6. TCls

TCls: TextureClone: List: Displays a list of all Clones associated with the active Texture.

B.9.7. TCmap

TCmap: TextureClone: Map: Displays a graphical map of the parameter values of the active Clone. With the use of one optional argument, the TCmap display can be presented in two orientations. A TCmap diagram can position values on the x-axis in an equal-spaced orientation for each event (event-base), or in a time-proportional orientation, where width is relative to the time of each event (time-base). As Clones process values produced by a Texture, all TCmap displays are post-TM. This command uses the active graphic output format; this can be selected with the "APgfx" command. Output in "tk" requires the Python Tkinter GUI installation; output in "png" and "jpg" requires the Python Imaging Library (PIL) library installation; output in "eps" and "text" do not require any additional software or configuration.

B.9.8. TCmute

TCmute: TextureClone: Mute: Toggle the active Clone (or any number of Clones named with arguments) on or off. Muting a Clone prevents it from producing EventOutputs.

B.9.9. TCn

TCn: TextureClone: New: Creates a new Clone associated with the active Texture.

B.9.10. TCo

TCo: TextureClone: Select: Choose the active Clone from all available Clones associated with the active Texture.

B.9.11. TCrm

TCrm: TextureClone: Remove: Deletes a Clone from the active Texture.

B.9.12. TCv

TCv: TextureClone: View: Displays all editable attributes of the active Clone, or a Clone named with a single argument.

B.10. TextureEnsemble Commands

B.10.1. TE

TextureEnsemble: Commands: Displays a list of all TextureEnsemble commands.

B.10.2. TEe

TEe: TextureEnsemble: Edit: Edit a user-selected attribute for all Textures.

B.10.3. TEmap

TEmap: TextureEnsemble: Map: Provides a text-based display and/or graphical display of the temporal distribution of Textures and Clones. This command uses the active graphic output format; this can be selected with the "APgfx" command. Output in "tk" requires the Python Tkinter GUI installation; output in "png" and "jpg" requires the Python Imaging Library (PIL) library installation; output in "eps" and "text" do not require any additional software or configuration.

B.10.4. TEmidi

TEmidi: TextureEnsemble: MidiTempo: Edit the tempo written in a MIDI file. Where each Texture may have an independent tempo, a MIDI file has one tempo. The tempo written in the MIDI file does not effect playback, but may effect transcription into Western notation. The default tempo is 120 BPM.

B.10.5. TEv

TEv: TextureEnsemble: View: Displays a list of ParameterObject arguments for a single attribute of all Textures.

B.11. TextureInstance Commands

B.11.1. TI

TextureInstance: Commands: Displays a list of all TextureInstance commands.

B.11.2. TIals

TIals: TextureInstance: Attribute List: Displays raw attributes of a Texture.

B.11.3. TIcp

TIcp: TextureInstance: Copy: Duplicates a user-selected Texture.

B.11.4. TIdoc

TIdoc: TextureInstance: Documentation: Displays auxiliary parameter field documentation for a Texture's instrument, as well as argument details for static and dynamic Texture parameters.

B.11.5. TIe

TIe: TextureInstance: Edit: Edit a user-selected attribute of the active Texture.

B.11.6. TIls

TIls: TextureInstance: List: Displays a list of all Textures.

B.11.7. TImap

TImap: TextureInstance: Map: Displays a graphical map of the parameter values of the active Texture. With the use of two optional arguments, the TImap display can be presented in four orientations. A TImap diagram can position values on the x-axis in an equal-spaced orientation for each event (event-base), or in a time-proportional orientation, where width is relative to the time of each event (time-base). A TImap diagram can display, for each parameter, direct ParameterObject values as provided to the TextureModule (pre-TM), or the values of each parameter of each event after TextureModule processing (post-TM). This command uses the active graphic output format; this can be selected with the "APgfx" command. Output in "tk" requires the Python Tkinter GUI installation; output in "png" and "jpg" requires the Python Imaging Library (PIL) library installation; output in "eps" and "text" do not require any additional software or configuration.

B.11.8. TImidi

TImidi: TextureInstance: MIDI: Set the MIDI program and MIDI channel of a Texture, used when a "midiFile" EventOutput is selected. Users can select from one of the 128 GM MIDI programs by name or number. MIDI channels are normally auto-assigned during event list production; manually entered channel numbers (1 through 16) will override this feature.

B.11.9. TImode

TImode: TextureInstance: Mode: Set the pitch, polyphony, silence, and orcMap modes for the active Texture. The pitchMode (either "sc", "pcs", or "ps") designates which Path form is used within the Texture: "sc" designates the set class Path, which consists of non-transposed, non-redundant pitch classes; "pcs" designates the pitch class space Path, retaining set order and transposition; "ps" designates the pitch space Path, retaining order, transposition, and register.

B.11.10. TImute

TImute: TextureInstance: Mute: Toggle the active Texture (or any number of Textures named with arguments) on or off. Muting a Texture prevents it from producing EventOutputs. Clones can be created from muted Textures.

B.11.11. TImv

TImv: TextureInstance: Move: Renames a Texture, and all references in existing Clones.

B.11.12. TIn

TIn: TextureInstance: New: Creates a new instance of a Texture with a user supplied Instrument and Texture name. The new instance uses the active TextureModule, the active Path, and an Instrument selected from the active EventMode-determined Orchestra. For some Orchestras, the user must supply the number of auxiliary parameters.

B.11.13. TIo

TIo: TextureInstance: Select: Select the active Texture from all available Textures.

B.11.14. TIrm

TIrm: TextureInstance: Remove: Deletes a user-selected Texture.

B.11.15. TIv

TIv: TextureInstance: View: Displays all editable attributes of the active Texture, or a Texture named with a single argument.

B.12. TextureModule Commands

B.12.1. TM

TextureModule: Commands: Displays a list of all TextureModule commands.

B.12.2. TMls

TMls: TextureModule: List: Displays a list of all TextureModules.

B.12.3. TMo

TMo: TextureModule: Select: Choose the active TextureModule. This TextureModule is used with the "TIn" and "TMv" commands.

B.12.4. TMv

TMv: TextureModule: View: Displays documentation for the active TextureModule.

B.13. TextureParameter Commands

B.13.1. TP

TextureParameter: Commands: Displays a list of all TextureParameter commands.

B.13.2. TPe

TPe: TextureParameter: Export: Export ParameterObject data as a file; file types available are pureDataArray, audioFile, textSpace, and textTab.

B.13.3. TPls

TPls: TextureParameter: List: Displays a list of all ParameterObjects.

B.13.4. TPmap

TPmap: TextureParameter: Map: Displays a graphical map of any ParameterObject. User must supply parameter library name, the number of events to be calculated, and appropriate parameter arguments. This command uses the active graphic output format; this can be selected with the "APgfx" command. Output in "tk" requires the Python Tkinter GUI installation; output in "png" and "jpg" requires the Python Imaging Library (PIL) library installation; output in "eps" and "text" do not require any additional software or configuration.

B.13.5. TPv

TPv: TextureParameter: View: Displays documentation for one or more ParameterObjects. All ParameterObjects that match the user-supplied search string will be displayed. ParameterObject acronyms are accepted.

B.14. TextureTemperament Commands

B.14.1. TT

TextureTemperament: Commands: Displays a list of all TextureTemperament commands.

B.14.2. TTls

TTls: TextureTemperament: List: Displays a list of all temperaments available.

B.14.3. TTo

TTo: TextureTemperament: Select: Choose a Temperament for the active Texture. The Temperament provides fixed or dynamic mapping of pitch values. Fixed mappings emulate historical Temperaments, such as MeanTone and Pythagorean; dynamic mappings provide algorithmic variation to each pitch processed, such as microtonal noise.

B.15. Other Commands

B.15.1. cmd

cmd: Displays a hierarchical menu of all athenaCL commands.

B.15.2. help

help: To get help for a command or any available topic, enter "help" or "?" followed by a search string. If no command is provided, a menu of all commands available is displayed.

B.15.3. py

Begins an interactive Python session inside the current athenaCL session.

B.15.4. pypath

pypath: Lists all file paths in the Python search path.

B.15.5. q

q: Exit athenaCL.

B.15.6. quit

quit: Exit athenaCL.

B.15.7. shell

On UNIX-based platforms, the "shell" or "!" command executes a command-line argument in the default shell.

Appendix C. ParameterObject Reference and Examples

C.1. Generator ParameterObjects

C.1.1. accumulator (a)

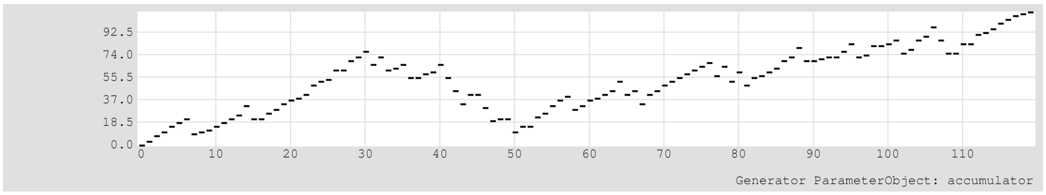
accumulator, initValue, parameterObject

Description: For each evaluation, this Generator adds the result of the Generator ParameterObject to the stored cumulative numeric value; the initialization value argument initValue is the origin of the cumulative value, and is the first value returned.

Arguments: (1) name, (2) initValue, (3) parameterObject {Generator}

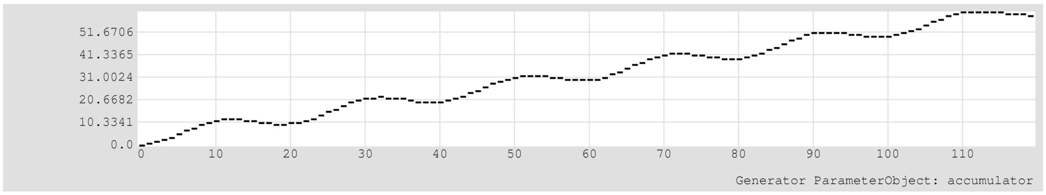
Sample Arguments: `a, 0, (bg,rc,(1,3,4,7,-11))`

Example C-1. accumulator Demonstration 1



```
accumulator, 0, (basketGen, randomChoice, (1,3,4,7,-11))
```

Example C-2. accumulator Demonstration 2



```
accumulator, 0, (waveSine, event, (constant, 20), 0, (constant, -0.5),
(constant, 1.5))
```

C.1.2. basketFill (bf)

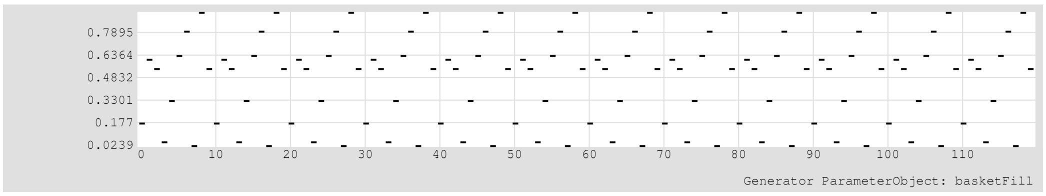
basketFill, selectionString, parameterObject, valueCount

Description: Chooses values from a ParameterObject generated list of values. The number of values generated is determined by the valueCount integer. Valuse are generated only at initialization. Values are chosen from this list using the selector specified by the selectionString argument.

Arguments: (1) name, (2) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}, (3) parameterObject {source Generator}, (4) valueCount

Sample Arguments: `bf, oc, (ru,0,1), 10`

Example C-3. basketFill Demonstration 1



```
basketFill, orderedCyclic, (randomUniform, (constant, 0), (constant, 1)), 10
```

C.1.3. basketFillSelect (bfs)

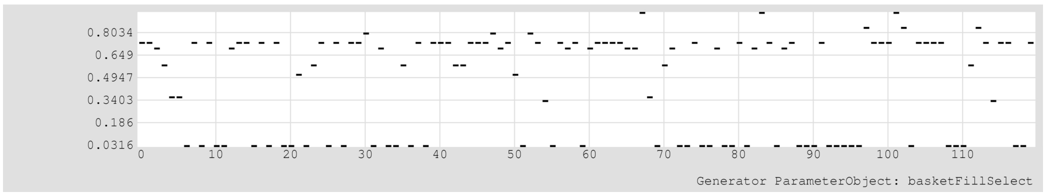
basketFillSelect, parameterObject, valueCount, parameterObject

Description: Chooses values from a ParameterObject generated list of values. The number of values generated is determined by the valueCount integer. Valuse are generated only at initialization. Values are choosen from the list with values within the unit interval produced by an embedded ParameterObject. Values that exceed the unit interval are limited within the unit interval.

Arguments: (1) name, (2) parameterObject {source Generator}, (3) valueCount, (4) parameterObject {selection Generator}

Sample Arguments: `bfs, (ru,0,1), 10, (rb,0.2,0.2,0,1)`

Example C-4. basketFillSelect Demonstration 1



```
basketFillSelect, (randomUniform, (constant, 0), (constant, 1)), 10,
(randomBeta, 0.2, 0.2, (constant, 0), (constant, 1))
```

C.1.4. basketGen (bg)

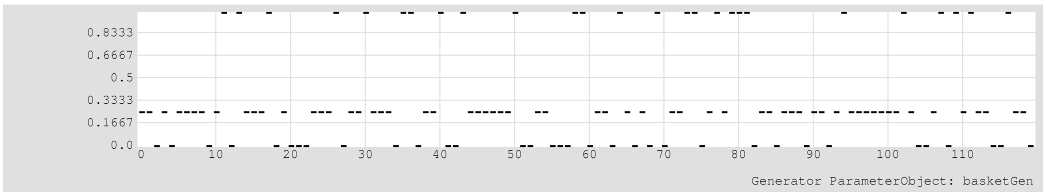
basketGen, selectionString, valueList

Description: Chooses values from a user-supplied list (valueList). Values can be strings or numbers. Values are chosen from this list using the selector specified by the selectionString argument.

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

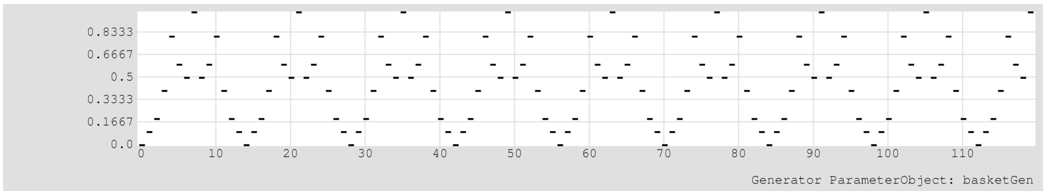
Sample Arguments: `bg, rc, (0,0.25,0.25,1)`

Example C-5. basketGen Demonstration 1



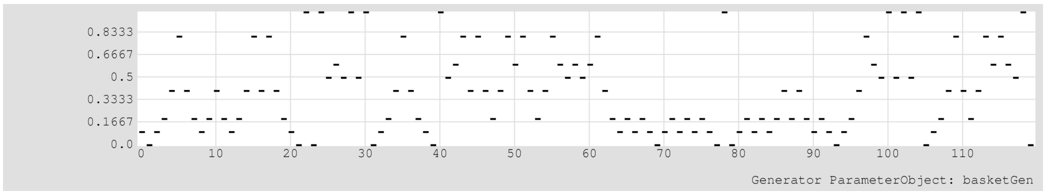
```
basketGen, randomChoice, (0,0.25,0.25,1)
```

Example C-6. basketGen Demonstration 2



```
basketGen, orderedOscillate, (0,0.1,0.2,0.4,0.8,0.6,0.5,1)
```

Example C-7. basketGen Demonstration 3



```
basketGen, randomWalk, (0,0.1,0.2,0.4,0.8,0.6,0.5,1)
```

C.1.5. breakGraphFlat (bgf)

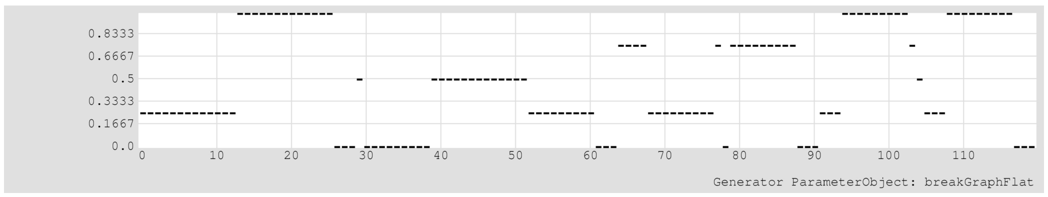
breakGraphFlat, stepString, edgeString, parameterObject, parameterObject, pointCount

Description: Provides a dynamic break-point function without interpolation. A list of (x,y) coordinate pairs is generated from two embedded Generator ParameterObjects. The number of generated pairs is determined by the count argument. A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) parameterObject {x point Generator}, (5) parameterObject {y point Generator}, (6) pointCount

Sample Arguments: `bgf, e, l, (a,0,(bg,rp,(1,3,9))), (bg,rc,(0,0.25,0.5,0.75,1)), 60`

Example C-8. breakGraphFlat Demonstration 1



```
breakGraphFlat, event, loop, (accumulator, 0, (basketGen, randomPermutate,
(1,3,9))), (basketGen, randomChoice, (0,0.25,0.5,0.75,1)), 60
```

C.1.6. breakGraphHalfCosine (bghc)

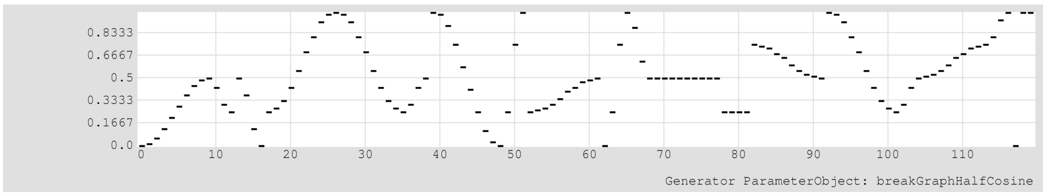
breakGraphHalfCosine, stepString, edgeString, parameterObject, parameterObject, pointCount

Description: Provides a dynamic break-point function with half-cosine interpolation. A list of (x,y) coordinate pairs is generated from two embedded Generator ParameterObjects. The number of generated pairs is determined by the count argument. A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates. The exponent argument may be any positive or negative numeric value.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) parameterObject {x point Generator}, (5) parameterObject {y point Generator}, (6) pointCount

Sample Arguments: `bghc, e, l, (a,0,(bg,rp,(1,3,9))), (bg,rc,(0,0.25,0.5,0.75,1)), 60`

Example C-9. breakGraphHalfCosine Demonstration 1



```
breakGraphHalfCosine, event, loop, (accumulator, 0, (basketGen,
randomPermutate, (1,3,9))), (basketGen, randomChoice, (0,0.25,0.5,0.75,1)), 60
```

C.1.7. breakGraphLinear (bgl)

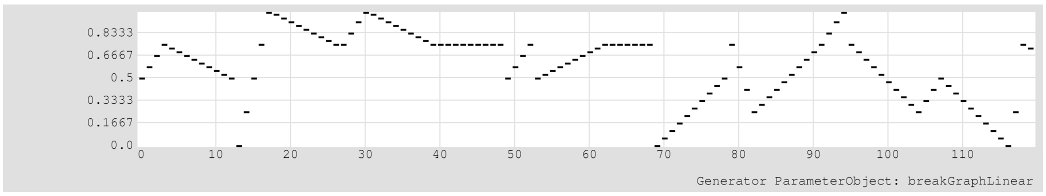
breakGraphLinear, stepString, edgeString, parameterObject, parameterObject, pointCount

Description: Provides a dynamic break-point function with linear interpolation. A list of (x,y) coordinate pairs is generated from two embedded Generator ParameterObjects. The number of generated pairs is determined by the count argument. A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) parameterObject {x point Generator}, (5) parameterObject {y point Generator}, (6) pointCount

Sample Arguments: `bgl, e, l, (a,0,(bg,rp,(1,3,9))), (bg,rc,(0,0.25,0.5,0.75,1)), 60`

Example C-10. breakGraphLinear Demonstration 1



```
breakGraphLinear, event, loop, (accumulator, 0, (basketGen, randomPermutate,
(1,3,9))), (basketGen, randomChoice, (0,0.25,0.5,0.75,1)), 60
```

C.1.8. breakGraphPower (bgp)

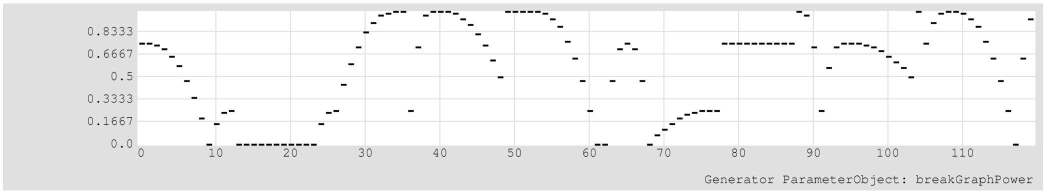
breakGraphPower, stepString, edgeString, parameterObject, parameterObject, pointCount, exponent

Description: Provides a dynamic break-point function with exponential interpolation. A list of (x,y) coordinate pairs is generated from two embedded Generator ParameterObjects. The number of generated pairs is determined by the count argument. A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates. The exponent argument may be any positive or negative numeric value.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) parameterObject {x point Generator}, (5) parameterObject {y point Generator}, (6) pointCount, (7) exponent

Sample Arguments: `bgp, e, l, (a,0,(bg,rp,(1,3,9))), (bg,rc,(0,0.25,0.5,0.75,1)), 60, -1.5`

Example C-11. breakGraphPower Demonstration 1



```
breakGraphPower, event, loop, (accumulator, 0, (basketGen, randomPermutate,
(1,3,9))), (basketGen, randomChoice, (0,0.25,0.5,0.75,1)), 60, -1.5
```

C.1.9. breakPointFlat (bpf)

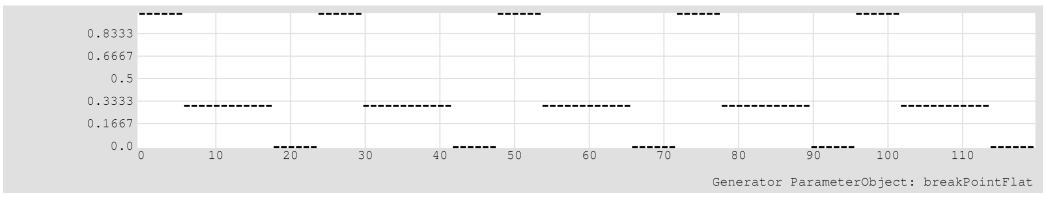
breakPointFlat, stepString, edgeString, pointList

Description: Provides a break-point function without interpolation from a list of (x,y) coordinate pairs (pointList). A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) pointList

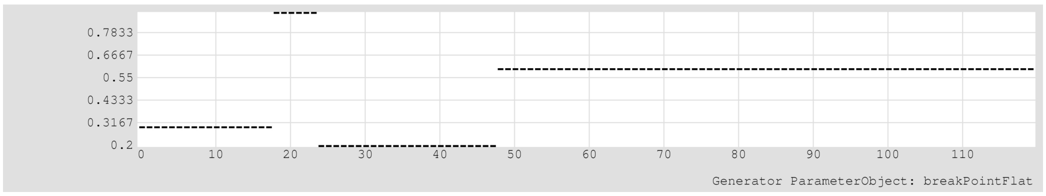
Sample Arguments: `bpf, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))`

Example C-12. breakPointFlat Demonstration 1



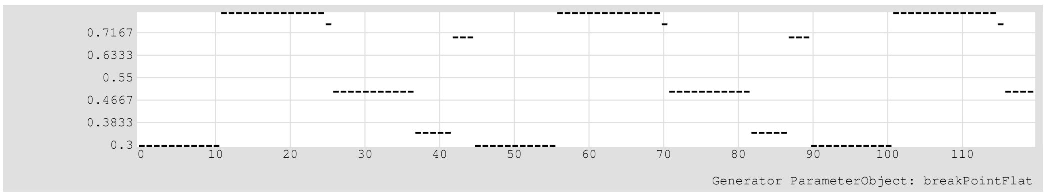
```
breakPointFlat, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))
```

Example C-13. breakPointFlat Demonstration 2



```
breakPointFlat, event, single, ((12,0.3),(18,0.9),(24,0.2),(48,0.6))
```

Example C-14. breakPointFlat Demonstration 3



```
breakPointFlat, event, loop,
((0,0.3),(10,0.3),(11,0.8),(25,0.75),(26,0.5),(37,0.35),(42,0.7),(45,0.5))
```

C.1.10. breakPointHalfCosine (bphc)

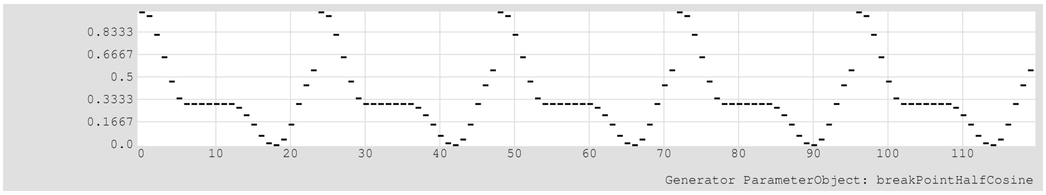
breakPointHalfCosine, stepString, edgeString, pointList

Description: Provides a break-point function with half-cosine interpolation from a list of (x,y) coordinate pairs (pointList). A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) pointList

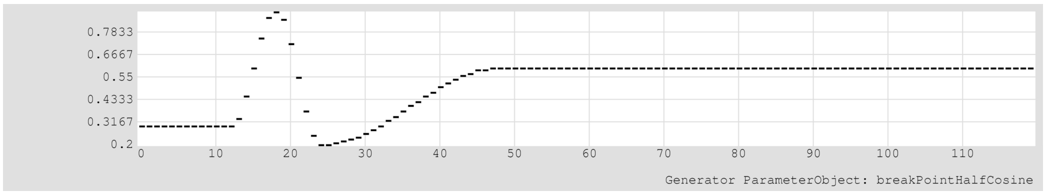
Sample Arguments: `bphc, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))`

Example C-15. breakPointHalfCosine Demonstration 1



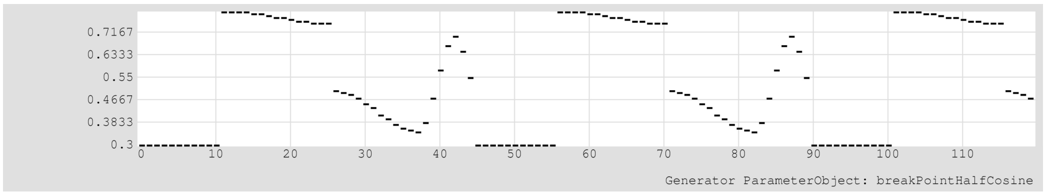
```
breakPointHalfCosine, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))
```

Example C-16. breakPointHalfCosine Demonstration 2



```
breakPointHalfCosine, event, single, ((12,0.3),(18,0.9),(24,0.2),(48,0.6))
```

Example C-17. breakPointHalfCosine Demonstration 3



```
breakPointHalfCosine, event, loop,
((0,0.3),(10,0.3),(11,0.8),(25,0.75),(26,0.5),(37,0.35),(42,0.7),(45,0.5))
```

C.1.11. breakPointLinear (bpl)

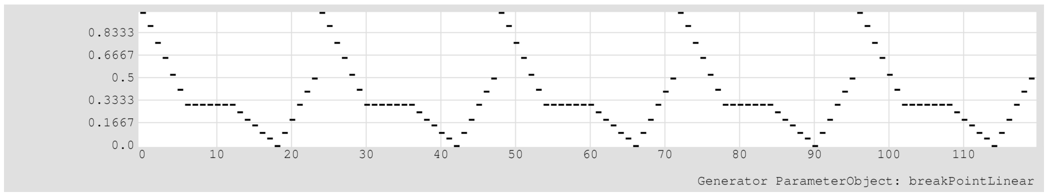
breakPointLinear, stepString, edgeString, pointList

Description: Provides a break-point function with linear interpolation from a list of (x,y) coordinate pairs (pointList). A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) pointList

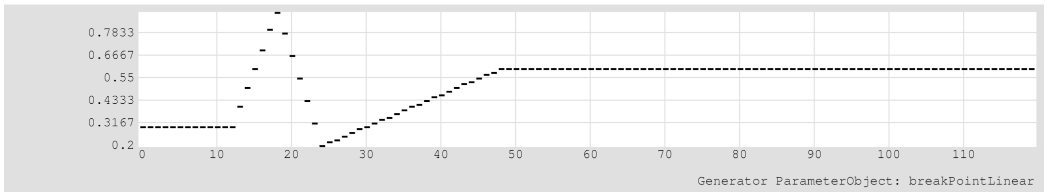
Sample Arguments: `bpl, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))`

Example C-18. breakPointLinear Demonstration 1



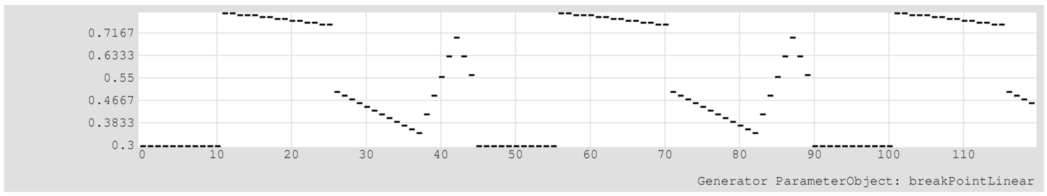
```
breakPointLinear, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))
```

Example C-19. breakPointLinear Demonstration 2



```
breakPointLinear, event, single, ((12,0.3),(18,0.9),(24,0.2),(48,0.6))
```

Example C-20. breakPointLinear Demonstration 3



```
breakPointLinear, event, loop,
((0,0.3),(10,0.3),(11,0.8),(25,0.75),(26,0.5),(37,0.35),(42,0.7),(45,0.5))
```

C.1.12. breakPointPower (bpp)

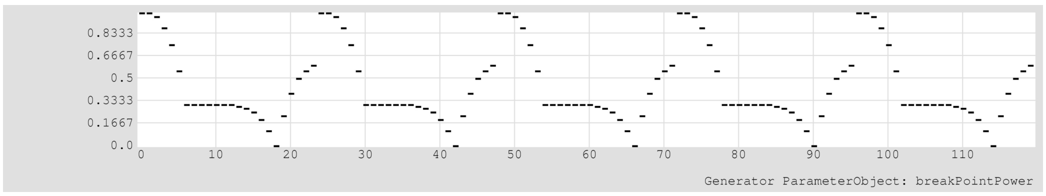
breakPointPower, stepString, edgeString, pointList, exponent

Description: Provides a break-point function with exponential interpolation from a list of (x,y) coordinate pairs (pointList). A step type (stepString) determines if x values in the pointList refer to events or real-time values. Interpolated y values are the output of the Generator. The edgeString argument determines if the break-point function loops, or is executed once at the given coordinates. The exponent argument may be any positive or negative numeric value.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) edgeString {'loop', 'single'}, (4) pointList, (5) exponent

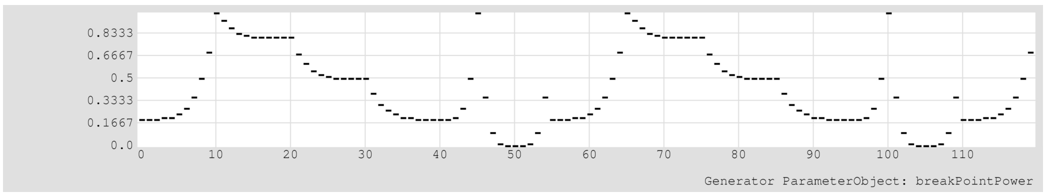
Sample Arguments: `bpp, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6)), -1.5`

Example C-21. breakPointPower Demonstration 1



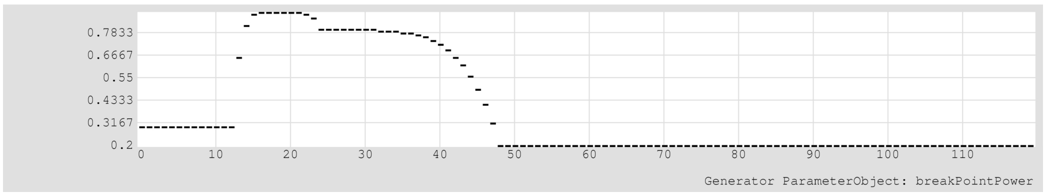
```
breakPointPower, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6)), -1.5
```

Example C-22. breakPointPower Demonstration 2



```
breakPointPower, event, loop,
((0,0.2),(10,1),(20,0.8),(30,0.5),(40,0.2),(45,1),(50,0),(55,1)), 3.5
```

Example C-23. breakPointPower Demonstration 3



```
breakPointPower, event, single, ((12,0.3),(18,0.9),(24,0.8),(48,0.2)), -4
```

C.1.13. basketSelect (bs)

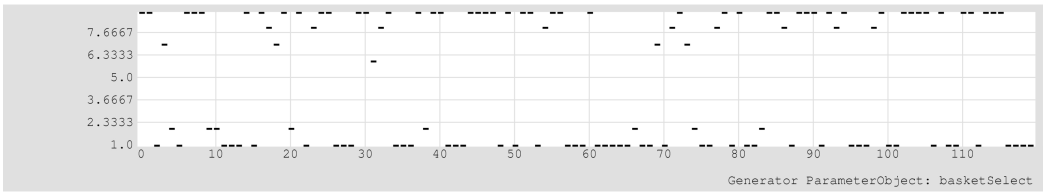
basketSelect, valueList, parameterObject

Description: Chooses values from a user-supplied list (valueList). Values can be strings or numbers. Values are choosen from the list with values within the unit interval produced by an embedded ParameterObject. Values that exceed the unit interval are limited within the unit interval.

Arguments: (1) name, (2) valueList, (3) parameterObject {selection Generator}

Sample Arguments: `bs, (1,2,3,4,5,6,7,8,9), (rb,0.2,0.2,(bpl,e,s,((0,0.4),(120,0))),(bpl,e,s,((0,0.6),(120,1))))`

Example C-24. basketSelect Demonstration 1



```
basketSelect, (1,2,3,4,5,6,7,8,9), (randomBeta, 0.2, 0.2, (breakPointLinear,
event, single, ((0,0.4),(120,0))), (breakPointLinear, event, single,
((0,0.6),(120,1)))),
```

C.1.14. constant (c)

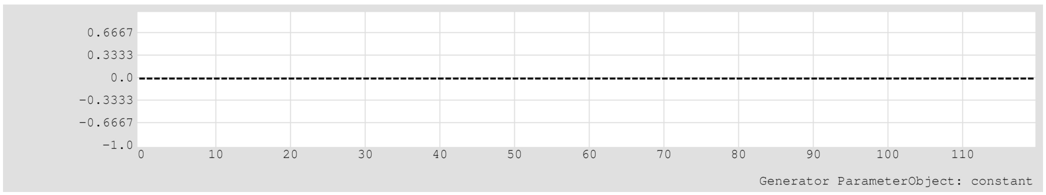
constant, value

Description: Return a constant string or numeric value.

Arguments: (1) name, (2) value

Sample Arguments: `c, 0`

Example C-25. constant Demonstration 1



```
constant, 0
```

C.1.15. constantFile (cf)

constantFile, absoluteFilePath

Description: Given an absolute file path, a constant file path is returned as a string. Note: symbolic links (aliases or shortcuts) and home directory symbols (~) are expanded into complete paths.

Arguments: (1) name, (2) absoluteFilePath

Sample Arguments: `cf,`

C.1.16. cyclicGen (cg)

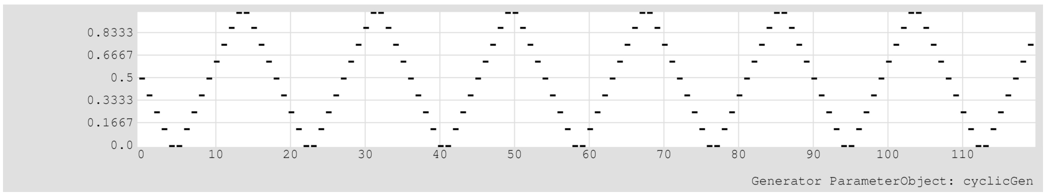
cyclicGen, directionString, min, max, increment

Description: Cycles between static minimum (min) and maximum (max) values with a static increment value. Cycling direction and type is controlled by the directionString argument.

Arguments: (1) name, (2) directionString {'upDown', 'downUp', 'up', 'down'}, (3) min, (4) max, (5) increment

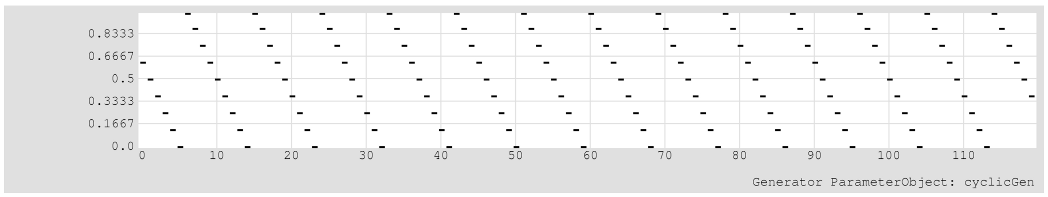
Sample Arguments: `cg, ud, 0, 1, 0.13`

Example C-26. cyclicGen Demonstration 1



```
cyclicGen, upDown, 0, 1, 0.13
```

Example C-27. cyclicGen Demonstration 2



```
cyclicGen, down, 0, 1, 0.13
```

C.1.17. caList (cl)

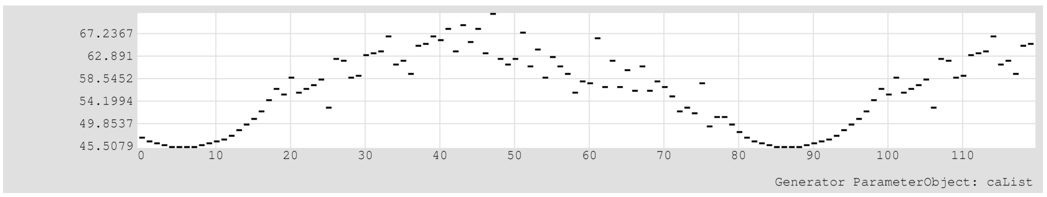
caList, caSpec, parameterObject, parameterObject, tableExtractionString, selectionString

Description: Produces values from a one-dimensional cellular automata table. One dimensional cellular automata may be standard, totalistic, continuous, or float formats, and are defined with a caSpec string. The caSpec string may contain one or more CA parameters defined in key{value} pairs. All parameters not defined assume default values. Valid parameters include f (format), k, r, i (initialization), x (row width), y (number of steps), w (extracted width), c (extracted center), and s (initial step skip). Rule and mutation values may be provided by embedded Generator ParameterObjects. Values may be extracted into a list from the resulting table as defined by the tableFormatString. Values are chosen from this list using the selector specified by the selectionString argument.

Arguments: (1) name, (2) caSpec, (3) parameterObject {rule}, (4) parameterObject {mutation}, (5) tableExtractionString {'averageColumn', 'averageColumnActive', 'averageColumnIndex', 'averageColumnIndexActive', 'averageColumnIndexPassive', 'averageColumnPassive', 'averageRow', 'averageRowActive', 'averageRowIndex', 'averageRowIndexActive', 'averageRowIndexPassive', 'averageRowPassive', 'flatColumn', 'flatColumnActive', 'flatColumnIndex', 'flatColumnIndexActive', 'flatColumnIndexPassive', 'flatColumnPassive', 'flatColumnReflect', 'flatColumnReflectActive', 'flatColumnReflectIndex', 'flatColumnReflectIndexActive', 'flatColumnReflectIndexPassive', 'flatColumnReflectPassive', 'flatRow', 'flatRowActive', 'flatRowIndex', 'flatRowIndexActive', 'flatRowIndexPassive', 'flatRowPassive', 'flatRowReflect', 'flatRowReflectActive', 'flatRowReflectIndex', 'flatRowReflectIndexActive', 'flatRowReflectIndexPassive', 'flatRowReflectPassive', 'productColumn', 'productColumnActive', 'productColumnIndex', 'productColumnIndexActive', 'productColumnIndexPassive', 'productColumnPassive', 'productRow', 'productRowActive', 'productRowIndex', 'productRowIndexActive', 'productRowIndexPassive', 'productRowPassive', 'sumColumn', 'sumColumnActive', 'sumColumnIndex', 'sumColumnIndexActive', 'sumColumnIndexPassive', 'sumColumnPassive', 'sumRow', 'sumRowActive', 'sumRowIndex', 'sumRowIndexActive', 'sumRowIndexPassive', 'sumRowPassive'}, (6) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

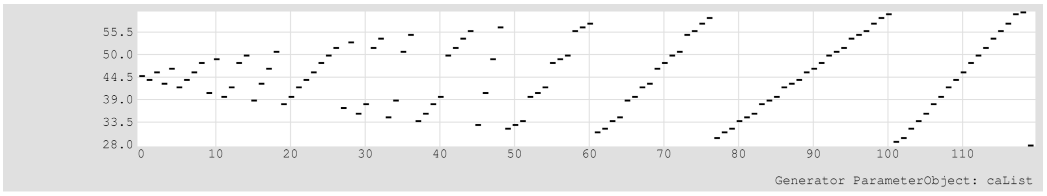
Sample Arguments: `cl, f{f}i{c}x{81}y{120}, 0.25, 0.0005, sc, oc`

Example C-28. caList Demonstration 1



```
caList, f{f}k{0}r{1}i{center}x{81}y{120}w{81}c{0}s{0}, (constant, 0.25),
(constant, 0.0005), sumColumn, orderedCyclic
```

Example C-29. caList Demonstration 2



```
caList, f{s}k{2}r{1}i{center}x{91}y{120}w{91}c{0}s{0}, (markovValue,
a{90}b{182}:{a=29|b=1}, (constant, 0)), (constant, 0), flatRowIndexActive,
orderedCyclic
```

C.1.18. caValue (cv)

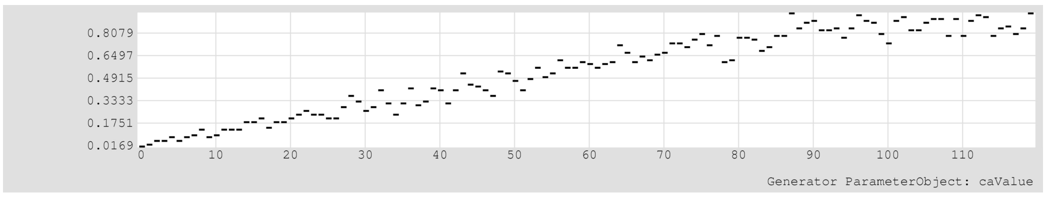
caValue, caSpec, parameterObject, parameterObject, tableExtractionString, min, max, selectionString

Description: Produces values from a one-dimensional cellular automata table scaled within dynamic min and max values. One dimensional cellular automata may be standard, totalistic, continuous, or float formats, and are defined with a caSpec string. The caSpec string may contain one or more CA parameters defined in key{value} pairs. All parameters not defined assume default values. Valid parameters include f (format), k, r, i (initialization), x (row width), y (number of steps), w (extracted width), c (extracted center), and s (initial step skip). Rule and mutation values may be provided by embedded Generator ParameterObjects. Values may be extracted into a list from the resulting table as defined by the tableFormatString. 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) caSpec, (3) parameterObject {rule}, (4) parameterObject {mutation}, (5) tableExtractionString {'averageColumn', 'averageColumnActive', 'averageColumnIndex', 'averageColumnIndexActive', 'averageColumnIndexPassive', 'averageColumnPassive', 'averageRow', 'averageRowActive', 'averageRowIndex', 'averageRowIndexActive', 'averageRowIndexPassive', 'averageRowPassive', 'flatColumn', 'flatColumnActive', 'flatColumnIndex', 'flatColumnIndexActive', 'flatColumnIndexPassive', 'flatColumnPassive', 'flatColumnReflect', 'flatColumnReflectActive', 'flatColumnReflectIndex', 'flatColumnReflectIndexActive', 'flatColumnReflectIndexPassive', 'flatColumnReflectPassive', 'flatRow', 'flatRowActive', 'flatRowIndex', 'flatRowIndexActive', 'flatRowIndexPassive', 'flatRowPassive', 'flatRowReflect', 'flatRowReflectActive', 'flatRowReflectIndex', 'flatRowReflectIndexActive', 'flatRowReflectIndexPassive', 'flatRowReflectPassive', 'productColumn', 'productColumnActive', 'productColumnIndex', 'productColumnIndexActive', 'productColumnIndexPassive', 'productColumnPassive', 'productRow', 'productRowActive', 'productRowIndex', 'productRowIndexActive', 'productRowIndexPassive', 'productRowPassive', 'sumColumn', 'sumColumnActive', 'sumColumnIndex', 'sumColumnIndexActive', 'sumColumnIndexPassive', 'sumColumnPassive', 'sumRow', 'sumRowActive', 'sumRowIndex', 'sumRowIndexActive', 'sumRowIndexPassive', 'sumRowPassive'}, (6) min, (7) max, (8) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

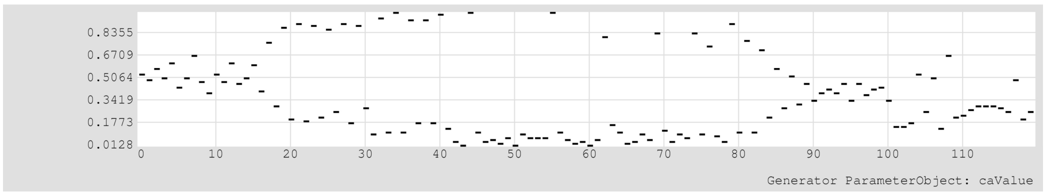
Sample Arguments: `cv, f{s}, (c,110), (c,0), sr, 0, 1, oc`

Example C-30. caValue Demonstration 1



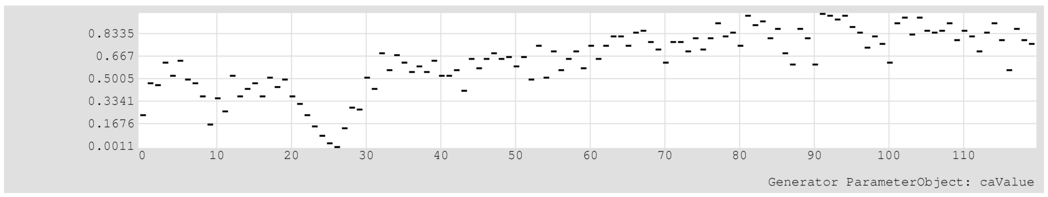
```
caValue, f{s}k{2}r{1}i{center}x{91}y{135}w{91}c{0}s{0}, (constant, 110),
(constant, 0), sumRow, (constant, 0), (constant, 1), orderedCyclic
```

Example C-31. caValue Demonstration 2



```
caValue, f{s}k{2}r{1}i{random}x{81}y{120}w{81}c{0}s{0}, (breakPointLinear,
event, single, ((0,30),(119,34))), (constant, 0.05), sumRow, (constant, 0),
(constant, 1), orderedCyclic
```

Example C-32. caValue Demonstration 3



```
caValue, f{t}k{3}r{1}i{center}x{81}y{120}w{12}c{0}s{0}, (constant, 1842),
(breakPointLinear, event, loop, ((0,0),(80,0.02))), sumRow, (waveSine, event,
(constant, 15), 0, (constant, 0), (constant, 0.4)), (constant, 1),
orderedCyclic
```

C.1.19. directorySelect (ds)

directorySelect, directoryFilePath, fileExtension, selectionString

Description: Within a user-provided directory (directoryFilePath) and all sub-directories, this Generator finds all files named with a file extension that matches the fileExtension argument, and collects these complete file paths into a list. Values are chosen from this list using the selector specified by the selectionString argument. Note: the fileExtension argument string may not include a leading period (for example, use "aif", not ".aif"); symbolic links (aliases or shortcuts) and home directory symbols (~) are expanded into complete paths.

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

Sample Arguments: `ds, ., aif, rw`

C.1.20. envelopeGeneratorAdsr (ega)

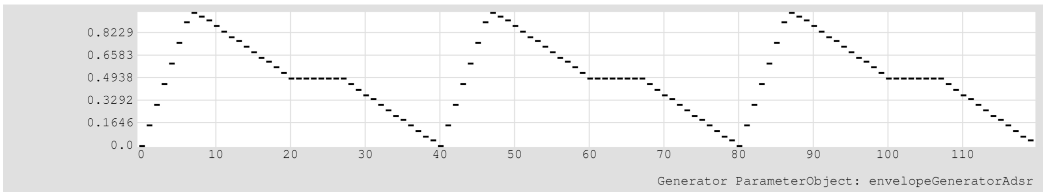
envelopeGeneratorAdsr, scaleString, edgeString, eventCount, parameterObject, parameterObject, parameterObject, parameterObject, parameterObject, parameterObject, min, max

Description: Generates a sequence of dynamic envelopes with durations controlled by a Generator Parameter Object. Envelope duration is specified by the duration ParameterObject; all values are interpreted in seconds. The scaleString parameter determines if shape values are interpreted as proportional values or absolute values in seconds. The number of envelopes generated is controlled by the eventCount parameter; envelopes are looped when necessary. The minimum and maximum envelope value is scaled within the range designated by min and max; min and max are selected once per envelope; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) scaleString {'absolute', 'proportional'}, (3) edgeString {'loop', 'single'}, (4) eventCount, (5) parameterObject {duration Generator}, (6) parameterObject {attack Generator}, (7) parameterObject {decay Generator}, (8) parameterObject {sustain Generator}, (9) parameterObject {release Generator}, (10) parameterObject {sustain scalar Generator}, (11) min, (12) max

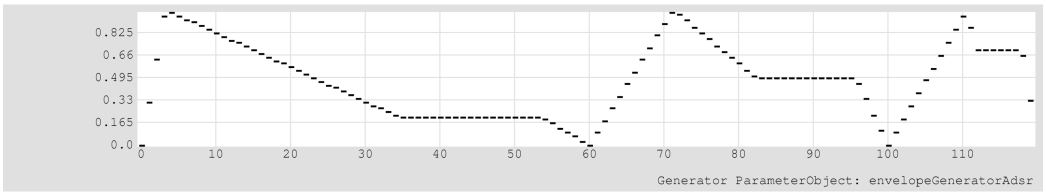
Sample Arguments: `ega, proportional, l, 100, (c,40), (c,2), (c,4), (c,2), (c,4), (c,0.5), 0, 1`

Example C-33. envelopeGeneratorAdsr Demonstration 1



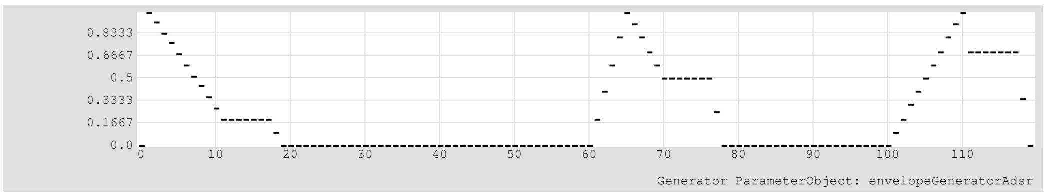
```
envelopeGeneratorAdsr, proportional, loop, 100, (constant, 40), (constant, 2),
(constant, 4), (constant, 2), (constant, 4), (constant, 0.5), (constant, 0),
(constant, 1)
```

Example C-34. envelopeGeneratorAdsr Demonstration 2



```
envelopeGeneratorAdsr, proportional, loop, 100, (basketGen, orderedCyclic,
(60,40,20)), (basketGen, orderedCyclic, (1,5,10)), (basketGen, orderedCyclic,
(10,5,1)), (constant, 6), (constant, 2), (basketGen, orderedCyclic,
(0.2,0.5,0.7)), (constant, 0), (constant, 1)
```

Example C-35. envelopeGeneratorAdsr Demonstration 3



```
envelopeGeneratorAdsr, absolute, loop, 100, (basketGen, orderedCyclic,
(60,40,20)), (basketGen, orderedCyclic, (1,5,10)), (basketGen, orderedCyclic,
(10,5,1)), (constant, 6), (constant, 2), (basketGen, orderedCyclic,
(0.2,0.5,0.7)), (constant, 0), (constant, 1)
```

C.1.21. envelopeGeneratorTrapezoid (egt)

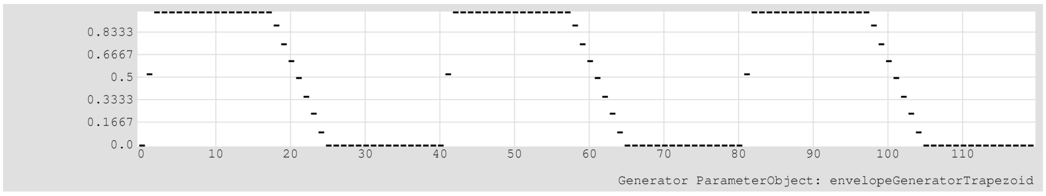
envelopeGeneratorTrapezoid, scaleString, edgeString, eventCount, parameterObject, parameterObject, parameterObject, parameterObject, parameterObject, min, max

Description: Generates a sequence of dynamic envelopes with durations controlled by a Generator Parameter Object. Envelope duration is specified by the duration ParameterObject; all values are interpreted in seconds. The scaleString parameter determines if shape values are interpreted as proportional values or absolute values in seconds. The number of envelopes generated is controlled by the eventCount parameter; envelopes are looped when necessary. The minimum and maximum envelope value is scaled within the range designated by min and max; min and max are selected once per envelope; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) scaleString {'absolute', 'proportional'}, (3) edgeString {'loop', 'single'}, (4) eventCount, (5) parameterObject {duration Generator}, (6) parameterObject {ramp up Generator}, (7) parameterObject {width max Generator}, (8) parameterObject {ramp down Generator}, (9) parameterObject {width min Generator}, (10) min, (11) max

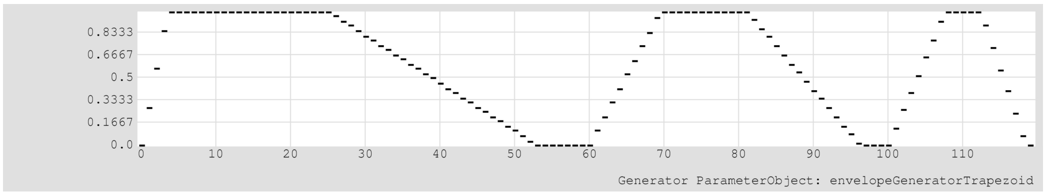
Sample Arguments: `egt, proportional, l, 100, (c,40), (c,0.5), (c,4), (c,2), (c,4), 0, 1`

Example C-36. envelopeGeneratorTrapezoid Demonstration 1



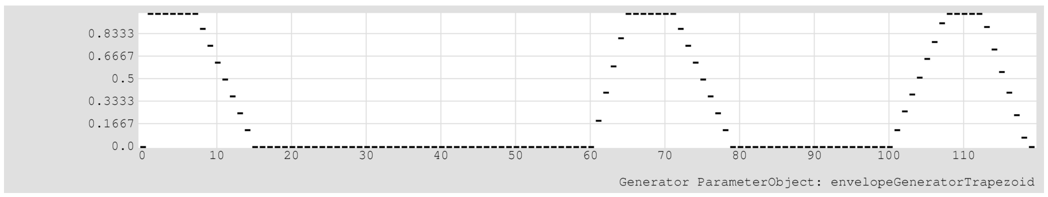
```
envelopeGeneratorTrapezoid, proportional, loop, 100, (constant, 40),
(constant, 0.5), (constant, 4), (constant, 2), (constant, 4), (constant, 0),
(constant, 1)
```

Example C-37. envelopeGeneratorTrapezoid Demonstration 2



```
envelopeGeneratorTrapezoid, proportional, loop, 100, (basketGen,
orderedCyclic, (60,40,20)), (basketGen, orderedCyclic, (1,5,10)), (constant,
6), (constant, 8), (constant, 2), (constant, 0), (constant, 1)
```

Example C-38. envelopeGeneratorTrapezoid Demonstration 3



```
envelopeGeneratorTrapezoid, absolute, loop, 100, (basketGen, orderedCyclic,
(60,40,20)), (basketGen, orderedCyclic, (1,5,10)), (constant, 6), (constant,
8), (constant, 2), (constant, 0), (constant, 1)
```

C.1.22. envelopeGeneratorUnit (egu)

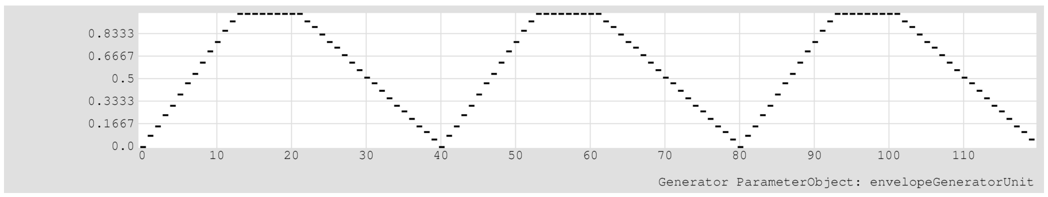
envelopeGeneratorUnit, edgeString, eventCount, parameterObject, parameterObject, parameterObject, min, max

Description: Generates a sequence of dynamic envelopes with durations controlled by a Generator Parameter Object. Envelope duration is specified by the duration ParameterObject; all values are interpreted in seconds. The scaleString parameter determines if shape values are interpreted as proportional values or absolute values in seconds. The number of envelopes generated is controlled by the eventCount parameter; envelopes are looped when necessary. The minimum and maximum envelope value is scaled within the range designated by min and max; min and max are selected once per envelope; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) edgeString {'loop', 'single'}, (3) eventCount, (4) parameterObject {duration Generator}, (5) parameterObject {sustain center unit Generator}, (6) parameterObject {sustain width unit Generator}, (7) min, (8) max

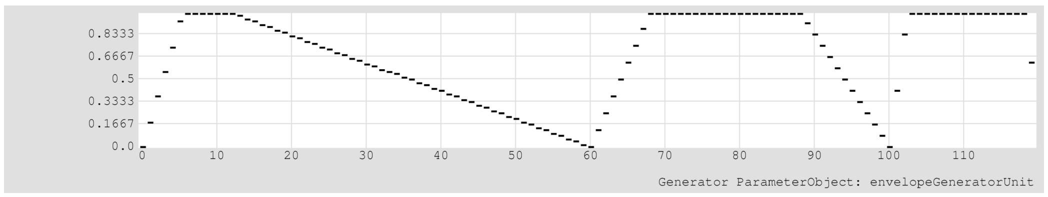
Sample Arguments: `egu, l, 100, (c,40), (c,0.4), (c,0.2), 0, 1`

Example C-39. envelopeGeneratorUnit Demonstration 1



```
envelopeGeneratorUnit, loop, 100, (constant, 40), (constant, 0.4), (constant,
0.2), (constant, 0), (constant, 1)
```

Example C-40. envelopeGeneratorUnit Demonstration 2



```
envelopeGeneratorUnit, loop, 100, (basketGen, orderedCyclic, (60,40,20)),
(basketGen, orderedCyclic, (0.1,0.4,0.6)), (basketGen, orderedCyclic,
(0.1,0.5,0.8)), (constant, 0), (constant, 1)
```

C.1.23. funnelBinary (fb)

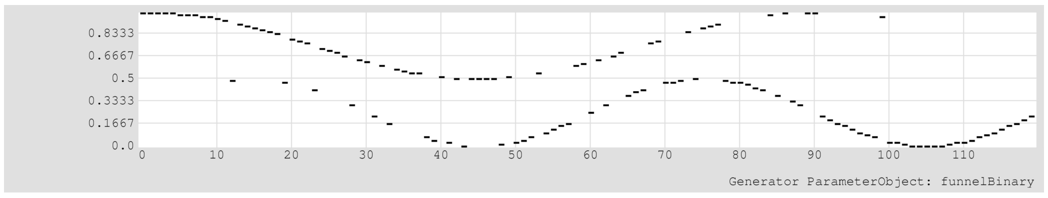
funnelBinary, thresholdMatchString, parameterObject, parameterObject, parameterObject, parameterObject

Description: A dynamic, two-part variable funnel. 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}, (6) parameterObject {generator of masked values}

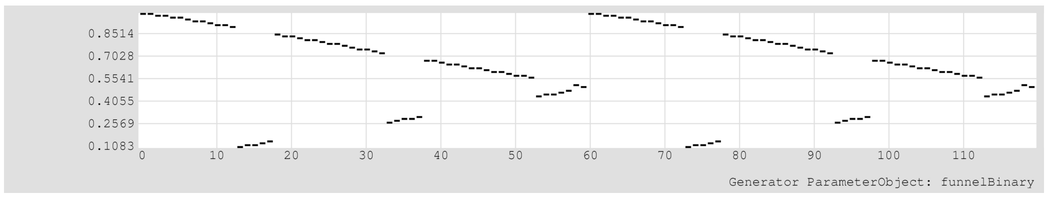
Sample Arguments: `fb, u, (bpl,e,s,((0,0),(120,1))), (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), (ru,0,1)`

Example C-41. funnelBinary Demonstration 1



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

Example C-42. funnelBinary Demonstration 2



```
funnelBinary, match, (constant, 0.2), (breakPointLinear, event, loop,
((0,0),(60,0.5))), (breakPointLinear, event, loop, ((0,1),(60,0.5))),
(waveSine, event, (constant, 20), 0, (constant, 0), (constant, 1))
```

C.1.24. feedbackModelLibrary (fml)

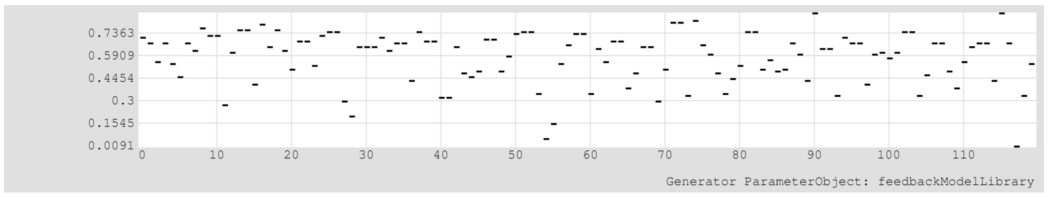
feedbackModelLibrary, feedbackModelName, parameterObject, parameterObject, min, max

Description: A model of a feedback system made from discrete particles. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) feedbackModelName, (3) parameterObject {aging step}, (4) parameterObject {threshold}, (5) min, (6) max

Sample Arguments: `fml, cc, (bg,rc,(1,3)), (c,0.9), 0, 1`

Example C-43. feedbackModelLibrary Demonstration 1



```
feedbackModelLibrary, climateControl, (basketGen, randomChoice, (1,3)),
(constant, 0.9), (constant, 0), (constant, 1)
```

C.1.25. fibonacciSeries (fs)

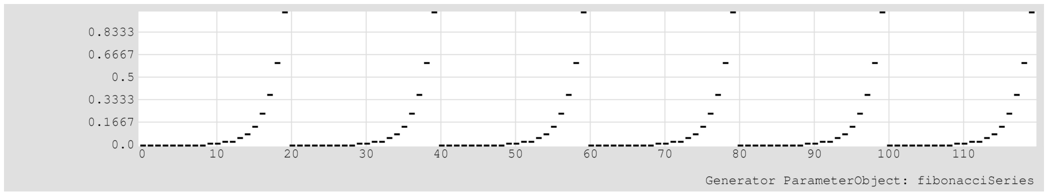
fibonacciSeries, start, length, min, max, selectionString

Description: Provides values derived from a contigous section of the Fibonacci series. A section is built from an initial value (start) and as many additional values as specified by the length argument. Negative length values reverse the direction of the series. 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'}

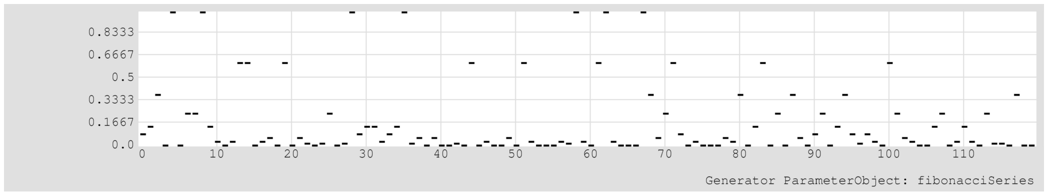
Sample Arguments: `fs, 200, 20, 0, 1, oc`

Example C-44. fibonacciSeries Demonstration 1



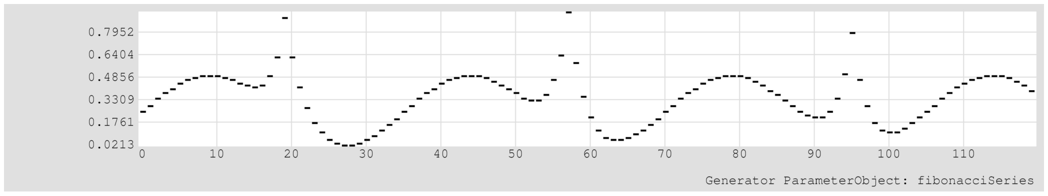
```
fibonacciSeries, 200, 20, (constant, 0), (constant, 1), orderedCyclic
```

Example C-45. fibonacciSeries Demonstration 2



```
fibonacciSeries, 40, 20, (constant, 0), (constant, 1), randomChoice
```

Example C-46. fibonacciSeries Demonstration 3



```
fibonacciSeries, 400, 20, (waveSine, event, (constant, 35), 0, (constant,
0.5), (constant, 0)), (cyclicGen, upDown, 0.6, 1, 0.03), orderedOscillate
```

C.1.26. grammarTerminus (gt)

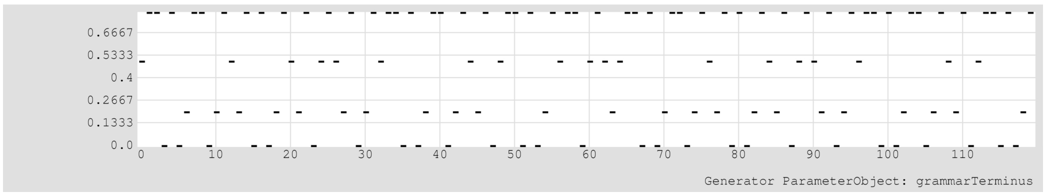
grammarTerminus, grammarString, stepCount, selectionString

Description: Produces values from a one-dimensional string rewrite rule, or Lindenmayer-system generative grammar. The terminus, or final result of the number of generations of values specifed by the stepCount parameter, is used to produce a list of defined values. Values are chosen from this list using the selector specified by the selectionString argument.

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

Sample Arguments: `gt, a{.2}b{.5}c{.8}d{0}@a{ba}b{bc}c{cd}d{ac}@a, 6, oc`

Example C-47. grammarTerminus Demonstration 1



`grammarTerminus, a{.2}b{.5}c{.8}d{0}@a{ba}c{cd}b{bc}d{ac}@a, 6, orderedCyclic`

C.1.27. henonBasket (hb)

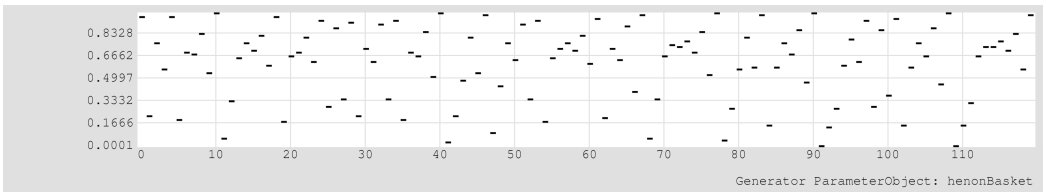
henonBasket, xInit, yInit, parameterObject, parameterObject, valueCount, valueSelect, min, max, selectionString

Description: Performs the Henon map, a non-linear two-dimensional discrete deterministic dynamical system. For some parameter settings the system exhibits chaotic behavior, for others, periodic behavior; small changes in initial parameters may demonstrate the butterfly effect. Variables x and y describe coordinate positions; values a (alpha) and b (beta) configure the system. As the output range cannot be predicted, as many values as specified by the valueCount argument, as well as any combination of variables with the valueSelect argument, are generated and stored at initialization. These values are then scaled 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. Note: some values may cause unexpected results; alpha values should not exceed 2.0.

Arguments: (1) name, (2) xInit, (3) yInit, (4) parameterObject {a value}, (5) parameterObject {b value}, (6) valueCount, (7) valueSelect {'x', 'y', 'xy', 'yx'}, (8) min, (9) max, (10) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

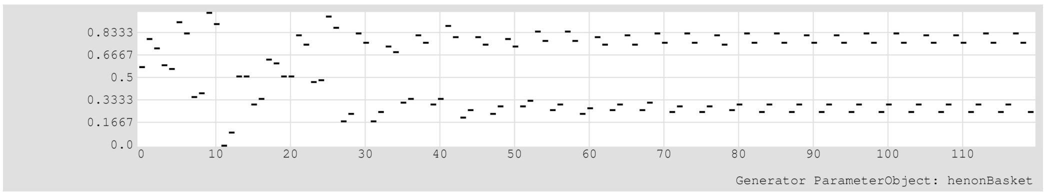
Sample Arguments: `hb, 0.5, 0.5, 1.4, 0.3, 1000, x, 0, 1, oc`

Example C-48. henonBasket Demonstration 1



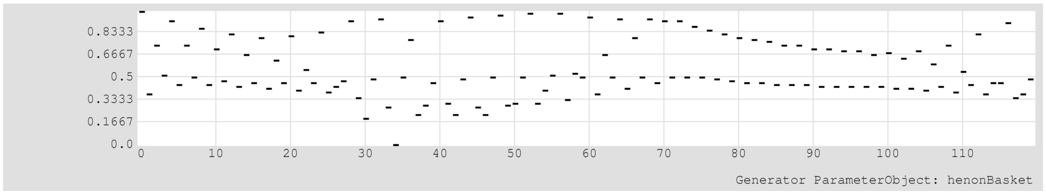
```
henonBasket, 0.5, 0.5, (constant, 1.4), (constant, 0.3), 1000, x, (constant,
0), (constant, 1), orderedCyclic
```

Example C-49. henonBasket Demonstration 2



```
henonBasket, 0.5, 0.5, (constant, 0.5), (constant, 0.8), 1000, yx, (constant,
0), (constant, 1), orderedCyclic
```

Example C-50. henonBasket Demonstration 3



```
henonBasket, 0.5, 0.5, (cyclicGen, upDown, 0, 0.9, 0.05), (constant, 0.3),
1000, xy, (constant, 0), (constant, 1), orderedCyclic
```

C.1.28. iterateCross (ic)

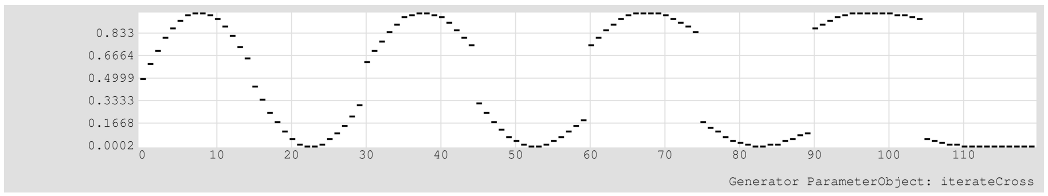
iterateCross, parameterObject, parameterObject, parameterObject

Description: Produces a single value cross faded between two values created by two Generator ParameterObjects in parallel. The cross fade is expressed as a number within the unit interval, where a value of zero is the output of the first Generator, a value of one is the output of the second Generator, and all other values are proportionally and linearly cross faded.

Arguments: (1) name, (2) parameterObject {first source Generator}, (3) parameterObject {second source Generator}, (4) parameterObject {interpolation between first and second Generator}

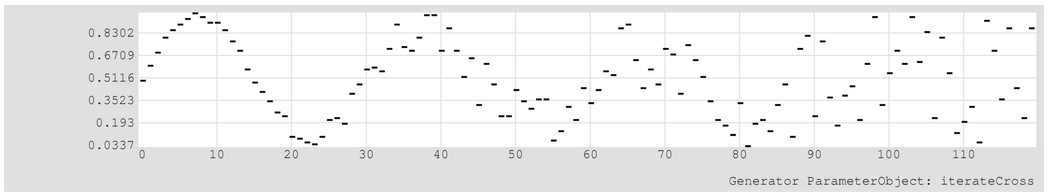
Sample Arguments: `ic, (ws,e,30,0,0,1), (wp,e,30,0,0,1), (bpl,e,l,((0,0),(120,1)))`

Example C-51. iterateCross Demonstration 1



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

Example C-52. iterateCross Demonstration 2



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

C.1.29. iterateGroup (ig)

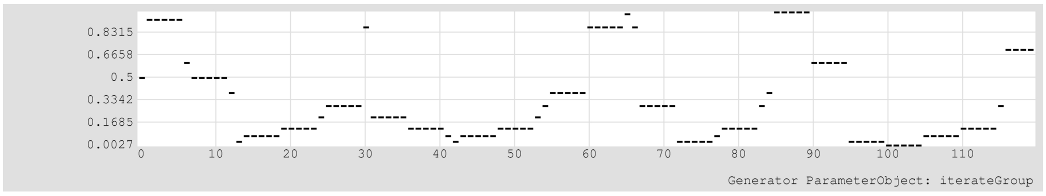
iterateGroup, parameterObject, parameterObject

Description: Allows the output of a source 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 Generator}, (3) parameterObject {group or skip control Generator}

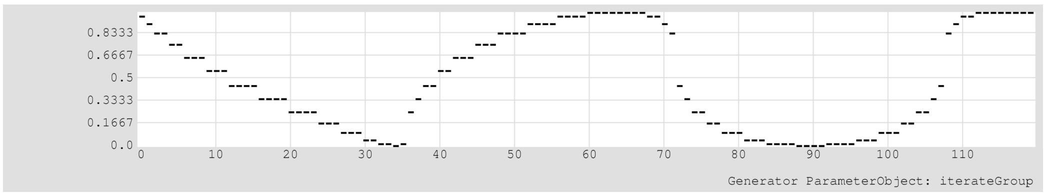
Sample Arguments: `ig, (ws,e,30,0,0,1), (bg,rc,(-3,1,-1,5))`

Example C-53. iterateGroup Demonstration 1



```
iterateGroup, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1)), (basketGen, randomChoice, (-3,1,-1,5))
```

Example C-54. iterateGroup Demonstration 2



```
iterateGroup, (waveCosine, event, (constant, 30), 0, (constant, 0), (constant,
1)), (waveTriangle, event, (constant, 20), 0, (constant, 4), (constant, -1))
```

C.1.30. iterateHold (ih)

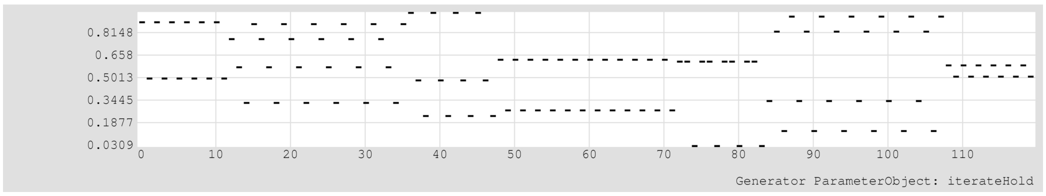
iterateHold, parameterObject, parameterObject, parameterObject, selectionString

Description: Allows a variable number of outputs from a source 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 Generator}, (3) parameterObject {size Generator}, (4) parameterObject {refresh count Generator}, (5) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

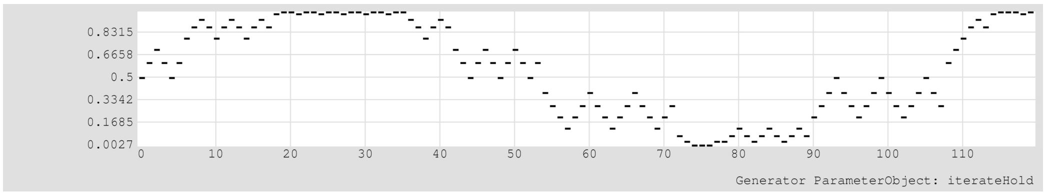
Sample Arguments: `ih, (ru,0,1), (bg,rc,(2,3,4)), (bg,oc,(12,24)), oc`

Example C-55. iterateHold Demonstration 1



```
iterateHold, (randomUniform, (constant, 0), (constant, 1)), (basketGen,
randomChoice, (2,3,4)), (basketGen, orderedCyclic, (12,24)), orderedCyclic
```

Example C-56. iterateHold Demonstration 2



```
iterateHold, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1)), (basketGen, randomChoice, (3,4,5)), (basketGen, orderedCyclic,
(6,12,18)), orderedOscillate
```

C.1.31. iterateSelect (is)

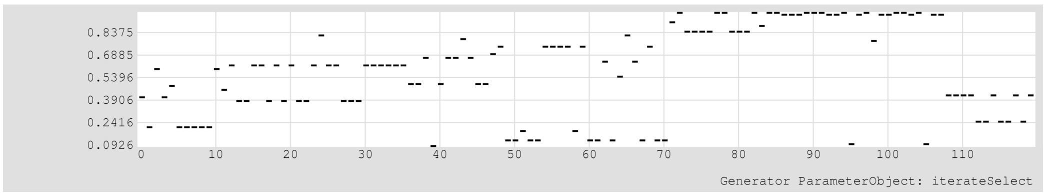
iterateSelect, parameterObject, parameterObject, parameterObject, parameterObject

Description: Allows a variable number of outputs from a source ParameterObject, collected and stored in a list, to be selected with values within the unit interval produced by an embedded ParameterObject. Values that exceed the unit interval are limited within the unit interval. 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 Generator}, (3) parameterObject {size Generator}, (4) parameterObject {refresh count Generator}, (5) parameterObject {selection Generator}

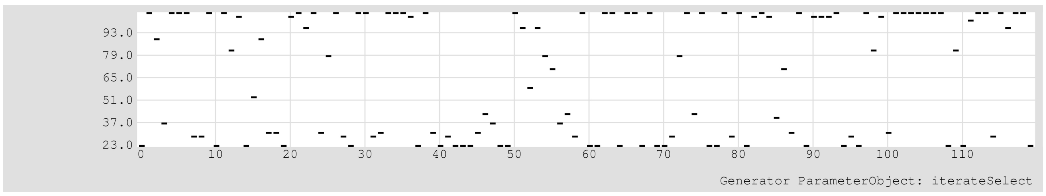
Sample Arguments: `is, (ru,0,1), (bg,rc,(10,11,12)), (bg,oc,(12,24)), (rb,0.15,0.15,0,1)`

Example C-57. iterateSelect Demonstration 1



```
iterateSelect, (randomUniform, (constant, 0), (constant, 1)), (basketGen,
randomChoice, (10,11,12)), (basketGen, orderedCyclic, (12,24)), (randomBeta,
0.15, 0.15, (constant, 0), (constant, 1))
```

Example C-58. iterateSelect Demonstration 2



```
iterateSelect, (listPrime, 20, 20, integer, orderedCyclic), (constant, 20),
(constant, 20), (randomBeta, 0.2, 0.2, (constant, 0), (constant, 1))
```

C.1.32. iterateWindow (iw)

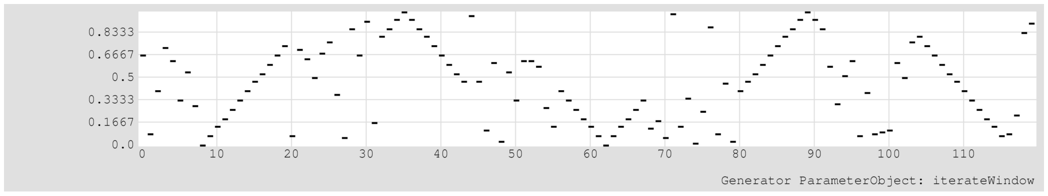
iterateWindow, parameterObjectList, parameterObject, selectionString

Description: Allows a ParameterObject, selected from a list of 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 Generators}, (3) parameterObject {generate or skip control Generator}, (4) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

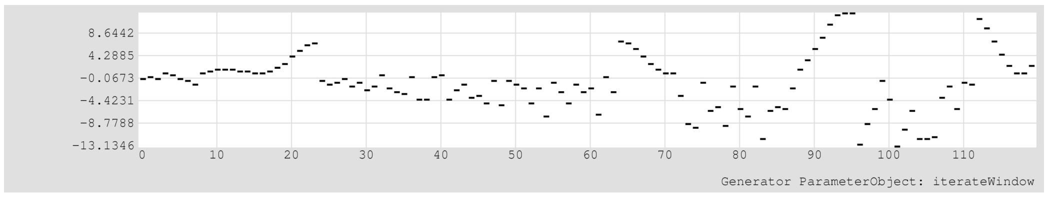
Sample Arguments: `iw, ((ru,0,1),(wt,e,30,0,0,1)), (bg,oc,(8,4,-2)), oc`

Example C-59. iterateWindow Demonstration 1



```
iterateWindow, ((randomUniform, (constant, 0), (constant, 1)), (waveTriangle,
event, (constant, 30), 0, (constant, 0), (constant, 1))), (basketGen,
orderedCyclic, (8,4,-2)), orderedCyclic
```

Example C-60. iterateWindow Demonstration 2



```
iterateWindow, ((randomUniform, (constant, 1), (accumulator, 0, (constant,
-0.2))), (waveSine, event, (constant, 15), 0.25, (accumulator, 1, (constant,
0.4)), (constant, 1))), (basketGen, orderedCyclic, (8,8,-11)), randomChoice
```

C.1.33. lorenzBasket (lb)

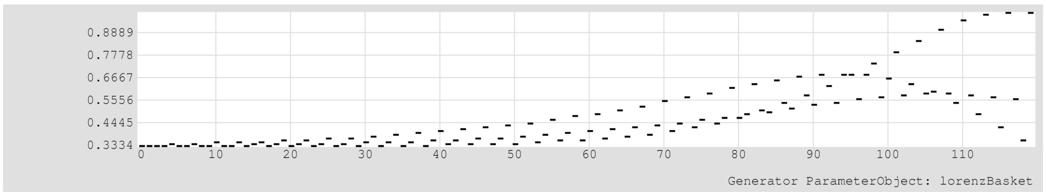
lorenzBasket, xInit, yInit, zInit, parameterObject, parameterObject, parameterObject, valueCount, valueSelect, min, max, selectionString

Description: Performs the Lorenz attractor, a non-linear three-dimensional discrete deterministic dynamical system. The equations are derived from a simplified model of atmospheric convection rolls. For some parameter settings the system exhibits chaotic behavior, for others, periodic behavior; small changes in initial parameters may demonstrate the butterfly effect. Variables x, y, and z are proportional to convective intensity, temperature difference between descending and ascending currents, and the difference in vertical temperature profile from linearity. Values s (sigma), r, and b are the Prandtl number, the quotient of the Rayleigh number and the critical Rayleigh number, and the geometric factor. As the output range cannot be predicted, as many values as specified by the valueCount argument, as well as any combination of variables with the valueSelect argument, are generated and stored at initialization. These values are then scaled 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. Note: some values may cause unexpected results; r should not exceed 90.

Arguments: (1) name, (2) xInit, (3) yInit, (4) zInit, (5) parameterObject {r value}, (6) parameterObject {s value}, (7) parameterObject {b value}, (8) valueCount, (9) valueSelect {'x', 'y', 'z', 'xy', 'xz', 'yx', 'yz', 'zx', 'zy', 'xyz', 'xzy', 'yxz', 'yzx', 'zxy', 'zyx'}, (10) min, (11) max, (12) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

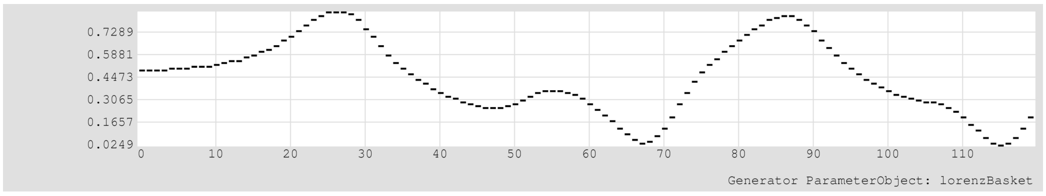
Sample Arguments: `lb, 1.0, 1.0, 1.0, 28, 10, 2.67, 1000, xyz, 0, 1, oc`

Example C-61. lorenzBasket Demonstration 1



```
lorenzBasket, 1.0, 1.0, 1.0, (constant, 28), (constant, 10), (constant, 2.67),
1000, xyz, (constant, 0), (constant, 1), orderedCyclic
```

Example C-62. lorenzBasket Demonstration 2



```
lorenzBasket, 0.5, 1.5, 10, (cyclicGen, down, 1, 80, 1.5), (constant, 10),
(constant, 12.4), 1000, x, (constant, 0), (constant, 1), orderedCyclic
```

C.1.34. logisticMap (lm)

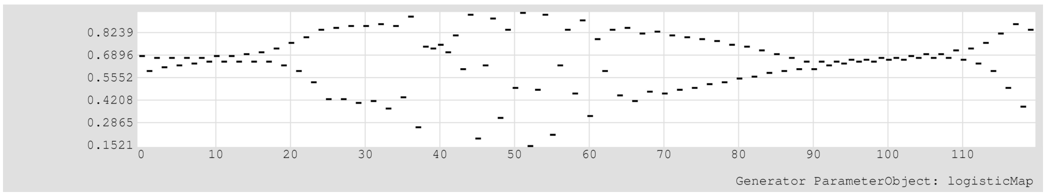
logisticMap, initValue, parameterObject, min, max

Description: Performs the logistic map, or the Verhulst population growth equation. The logistic map is a non-linear one-dimensional discrete deterministic dynamical system. For some parameter settings the system exhibits chaotic behavior, for others, periodic behavior; small changes in initial parameters may demonstrate the butterfly effect. Variable x represents the population value; value p represents a combined rate for reproduction and starvation. The p argument allows the user to provide a static or dynamic value to the equation. Certain p-value presets can be provided with strings: 'bi', 'quad', or 'chaos'. If a number is provided for p, the value will be used to create a constant ParameterObject. The equation outputs values within the unit interval. These values are scaled within the range designated by min and max; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) initValue, (3) parameterObject {p value}, (4) min, (5) max

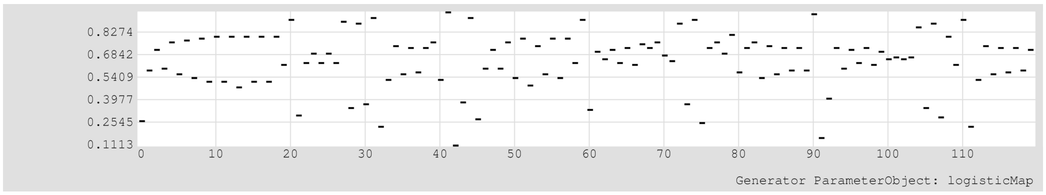
Sample Arguments: `lm, 0.5, (wt,e,90,0,2.75,4), 0, 1`

Example C-63. logisticMap Demonstration 1



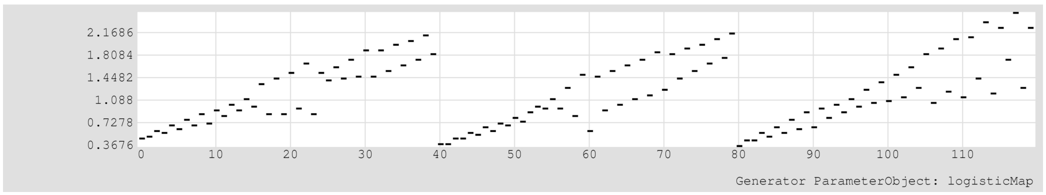
```
logisticMap, 0.5, (waveTriangle, event, (constant, 90), 0, (constant, 2.75),
(constant, 4)), (constant, 0), (constant, 1)
```

Example C-64. logisticMap Demonstration 2



```
logisticMap, 0.1, (basketGen, randomWalk, (3,3,3,3.2,3.2,3.2,3.9,3.9,3.9)),
(constant, 0), (constant, 1)
```

Example C-65. logisticMap Demonstration 3



```
logisticMap, 0.5, (iterateGroup, (basketGen, randomChoice, (3,3.2,3.57)),
(basketGen, randomChoice, (5,7,9))), (breakPointLinear, event, loop,
((0,0.5),(60,0),(120,0.5))), (breakPointLinear, event, loop, ((0,0.5),(40,3)))
```

C.1.35. listPrime (lp)

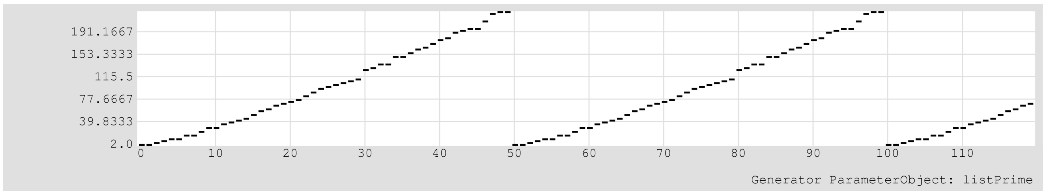
listPrime, start, length, format, selectionString

Description: Produces a segment of prime (pseudoprime) integers defined by a positive or negative start value and a length. 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) start, (3) length, (4) format {'integer', 'width', 'unit', 'binary'}, (5) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

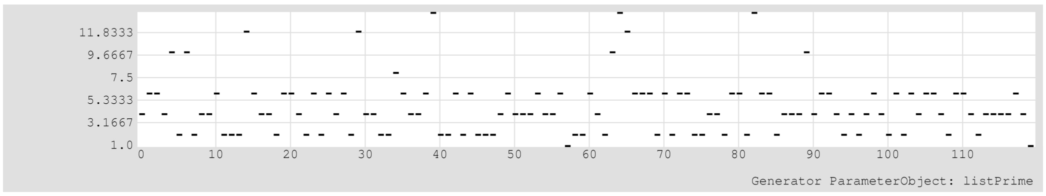
Sample Arguments: `lp, 2, 50, int, oc`

Example C-66. listPrime Demonstration 1



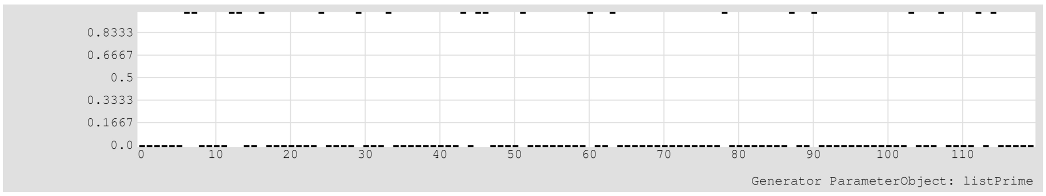
```
listPrime, 2, 50, integer, orderedCyclic
```

Example C-67. listPrime Demonstration 2



```
listPrime, -100, 100, width, randomChoice
```

Example C-68. listPrime Demonstration 3



```
listPrime, 200, -30, binary, randomPermutate
```

C.1.36. lineSegment (ls)

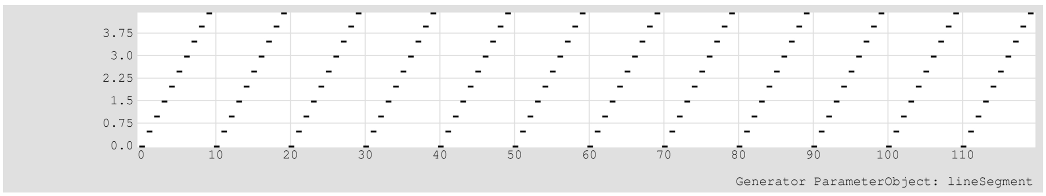
lineSegment, stepString, parameterObject, min, max

Description: Provides a dynamic line segment created from three embedded Generator ParameterObjects. Start and end values, taken from min and max generators, are used to create a line segment spaning the time or event distance provided by the secPerCycle argument. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle).

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

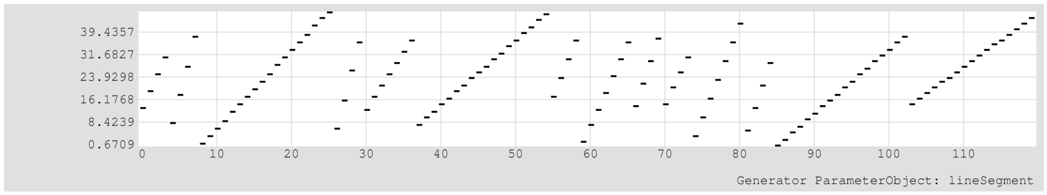
Sample Arguments: `ls, e, 10, 0, 5`

Example C-69. lineSegment Demonstration 1



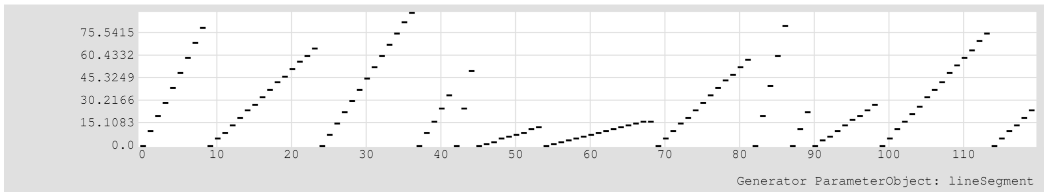
```
lineSegment, (constant, 10), (constant, 0), (constant, 5)
```

Example C-70. lineSegment Demonstration 2



```
lineSegment, (basketGen, randomChoice, (4,7,18)), (randomUniform, (constant,
0), (constant, 20)), (randomUniform, (constant, 30), (constant, 50))
```

Example C-71. lineSegment Demonstration 3



```
lineSegment, (waveSine, event, (constant, 5), 0, (constant, 2), (constant,
15)), (constant, 0), (randomUniform, (constant, 0), (constant, 100))
```

C.1.37. mask (m)

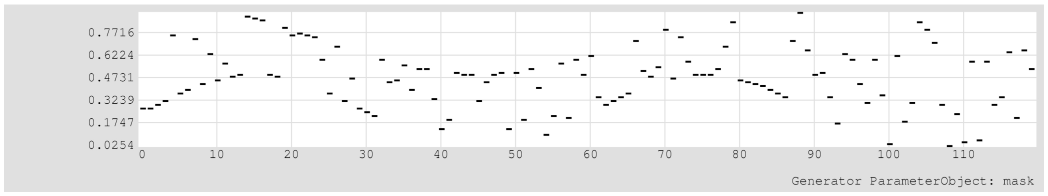
mask, boundaryString, parameterObject, parameterObject, parameterObject

Description: Given values produced by two boundary ParameterObjects in parallel, the Generator ParameterObject value is fit within these values. 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}, (5) parameterObject {generator of masked values}

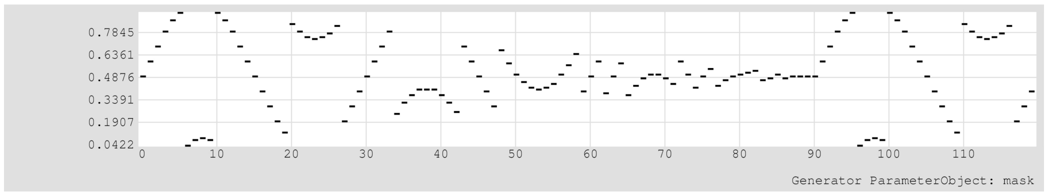
Sample Arguments: `m, l, (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), (ru,0,1)`

Example C-72. mask Demonstration 1



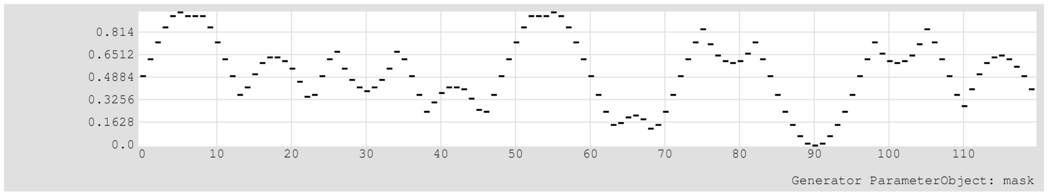
```
mask, limit, (waveSine, event, (constant, 60), 0, (constant, 0.5), (constant,
0)), (waveCosine, event, (constant, 90), 0, (constant, 0.5), (constant, 1)),
(randomUniform, (constant, 0), (constant, 1))
```

Example C-73. mask Demonstration 2



```
mask, wrap, (breakPointLinear, event, loop, ((0,0),(90,0.5))),
(breakPointLinear, event, loop, ((0,1),(90,0.5))), (waveSine, event,
(constant, 30), 0, (constant, 0), (constant, 1))
```

Example C-74. mask Demonstration 3



```
mask, reflect, (waveSine, event, (constant, 60), 0.25, (constant, 0.7),
(constant, 1)), (breakPointLinear, event, loop, ((0,0.4),(90,0),(120,0.4))),
(waveSine, event, (constant, 24), 0, (constant, 0), (constant, 1))
```

C.1.38. markovGeneratorAnalysis (mga)

markovGeneratorAnalysis, parameterObject, valueCount, maxAnalysisOrder, parameterObject

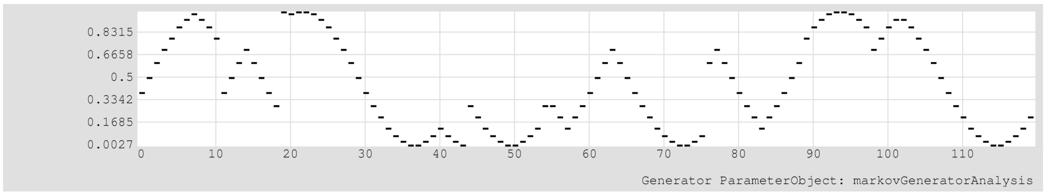
Description: Produces values by means of a Markov analysis of values provided by a source Generator ParameterObject; the analysis of these values is used with a dynamic transition order

Generator to produce new values. The number of values drawn from the source Generator is specified with the valueCount 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 Generator}, (3) valueCount, (4) maxAnalysisOrder, (5) parameterObject {output order value}

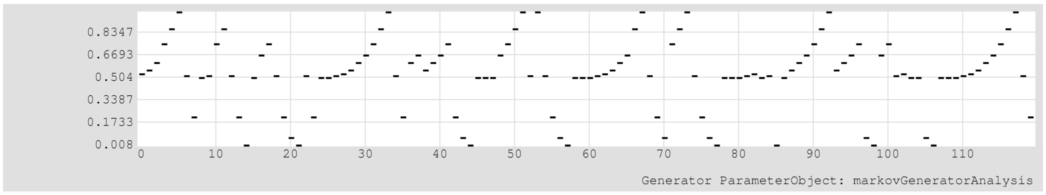
Sample Arguments: `mga, (ws,e,30,0,0,1), 30, 2, (mv,a{1}b{0}c{2}:{a=10|b=1|c=2},(c,0))`

Example C-75. markovGeneratorAnalysis Demonstration 1



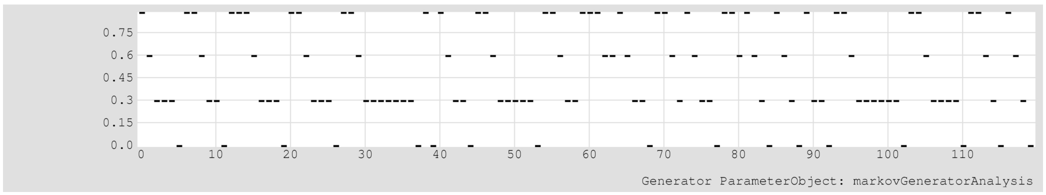
```
markovGeneratorAnalysis, (waveSine, event, (constant, 30), 0, (constant, 0),
(constant, 1)), 30, 2, (markovValue, a{1}b{0}c{2}:{a=10|b=1|c=2}, (constant,
0))
```

Example C-76. markovGeneratorAnalysis Demonstration 2



```
markovGeneratorAnalysis, (breakPointPower, event, loop,
((0,0.5),(10,1),(15,0)), 2), 15, 2, (basketGen, randomWalk, (0,1,2,2,1))
```

Example C-77. markovGeneratorAnalysis Demonstration 3



```
markovGeneratorAnalysis, (basketGen, orderedCyclic,
(0.3,0.3,0.3,0,0.9,0.9,0.6)), 28, 2, (markovValue,
a{1}b{0}c{2}:{a=10|b=1|c=2}, (constant, 0))
```

C.1.39. maskReject (mr)

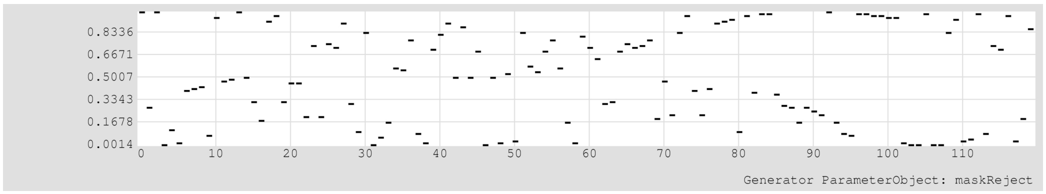
maskReject, boundaryString, parameterObject, parameterObject, parameterObject

Description: Given values produced by two boundary ParameterObjects in parallel, the Generator ParameterObject value is fit outside of these values. 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}, (5) parameterObject {generator of masked values}

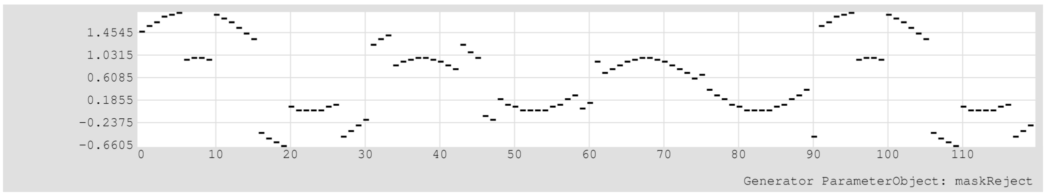
Sample Arguments: `mr, l, (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), (ru,0,1)`

Example C-78. maskReject Demonstration 1



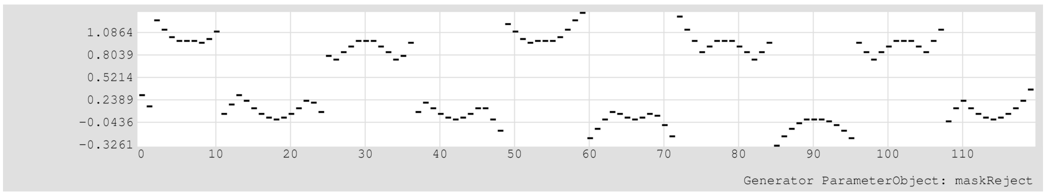
```
maskReject, limit, (waveSine, event, (constant, 60), 0, (constant, 0.5),
(constant, 0)), (waveCosine, event, (constant, 90), 0, (constant, 0.5),
(constant, 1)), (randomUniform, (constant, 0), (constant, 1))
```

Example C-79. maskReject Demonstration 2



```
maskReject, wrap, (breakPointLinear, event, loop, ((0,0),(90,0.5))),
(breakPointLinear, event, loop, ((0,1),(90,0.5))), (waveSine, event,
(constant, 30), 0, (constant, 0), (constant, 1))
```

Example C-80. maskReject Demonstration 3



```
maskReject, reflect, (waveSine, event, (constant, 60), 0.25, (constant, 0.7),
(constant, 1)), (breakPointLinear, event, loop, ((0,0.4),(90,0),(120,0.4))),
(waveSine, event, (constant, 24), 0, (constant, 0), (constant, 1))
```

C.1.40. maskScale (ms)

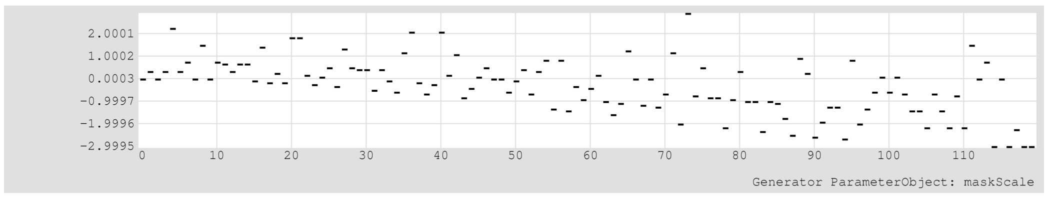
maskScale, parameterObject, valueCount, min, max, selectionString

Description: Given values produced by two boundary ParameterObjects in parallel, the Generator ParameterObject value is scaled within these values. A collection of values created by the Generator ParameterObject are stored. 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) parameterObject {source Generator}, (3) valueCount, (4) min, (5) max, (6) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `ms, (lp,100,120,w,oc), 120, (bphc,e,l,((0,0),(120,-3))), 3, oc`

Example C-81. maskScale Demonstration 1



```
maskScale, (listPrime, 100, 120, width, orderedCyclic), 120,
(breakPointHalfCosine, event, loop, ((0,0),(120,-3))), (constant, 3),
orderedCyclic
```

C.1.41. markovValue (mv)

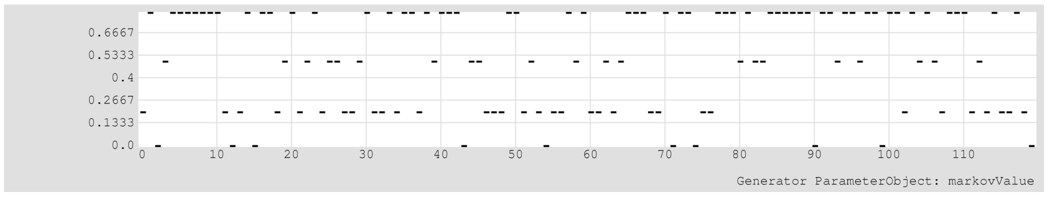
markovValue, transitionString, parameterObject

Description: Produces values by means of a Markov transition string specification and a dynamic transition order generator. 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}

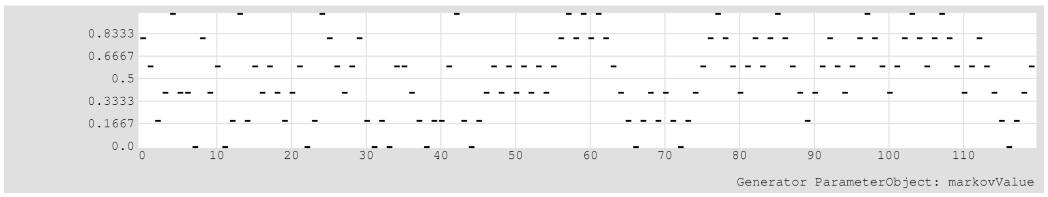
Sample Arguments: `mv, a{.2}b{.5}c{.8}d{0}:{a=5|b=4|c=7|d=1}, (c,0)`

Example C-82. markovValue Demonstration 1



```
markovValue, a{.2}b{.5}c{.8}d{0}:{a=5|b=4|c=7|d=1}, (constant, 0)
```

Example C-83. markovValue Demonstration 2



```
markovValue, a{0}b{.2}c{.4}d{.6}e{.8}f{1}:{a=3|b=6|c=8|d=8|e=5|f=2}a:{b=3}b:{a
=2|c=4}c:{b=3|d=5}d:{a=1|c=4|e=3}e:{d=3|f=2}f:{e=2}a:b:{c=3}b:a:{b=2}b:c:{d=4}
c:b:{a=2|c=1}c:d:{a=1|c=1|e=3}d:a:{b=1}d:c:{b=3|d=1}d:e:{d=1|f=2}e:d:{c=3}e:f:
{e=2}f:e:{d=2}, (breakPointLinear, event, single, ((0,0),(119,2)))
```

C.1.42. noise (n)

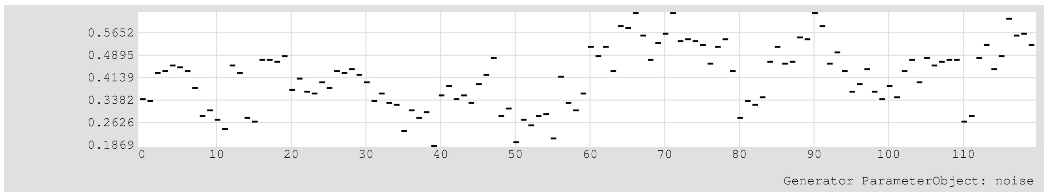
noise, resolution, parameterObject, min, max

Description: Fractional noise (1/fn) Generator, capable of producing states and transitions between 1/f white, pink, brown, and black noise. Resolution is an integer that describes how many generators are used. The gamma argument determines what type of noise is created. All gamma values are treated as negative. A gamma of 0 is white noise; a gamma of 1 is pink noise; a gamma of 2 is brown noise; and anything greater is black noise. Gamma can be controlled by a dynamic ParameterObject. The value produced by the noise generator is scaled within the unit interval. This normalized value is then scaled within the range designated by min and max; min and max may be specified by ParameterObjects.

Arguments: (1) name, (2) resolution, (3) parameterObject {gamma value as string or number}, (4) min, (5) max

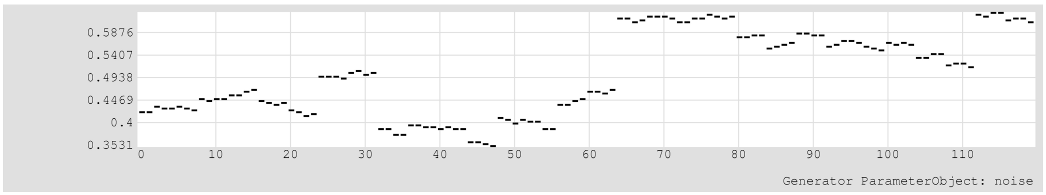
Sample Arguments: `n, 100, pink, 0, 1`

Example C-84. noise Demonstration 1



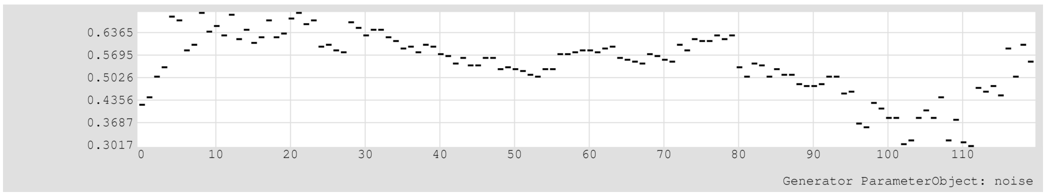
```
noise, 100, (constant, 1), (constant, 0), (constant, 1)
```

Example C-85. noise Demonstration 2



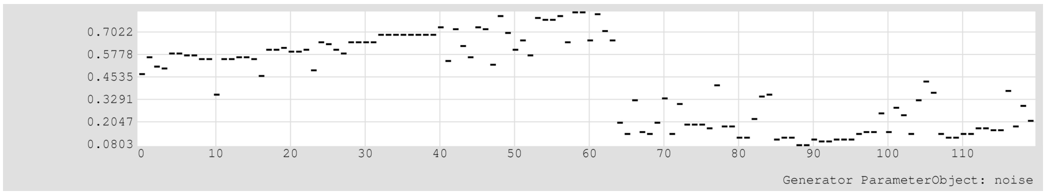
```
noise, 100, (constant, 3), (constant, 0), (constant, 1)
```

Example C-86. noise Demonstration 3



```
noise, 100, (waveTriangle, event, (constant, 120), 0, (constant, 1),
(constant, 3)), (constant, 0), (constant, 1)
```

Example C-87. noise Demonstration 4



```
noise, 100, (basketGen, randomChoice, (3,3,3,3,2,1)), (constant, 0),
(constant, 1)
```

C.1.43. operatorAdd (oa)

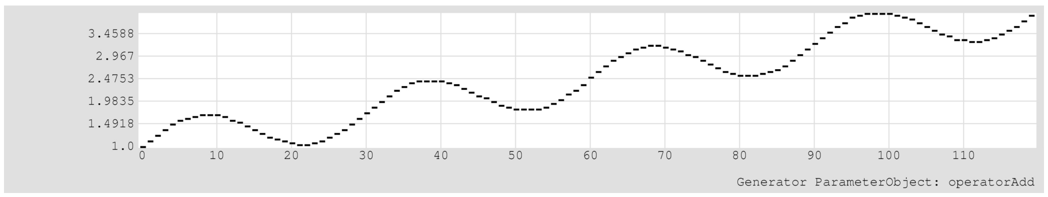
operatorAdd, parameterObject, parameterObject

Description: Adds the value of the first ParameterObject to the second ParameterObject.

Arguments: (1) name, (2) parameterObject {first value}, (3) parameterObject {second value}

Sample Arguments: `oa, (ws,e,30,0,0,1), (a,0.5,(c,0.03))`

Example C-88. operatorAdd Demonstration 1



```
operatorAdd, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1)), (accumulator, 0.5, (constant, 0.03))
```

C.1.44. operatorCongruence (oc)

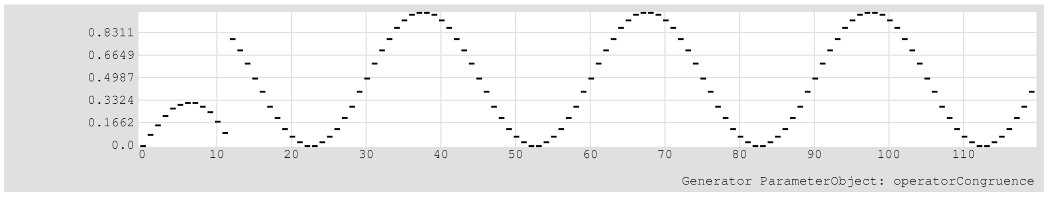
operatorCongruence, parameterObject, parameterObject

Description: Produces the congruent value of the first ParameterObject object as the modulus of the second ParameterObject. A modulus by zero, if encountered, returns the value of the first ParameterObject unaltered.

Arguments: (1) name, (2) parameterObject {first value}, (3) parameterObject {second value}

Sample Arguments: `oc, (ws,e,30,0,0,1), (a,0.5,(c,0.03))`

Example C-89. operatorCongruence Demonstration 1



```
operatorCongruence, (waveSine, event, (constant, 30), 0, (constant, 0),
(constant, 1)), (accumulator, 0.5, (constant, 0.03))
```

C.1.45. operatorDivide (od)

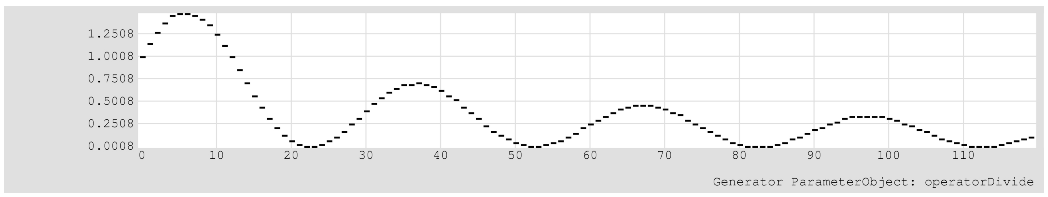
operatorDivide, parameterObject, parameterObject

Description: Divides the value of the first ParameterObject object by the second ParameterObject. Division by zero, if encountered, returns the value of the first Generator.

Arguments: (1) name, (2) parameterObject {first value}, (3) parameterObject {second value}

Sample Arguments: `od, (ws,e,30,0,0,1), (a,0.5,(c,0.03))`

Example C-90. operatorDivide Demonstration 1



```
operatorDivide, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1)), (accumulator, 0.5, (constant, 0.03))
```

C.1.46. operatorMultiply (om)

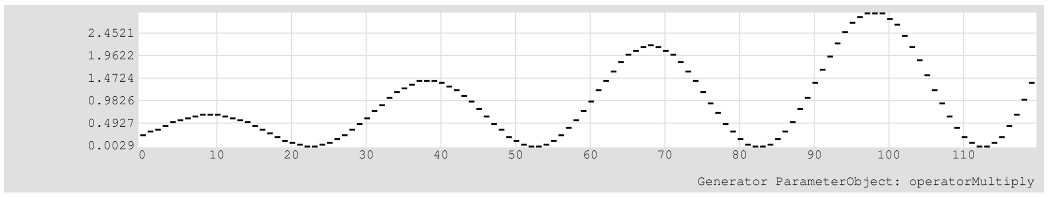
operatorMultiply, parameterObject, parameterObject

Description: Multiplies the value of the first ParameterObject by the second.

Arguments: (1) name, (2) parameterObject {first value}, (3) parameterObject {second value}

Sample Arguments: `om, (ws,e,30,0,0,1), (a,0.5,(c,0.03))`

Example C-91. operatorMultiply Demonstration 1



```
operatorMultiply, (waveSine, event, (constant, 30), 0, (constant, 0),
(constant, 1)), (accumulator, 0.5, (constant, 0.03))
```

C.1.47. oneOver (oo)

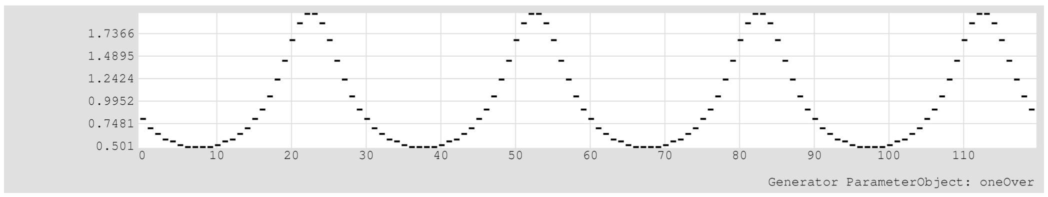
oneOver, parameterObject

Description: Produces the value of one over the value of a ParameterObject. Divisors of zero are resolved to 1.

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

Sample Arguments: `oo, (ws,e,30,0,0.5,2)`

Example C-92. oneOver Demonstration 1



```
oneOver, (waveSine, event, (constant, 30), 0, (constant, 0.5), (constant, 2))
```

C.1.48. operatorPower (op)

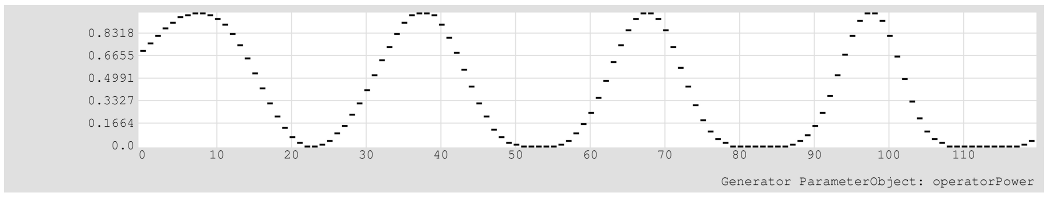
operatorPower, parameterObject, parameterObject

Description: Raises the value of the first ParameterObject to the power of the second ParameterObject.

Arguments: (1) name, (2) parameterObject {first value}, (3) parameterObject {second value}

Sample Arguments: `op, (ws,e,30,0,0,1), (a,0.5,(c,0.03))`

Example C-93. operatorPower Demonstration 1



```
operatorPower, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1)), (accumulator, 0.5, (constant, 0.03))
```

C.1.49. operatorSubtract (os)

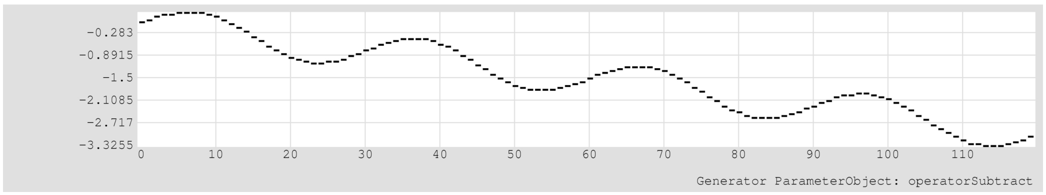
operatorSubtract, parameterObject, parameterObject

Description: Subtracts the value of the second ParameterObject from the first ParameterObject.

Arguments: (1) name, (2) parameterObject {first value}, (3) parameterObject {second value}

Sample Arguments: `os, (ws,e,30,0,0,1), (a,0.5,(c,0.03))`

Example C-94. operatorSubtract Demonstration 1



```
operatorSubtract, (waveSine, event, (constant, 30), 0, (constant, 0),
(constant, 1)), (accumulator, 0.5, (constant, 0.03))
```

C.1.50. pathRead (pr)

pathRead, pathFormatString

Description: Extracts pitch information from the current Multiset within a Texture's Path. Data can be presented in a variety of formats including representations of the Multiset as 'forte', 'mason', or data on the current active pitch as 'fq' (frequency), 'ps' (psReal), 'midi' (midi pitch values), 'pch' (Csound pitch octave format), or 'name' (alphabetic note names).

Arguments: (1) name, (2) pathFormatString {'forte', 'mason', 'fq', 'ps', 'midi', 'pch', 'name'}

Sample Arguments: `pr, forte`

C.1.51. quantize (q)

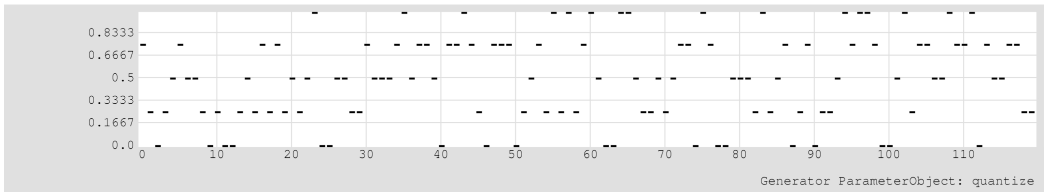
quantize, parameterObject, parameterObject, stepCount, parameterObject, parameterObject

Description: Dynamic grid size and grid position quantization. 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}, (6) parameterObject {source Generator}

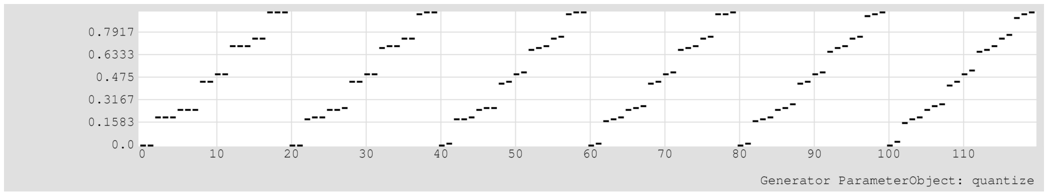
Sample Arguments: `q, (c,0), (c,0.25), 1, (c,1), (ru,0,1)`

Example C-95. quantize Demonstration 1



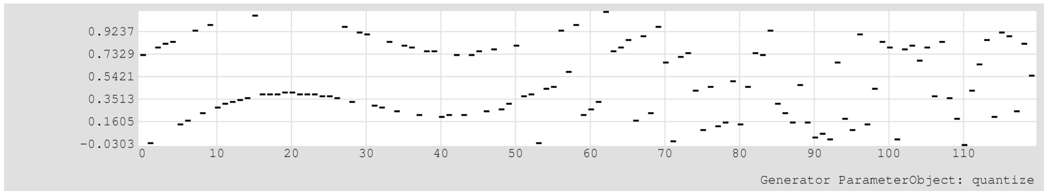
```
quantize, (constant, 0), (constant, 0.25), 1, (constant, 1), (randomUniform,
(constant, 0), (constant, 1))
```

Example C-96. quantize Demonstration 2



```
quantize, (constant, 0), (basketGen, orderedCyclic, (0.05,0.2)), 2,
(breakPointLinear, event, loop, ((0,1),(120,0.5))), (wavePowerUp, event,
(constant, 20), -2, 0, (constant, 0), (constant, 1))
```

Example C-97. quantize Demonstration 3



```
quantize, (waveSine, event, (constant, 60), 0, (constant, 1.25), (constant,
1.75)), (cyclicGen, upDown, 0.3, 0.9, 0.006), 1, (breakPointLinear, event,
loop, ((0,1),(40,1),(120,0.25))), (randomUniform, (constant, 0), (constant,
1))
```

C.1.52. randomBeta (rb)

randomBeta, alpha, beta, min, max

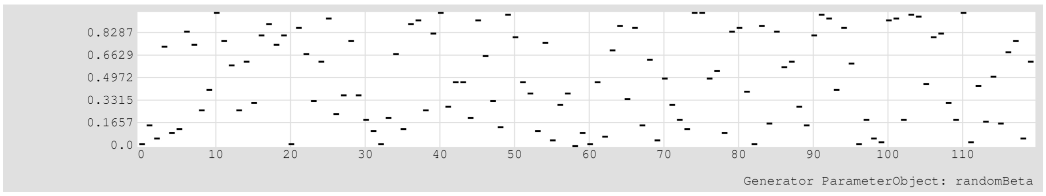
Description: Provides random numbers between 0 and 1 within a Beta distribution. This value is scaled within the range designated by min and max; min and max may be specified with

ParameterObjects. Alpha and beta values should be between 0 and 1; small alpha and beta values (such as 0.1) increase the probability of events at the boundaries.

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

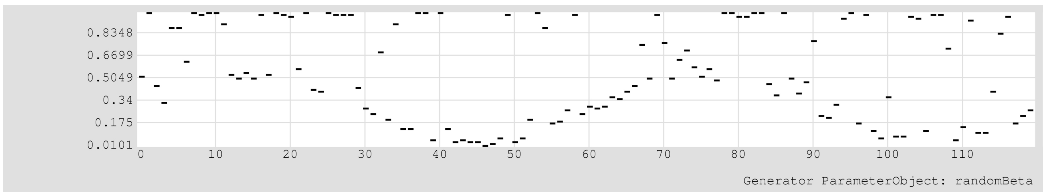
Sample Arguments: `rb, 0.5, 0.5, 0, 1`

Example C-98. randomBeta Demonstration 1



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

Example C-99. randomBeta Demonstration 2



```
randomBeta, 0.2, 0.2, (waveSine, event, (constant, 60), 0, (constant, 0),
(constant, 0.5)), (constant, 1)
```

C.1.53. randomBilateralExponential (rbe)

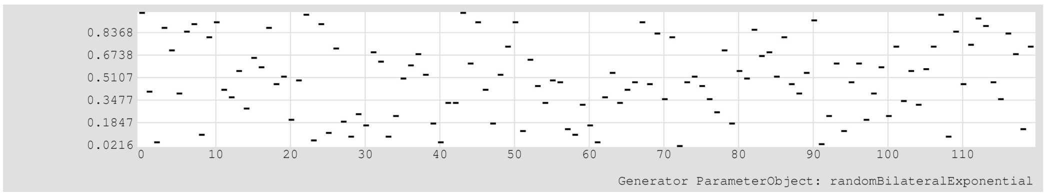
randomBilateralExponential, lambda, min, max

Description: Provides random numbers between 0 and 1 within a bilateral exponential distribution. 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

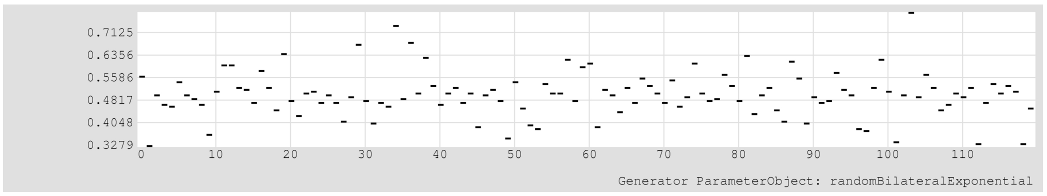
Sample Arguments: `rbe, 0.5, 0, 1`

Example C-100. randomBilateralExponential Demonstration 1



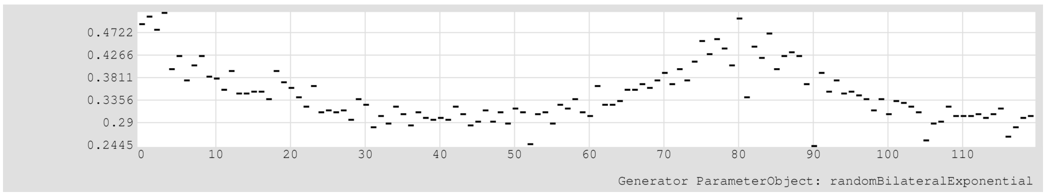
```
randomBilateralExponential, 0.5, (constant, 0), (constant, 1)
```

Example C-101. randomBilateralExponential Demonstration 2



```
randomBilateralExponential, 10.0, (constant, 0), (constant, 1)
```

Example C-102. randomBilateralExponential Demonstration 3



```
randomBilateralExponential, 20.0, (constant, 0), (breakPointPower, event,
loop, ((0,1),(40,0.6),(80,1)), 2)
```

C.1.54. randomCauchy (rc)

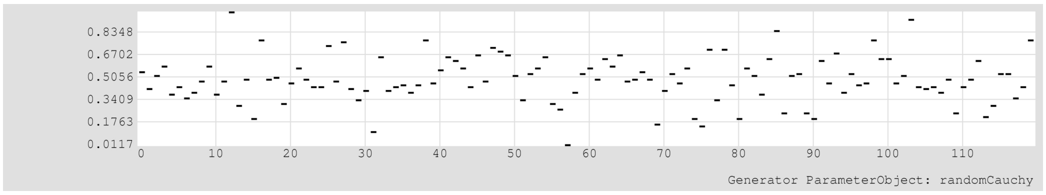
randomCauchy, alpha, mu, min, max

Description: Provides random numbers between 0 and 1 within a Cauchy 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 = 0.1, mu = 0.5.

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

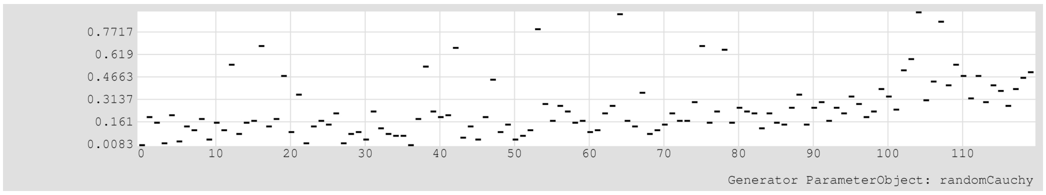
Sample Arguments: `rc, 0.1, 0.5, 0, 1`

Example C-103. randomCauchy Demonstration 1



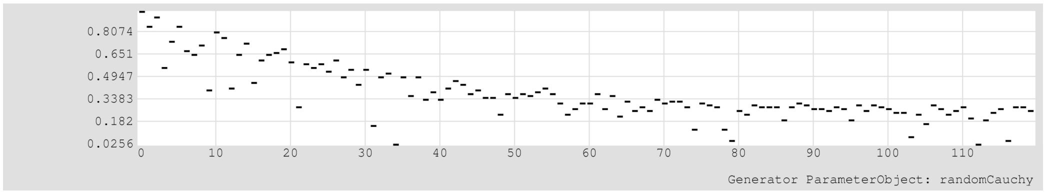
```
randomCauchy, 0.1, 0.5, (constant, 0), (constant, 1)
```

Example C-104. randomCauchy Demonstration 2



```
randomCauchy, 0.1, 0.1, (constant, 1), (breakPointPower, event, loop,
((0,0),(120,0.3)), 2)
```

Example C-105. randomCauchy Demonstration 3



```
randomCauchy, 0.1, 0.9, (constant, 0), (breakPointPower, event, loop,
((0,1),(120,0.3)), 2)
```

C.1.55. randomExponential (re)

randomExponential, lambda, min, max

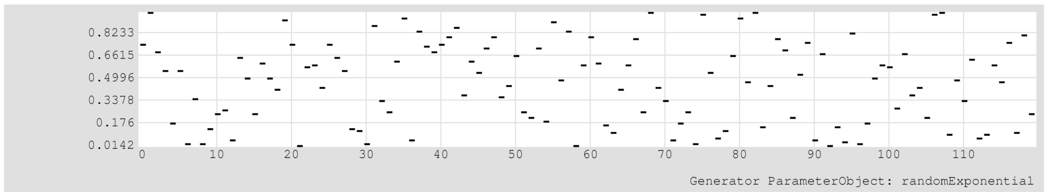
Description: Provides random numbers between 0 and 1 within an exponential distribution. This value is scaled within the range designated by min and max; min and max may be specified with

ParameterObjects. Lambda values should be greater than 0. Lambda values control the spread of values; larger values (such as 10) increase the probability of events near the minimum.

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

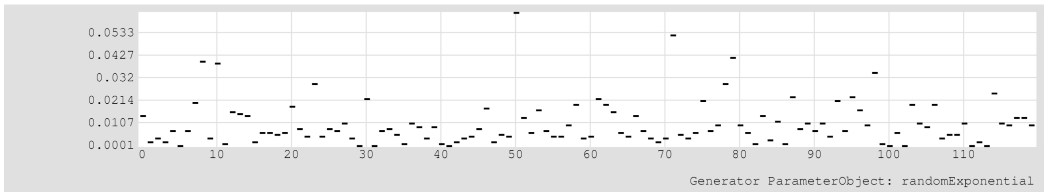
Sample Arguments: `re, 0.5, 0, 1`

Example C-106. randomExponential Demonstration 1



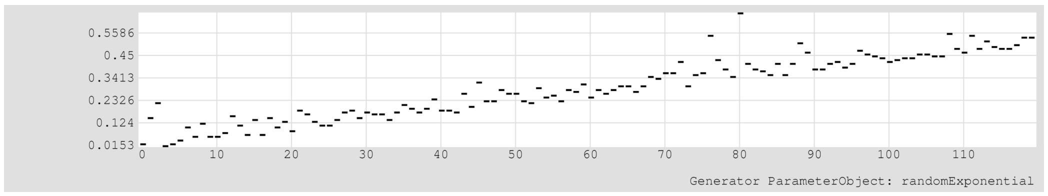
```
randomExponential, 0.5, (constant, 0), (constant, 1)
```

Example C-107. randomExponential Demonstration 2



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

Example C-108. randomExponential Demonstration 3



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