athenaCL Tutorial Manual

Third Edition, Version 2.0.0a15

Christopher Ariza

athenaCL Tutorial Manual: Third Edition, Version 2.0.0a15

by Christopher Ariza

athenaCL 2.0.0a15 Edition

Published 7 July 2010

Table of Contents

List of Examples

Preface

1. Overview of the athenaCL System

The athenaCL system is a software tool for creating musical structures. Music is rendered as a polyphonic event list, or an EventSequence object. This EventSequence can be converted into diverse forms, or OutputFormats, including scores for the Csound synthesis language, Musical Instrument Digital Interface (MIDI) files, and other specialized formats. Within athenaCL, Orchestra and Instrument models provide control of and integration with diverse OutputFormats. Orchestra models may include complete specification, at the code level, of external sound sources that are created in the process of OutputFormat generation.

The athenaCL system features specialized objects for creating and manipulating pitch structures, including the Pitch, the Multiset (a collection of Pitches), and the Path (a collection of Multisets). Paths define reusable pitch groups. When used as a compositional resource, a Path is interpreted by a Texture object (described below).

The athenaCL system features three levels of algorithmic design. The first two levels are provided by the ParameterObject and the Texture. The ParameterObject is a model of a low-level one-dimensional parameter generator and transformer. The Texture is a model of a multi-dimensional generative musical part. A Texture is controlled and configured by numerous embedded ParameterObjects. Each ParameterObject is assigned to either event parameters, such as amplitude and rhythm, or Texture configuration parameters. The Texture interprets ParameterObject values to create EventSequences. The number of ParameterObjects in a Texture, as well as their function and interaction, is determined by the Texture's parent type (TextureModule) and Instrument model. Each Texture is an instance of a TextureModule. TextureModules encode diverse approaches to multi-dimensional algorithmic generation. The TextureModule manages the deployment and interaction of lower level ParameterObjects, as well as linear or non-linear event generation. Specialized TextureModules may be designed to create a wide variety of musical structures.

The third layer of algorithmic design is provided by the Clone, a model of the multi-dimensional transformative part. The Clone transforms EventSequences generated by a Texture. Similar to Textures, Clones are controlled and configured by numerous embedded ParameterObjects.

Each Texture and Clone creates a collection of Events. Each Event is a rich data representation that includes detailed timing, pitch, rhythm, and parameter data. Events are stored in EventSequence objects. The collection all Texture and Clone EventSequences is the complete output of athenaCL. These EventSequences are transformed into various OutputFormats for compositional deployment.

For general information on computer aided algorithmic composition and generative music systems, see the resources listed here and in *References* (Ariza 2005b, 2009a).

The athenaCL system has been under development since June 2000. The software is cross platform, developed under an open-source license, and programmed in the Python language. An interactive command-line interface provides an easy-to-use environment for beginners and a quick reference for advanced users. The complete functionality of the system is alternatively available as a scriptable batch processor or as a programmable Python extension library.

2. Getting Started and Advanced Work

To learn the athenaCL system, many basic concepts of the system design and command interface must be examined in depth. The tutorials included in this document provide an overview to all essential concepts. Following the tutorials are appendices, providing documentation useful for reference. Much of this reference documentation is also available from within athenaCL.

All users should read Chapter 1 and Chapter 2 to gain familiarity with the interface and basic athenaCL concepts. Basic composition tools are covered in Chapter 4, Chapter 5, and Chapter 6. For more detailed information on organizing pitch structures, see Chapter 3.

Users with experience with Python and/or other generative music systems, or users who have mastered the athenaCL interactive command-line interface, will likely want to begin storing athenaCL command scripts in Python code files. This approach provides a wide range of opportunities for programmatically extending the power of athenaCL. A common approach for advanced usage of athenaCL is to use the interactive command-line interface for reference and sketching, and then store series of commands or command-generating procedures in Python files. The section Chapter 7 provides basic examples for this approach. Additional, over 30 demonstration Python scripts are distributed with athenaCL and included in Appendix F.

3. More Information

This document does not offer a complete description of the history, context, and internal structure of the athenaCL system; such a description, including comparative analysis to related historical and contemporary systems and detailed explanation of object models and interactions, is provided in the text *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL* (Ariza 2005a). Numerous additional articles are available that explore aspects of the athenaCL system in detail (Ariza 2002, 2003, 2004, 2005c, 2006, 2007a, 2007b, 2008, 2009b).

4. Conventions Used in This Manual

The following typographical conventions are used throughout this book:

`Constant width`

Used for athenaCL text output as transcribed in examples. This is what the program displays to the user.

`Constant width bold`

Used for user text input as transcribed in examples. This is what the user enters into the program.

5. Production of This Manual

The first edition of the *athenaCL Tutorial Manual* was released in August of 2001 and covered athenaCL versions 1.0 to 1.3. The second edition was released in June 2005 and covers athenaCL versions 1.4 and beyond. The third edition was released in July 2010 and convers athenaCL versions 2.0.

This manual is constructed and maintained with the help of various open-source tools: DocBook (http://www.docbook.org), the Modular DocBook Stylesheet distribution (http://docbook.sourceforge.net/projects/dsssl/), OpenJade (http://openjade.sourceforge.net/), Python (http://www.python.org/), and ImageMagick (http://www.imagemagick.org/).

Chapter 1. Tutorial 1: The Interactive Command Line Interface

This tutorial provides essential information and examples for using athenaCL's interactive command-line Interpreter. This material is essential for understanding basic athenaCL operation and how to obtain help within the program.

1.1. Starting the athenaCL Interpreter

Depending on your platform, there are a number of different ways to launch the athenaCL program and start the athenaCL Interpreter. For all platforms, using athenaCL requires installing (or finding) Python 2.6 (or better) on your system. Many advanced operating systems (UNIX-based operating systems including GNU/Linux and MacOS X) ship with Python installed.

For complete instructions on installing and launching athenaCL in each platform, please see the file "README.txt" included in the athenaCL distribution and in Appendix A.

After launching athenaCL, the user is presented with a text-based display in a terminal or Python-interactive window. The user is presented with the following initialization information:

Example 1-1. Initialization information

```
athenaCL 2.0.0 (on darwin via terminal)
Enter "cmd" to see all commands. For help enter "?".
Enter "c" for copyright, "w" for warranty, "r" for credits.

pi{}ti{} ::
```

When starting up the Interpreter, athenaCL looks in the athenaCL directory for the "libATH" folder, and then various directories within the "libATH" folder. These directories contain essential files and must be present for the program to run. The athenaCL prompt "::" is preceded by information concerning the AthenaObject. This will be explained in greater detail below.

1.2. Introduction to Commands

When using athenaCL, the user enters commands to get things done. athenaCL commands are organized by prefixes, two-letter codes that designate what the command operates upon. Prefixes are always displayed as capitalized letters, though the user, when entering commands, may use lower-case letters. Some common prefixes are "PI", for PathInstance, or "TI", for TextureInstance. What follows the prefix usually resembles UNIX shell commands: "ls" for listing objects, "rm" for removing objects. For example, the command to list all the available TextureModules is TMls: "TM" for TextureModule, "ls" for list. When no common UNIX command-abbreviation is available, intuitive short abbreviations are used. For example, the command to create the retrograde of a PathInstance is PIret: "PI" for PathInstance, "ret" for retrograde.

The division of commands into prefixes demonstrates, in part, the large-scale design of the AthenaObject. The AthenaObject consists of PathInstances and TextureInstances. PathInstances

are objects that define pitch materials. TextureInstances define algorithmic music layers. Users can create, copy, edit and store collections of Paths and Textures within the AthenaObject. All Texture related commands, for example, start with a "T", like TextureTemperament ("TT"), TextureClone("TC") and TextureModule ("TM").

In addition to the commands available for working with Paths and Textures, there are commands for creating various event list formats (such as Csound scores and MIDI files) with the EventList commands (prefix "EL"). The complete AthenaObject, with all its Paths and Textures, is handled with AthenaObject commands (prefix "AO"). These commands are used to save and control the complete collection of Paths and Textures.

1.3. Viewing Command Names

When starting athenaCL, the user is presented with a prompt (::). To display a listing of all commands enter "cmd", for command:

Example 1-2. Listing all commands

```
pi{y0}ti{a2} :: cmd
athenaCL Commands:
.....................................................................
PathInstance        PIn(new)            PIcp(copy)          PIrm(remove)
                    PIo(select)         PIv(view)           PIe(edit)
                    PIdf(duration)      PIls(list)          PIh(hear)
                    PIret(retro)        PIrot(rot)          PIslc(slice)
.....................................................................
TextureModule       TMo(select)         TMv(view)           TMls(list)
TextureParameter    TPls(list)          TPv(select)         TPmap(map)
                    TPe(export)
TextureInstance     TIn(new)            TIcp(copy)          TIrm(remove)
                    TIo(select)         TIv(view)           TIe(edit)
                    TIls(list)          TImode(mode)        TImute(mute)
                    TIdoc(doc)          TImap(map)          TImidi(midi)
TextureClone        TCn(new)            TCcp(copy)          TCrm(remove)
                    TCo(select)         TCv(view)           TCe(edit)
                    TCls(list)          TCmute(mute)        TCmap(map)
TextureTemperament  TTls(list)          TTo(select)
TextureEnsemble     TEv(view)           TEe(edit)           TEmap(map)
                    TEmidi(midi)
.....................................................................
EventOutput         EOls(list)          EOo(select)         EOrm(remove)
EventMode           EMls(list)          EMo(select)         EMv(view)
                    EMi(inst)
EventList           ELn(new)            ELw(save)           ELv(view)
                    ELh(hear)           ELr(render)         ELauto(auto)
.....................................................................
AthenaPreferences   APdir(directory)    APea(external)      APa(audio)
                    APgfx(graphics)     APcurs(cursor)      APdlg(dialogs)
                    APr(refresh)        APwid(width)
AthenaHistory       AHls(list)          AHexe(execute)
AthenaUtility       AUsys(system)       AUdoc(docs)         AUup(update)
                    AUbeat(beat)        AUpc(pitch)         AUmg(markov)
                    AUma(markov)        AUca(automata)
AthenaObject        AOw(save)           AOl(load)           AOmg(merge)
                    AOrm(remove)
```

This display, organized by prefix heading, shows each command followed by a longer description of the commands name.

1.4. Executing Commands

To use a command, simply enter its name. The user will be prompted for all additional information. For example, type "PIn" (or "pin") at the athenaCL prompt:

Example 1-3. Entering a command

```
pi{}ti{} :: pin
name this PathInstance: a
enter a pitch set, sieve, spectrum, or set-class: b,c#,g
      SC 3-8B as (B4,C#4,G4)? (y, n, or cancel): y
      add another set? (y, n, or cancel): n
PI a added to PathInstances.
```

This command prompts the user for a "pitch set, sieve, spectrum, or set-class" and then creates a multiset component of a Path. A Xenakis sieve (Xenakis 1990, 1992; Ariza 2004, 2005c, 2009b) can be entered using a logical string and a pitch range. Set class labels are given using Forte names. The user may enter the chord itself as pitch-names (with sharps as "#" and flats as "$") or pitch-classes (integers that represent the notes of the chromatic scale) (Straus 1990). For instance, the chord D-major can be represented with the following pitch-name string: (D, F#, A). Or, the same chord can be represented as a pitch class set: (2,6,9), where 0 is always C, 1=C#, 2=D, …, 10=A#, and 11=B. Calling the PIn command to create a new path named "b" with this pitch class set gives us the following results:

Example 1-4. Entering a command with arguments

```
pi{a}ti{} :: pin b d,f#,a
PI b added to PathInstances.
```

Notice that in the above example the Path name and pitch collection arguments are entered at the same time as the command: "pin b d,f#,a". As an interactive command-line program, athenaCL can obtain arguments from the user, and can, alternatively, accept space-separated arguments following a command. Command-line arguments allow advanced users ease and speed and, when called from an external environment (such as a UNIX shell or Python script), permit advanced scripting automation. All athenaCL commands can function both with arguments and with interactive prompts. Command-line arguments, however, are never required: if arguments are absent, the user is prompted for the necessary details.

1.5. Getting Help for Commands

athenaCL provides two ways of helping the user access and learn commands. If the user only remembers the prefix of a command, this prefix can be entered at the prompt to produce a list of all commands associated with that prefix:

Example 1-5. Displaying a command listing

```
pi{b}ti{} :: pi
PI (PathInstance) commands:
   PIn              new
   PIcp             copy
   PIrm             remove
   PIo              select
   PIv              view
   PIe              edit
   PIdf             duration
   PIls             list
   PIh              hear
   PIret            retro
   PIrot            rot
   PIslc            slice
```

Help information is available for each command and can be accessed from the athenaCL prompt by typing either "?" or "help" followed by the name of the command. The following example provides the documentation for the PIn command. Notice that the main documentation is followed by "usage" documentation, or the format required for providing command-line arguments:

Example 1-6. Using the help command

```
pi{b}ti{} :: help pin
{topic,documentation}
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.
usage:              pin name set1 ... setN
```

The same help command can be used to access information concerning additional topics, notations, and representations used within athenaCL. For example, information about Markov transition strings can be accessed with the same help command:

Example 1-7. Accessing additional help topics

```
pi{b}ti{} :: ? markov
{topic,documentation}
Markov Notation     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".
```

Throughout this document additional information for the reader may be recommended by suggesting the use of the help command. For example: (enter "help markov" for more information).

1.6. Configuring the User Environment

athenaCL has many configurable settings that are saved in a preference file and loaded for each athenaCL session. Some of these settings have default values; others will need to be configured the first time a command is used.

For example, following the athenaCL prompt ("::") is the the athenaCL "cursor tool." This tool, providing information on the active Texture and Path, can be turned on or off with the command APcurs, for AthenaPreferences cursor:

Example 1-8. Toggling the athenaCL cursor tool with APcurs

```
pi{b}ti{} :: apcurs
cursor tool set to off.

:: apcurs
cursor tool set to on.

pi{b}ti{} ::
```

athenaCL writes files. Some of these files are audio file formats, some are event list formats (scores, MIDI files), and some are image files. In most cases, athenaCL will write a file in a user specified "scratch" directory with an automatically-generated file name. This is convenient and fast. To set the scratch directory, enter the APdir command, for AthenaPreferences directory. (Replace "/Volumes/xdisc/_scratch" with a complete file path to a suitable directory.)

Example 1-9. Setting the scratch directory with APdir

```
pi{b}ti{} :: apdir
select directory to set: scratch or audio. (x or a): x
/Users/ariza/_x/src/athenaCL
.....................................................................
.cvsignore      .DS_Store       __init__.py     __init__.pyc    __init__.pyo
athenacl.py     athenacl.pyc    athenaObj.py    athenaObj.pyc   athenaObj.pyo
CVS             demo            docs            libATH          setup.py
tools
select a scratch directory:
to change directory enter name, path, or ".."
cancel or select? (c or s): /Volumes/xdisc/_scratch
/Volumes/xdisc/_scratch
.....................................................................
.DS_Store       a.mid
select a scratch directory:
to change directory enter name, path, or ".."
cancel or select? (c or s): s
user scratch directory set to /Volumes/xdisc/_scratch.
```

The command PIh, for PathInstance hear, allows the creation of a MIDI file from a single Path specification. In this case, athenaCL writes the MIDI file in the user-specified scratch directory. After the file is written, athenaCL opens the file with the operating system. Depending on how the operating system is configured, the MIDI file should open in an appropriate player. The athenaCL system frequently works in this manner with the operating system and external programs and resources.

Example 1-10. Creating a MIDI file with PIh

```
pi{b}ti{} :: pih
command.py: temporary file: /Volumes/xdisc/_scratch/ath2010.07.01.16.12.52.xml
PI b hear with TM LineGroove complete.
(/Volumes/xdisc/_scratch/ath2010.07.01.16.12.52.mid)
```

Numerous types of graphical aids are provided by athenaCL to assist in the representation of musical materials. Depending on the user's Python installation, numerous formats of graphic files are available. Formats include text (within the Interpreter display), Encapsulated PostScript (convertible to PDF), Tk GUI Windows, JPEG, and PNG. Tk requires the Python TkInter GUI installation; JPEG and PNG require the Python Imaging Library (PIL) installation.

The user can set an active graphic format with the APgfx command. For example:

Example 1-11. Setting the active graphics format with APgfx

```
pi{b}ti{} :: apgfx
active graphics format: png.
select text, eps, tk, jpg, png. (t, e, k, j, or p): p
graphics format changed to png.
```

To test the production of graphic output, the TPmap command, for TextureParameter map, can be used:

Example 1-12. Producing a graphical diagram with TPmap

```
pi{b}ti{} :: tpmap 100 ru
randomUniform, (constant, 0), (constant, 1)
TPmap display complete.
```

Chapter 2. Tutorial 2: AthenaObjects and EventModes

This tutorial provides essential information concerning saving and opening an athenaCL session, as well as basic information for creating and configuring EventLists and EventModes.

2.1. Introduction to AthenaObjects

The AthenaObject stores the collection of user-created PathInstances and TextureInstances, as well as the names of the active objects and other settings relevant to the active athenaCL session (and not stored in the user preference file). The AthenaObject, when saved, is stored as an XML file. When athenaCL creates an XML AthenaObject file, the resulting file contains the complete state of the active AthenaObject.

2.2. File Dialogs in athenaCL

The athenaCL system supports a variety of styles of file dialogs, or the interface used to obtain and write files or directories. The default style of file dialog uses a custom text interface that lets the user browse their file system. Alternatively, all commands that require file or directory paths may be executed by supplying the complete file path as a command-line argument.

Use of text-base file dialogs, however, may not be convenient for some users. For this reason athenaCL offers GUI-based graphical file dialogs on platforms and environments that support such features. On Python installations that have the Tk GUI library TkInter installed, Tk-based file dialogs are available. On the Macintosh platform (OS9 and OSX) native MacOS file-dialogs are available. To change they athenaCL dialog style, enter the command APdlg:

Example 2-1. Changing the file dialog style with APdlg

```
pi{}ti{} :: apdlg
active dialog visual method: text.
select text, tk, or mac. (t, k, or m): t
dialog visual method changed to text.
```

Note: on some platforms use of GUI windows from inside a text-environment may cause unexpected results. In some cases, the GUI window may appear behind all other windows, in the background.

2.3. Loading and Removing an AthenaObject

The command AOl, for AthenaObject load, permits the user to load an AthenaObject XML file. Numerous small demonstration files are included within athenaCL. In the following example, the user loads the file "demo01.xml".

The following display demonstrates use of the text-based file-dialogs. When using the text-based interface, the user must select a directory before selecting a file. In the example below, the user

enters "demo" to enter the "demo" directory in the athenaCL directory. The user then enter "s" to select this directory. Next, the user has the option the select a file from this directory, change the directory, or cancel. The user chooses to select a file with "f". After entering the name of the file ("demo01.xml") and confirming, the AthenaObject is loaded:

Example 2-2. Loading an AthenaObject with text-based file selection

```
pi{}ti{} :: aol
select an AthenaObject file:
name file, change directory, or cancel? (f, cd, c): cd
/Volumes/xdisc/_sync/_x/src/athenacl/athenaCL/demo/legacy
.........................................................................
.svn            __init__.py     demo01.xml      demo03.xml      demo05.xml
spectrum01.txt  tutorial02.xml  tutorial03.xml  tutorial04.xml  tutorial05.xml
tutorial06.xml  tutorial07.xml  tutorial09.xml
to change directory enter name, path, or ".."
cancel or select? (c or s): s
select an AthenaObject file:
name file, change directory, or cancel? (f, cd, c): f
name file? demo01.xml
      1.3.1 xml AthenaObject loaded (00:01):
/Volumes/xdisc/_sync/_x/src/athenacl/athenaCL/demo/legacy/demo01.xml
```

To confirm that the AthenaObject has been loaded, the user may enter TIls to display a list of all TextureInstances. (For more information concerning Textures, see Chapter 4).

Example 2-3. Listing TextureInstances with Tils

```
pi{y0}ti{a2} :: tils
TextureInstances available:
{name,status,TM,PI,instrument,time,TC}
   _space           + MonophonicOrnament x0          62  39.0--40.0   0
   a0               + MonophonicOrnament y0          50  01.0--41.0   0
   a1               + MonophonicOrnament y0          50  01.0--41.0   0
 + a2               + MonophonicOrnament y0          50  01.0--41.0   0
```

The entire AthenaObject can be erased and set to its initial state without restarting the athenaCL program. The following example uses AOrm, for AthenaObject remove, to re-initialize the AthenaObject. Note: the AOrm will permanently remove all objects within athenaCL and cannot be un-done.

Example 2-4. Reinitializing the AthenaObject with AOrm

```
pi{y0}ti{a2} :: aorm
destroy the current AthenaObject? (y or n): y
reinitializing AthenaObject.

pi{}ti{} ::
```

If the AthenaObject file is located in the athenaCL "demo" directory, or a directory from which a file was opened or saved-to by the user within the current session, athenaCL can find the file by giving the AOl command with the file's name as a command-line argument. To reload "demo01.xml", the user may enter the following arguments:

Example 2-5. Loading an AthenaObject from the command-line

```
pi{}ti{} :: aol demo01.xml
      1.3.1 xml AthenaObject loaded (00:01):
/Volumes/xdisc/_sync/_x/src/athenacl/athenaCL/demo/legacy/demo01.xml
```

2.4. EventModes and EventOutputs

After loading a demonstration file containing TextureInstances, athenaCL can be used to create an EventList. As a poly-paradigm system with integrated instrument models, athenaCL supports numerous formats of EventLists and can work with a wide variety of sound sources, including Csound and MIDI. What types of EventLists are created depends on two settings within athenaCL: the EventMode and the EventOutput.

The EventModes configure athenaCL for working with a particular sound source and Orchestra model, such as the internal Csound orchestra (csoundNative), external Csound orchestras (csoundExternal), various types of MIDI files (generalMidi an generalMidiPercussion), and others. The EventMode determines what instruments are available for Texture creation (see Chapter 4, as well as the operation of some EventList commands. In some cases, the EventMode forces certain EventOutput formats to be written as well.

The EventOutputs select what file formats will be created when a new EventList is generated. athenaCL permits the user to create an EventList in numerous formats simultaneously. For example, a Csound score and orchestra, a MIDI file, and tab-delimited table can all be produced from one call to the EventList new command. Some EventOutput formats are created only if the AthenaObject contains Textures created in the appropriate EventMode. Other EventOutput formats can be created with any Texture in any EventMode. Such conflicts, however, are never a problem: athenaCL simply creates whatever EventOutput formats are appropriate based on the user-specified request.

To view the current EventMode, enter EMls. To view the current list of selected EventOutputs, enter EOls. The following example demonstrates these commands:

Example 2-6. Viewing EventMode and EventOutputs

```
pi{y0}ti{a2} :: emls
EventMode modes available:
{name}
   csoundExternal
 + csoundNative
   csoundSilence
   midi
   midiPercussion
   superColliderNative
```

```
pi{y0}ti{a2} :: eols
EventOutput active:
{name}
   acToolbox
   audioFile
   csoundBatch
 + csoundData
   csoundOrchestra
   csoundScore
 + midiFile
   pureDataArray
   superColliderTask
   textSpace
   textTab
 + xmlAthenaObject
```

To select an additional EventOutput to be requested when a new EventList is created, enter the command EOo, for EventOutput select. To remove an EventOutput, enter the command EOrm, for EventOutput remove. In the following example, the user adds a tab-delimited table output ("textTab") and a specialized output file for the AC Toolbox ("acToolbox"). After viewing the EventOutput list, these EventOutputs are removed. Note: EventOutputs, like many selection in athenaCL, can be designated using automatic acronym expansion (AAE), the user providing only the leading character and capitals.

Example 2-7. Adding and Removing EventOutputs

```
pi{y0}ti{a2} :: eoo tt at
EventOutput formats: midiFile, xmlAthenaObject, csoundData, textTab, acToolbox.

pi{y0}ti{a2} :: eols
EventOutput active:
{name}
 + acToolbox
   audioFile
   csoundBatch
 + csoundData
   csoundOrchestra
   csoundScore
 + midiFile
   pureDataArray
   superColliderTask
   textSpace
 + textTab
 + xmlAthenaObject

pi{y0}ti{a2} :: eorm tt at
EventOutput formats: midiFile, xmlAthenaObject, csoundData.
```

2.5. Creating an EventList

To create an EventList, the command ELn, for EventList new, must be used. This command generates a new EventList for each Texture and Clone, and writes necessary EventOutput formats. Each time the ELn command is called, a new musical variant (depending on Texture, Clone, and ParameterObject specification) is produced. It is possible, even likely, that two EventLists, generated

from the same AthenaObject file, will not be identical. EventLists, further, are never stored within an AthenaObject. For this reason, users should be careful to save and preserve produced EventList files.

When using the ELn command alone, temporary files are created, either in a system-location, or in the scratch directory selected by the user. The user may also, optionally, name the EventList. The EventList name is given as a file name (or a complete file path) ending with an ".xml" extension. Although the ELn command may produce many files, only one file path needs to be provided: all other EventOutput format file names are derived from this source .xml file path. If EventOutput xmlAthenaObject is active, an XML AthenaObject file will be written along with whatever user-specified or EventMode-mandated EventOutput formats are created.

In the example above, the user's EventOutput format specification indicates that midiFile and xmlAthenaObject are active outputs. The current EventMode, however, is set to csoundNative, and the Textures of "demo01.xml", upon examination, were created with csoundNative instruments. For these reasons, the ELn command, in this case, will produce an .xml AthenaObject file, a Csound .csd file, a MIDI file (.mid), and a script file for processing the Csound orchestra and score (.bat). For example:

Example 2-8. Creating a new EventList with Eln

```
pi{y0}ti{a2} :: eln
      EventList ath2010.07.02.13.22.35 complete:
/Volumes/xdisc/_scratch/ath2010.07.02.13.22.35.bat
/Volumes/xdisc/_scratch/ath2010.07.02.13.22.35.csd
/Volumes/xdisc/_scratch/ath2010.07.02.13.22.35.mid
/Volumes/xdisc/_scratch/ath2010.07.02.13.22.35.xml
```

Csound files require additional processing to hear audio from the results: this will be demonstrated below. The MIDI file, however, can be listened to immediately with any MIDI file player, such as QuickTime. To hear the file produced by ELn, enter the command ELh, for EventList hear:

Example 2-9. Opening an EventList with Elh

```
pi{y0}ti{a2} :: elh
EventList hear initiated: /Volumes/xdisc/_scratch/ath2010.07.02.13.22.35.mid
```

Depending on operating system configuration, the ELh command should open the newly-created MIDI file in a MIDI-file player. Alternatively, the MIDI file can be opened in an application that supports MIDI files, such as a notation program or sequencer.

The ELn command, as all athenaCL commands, can be used with command-line arguments. To create an EventList in a specific directory, simply provide a complete file path following the the ELn command. (Replace "/Volumes/xdisc/_scratch/" with a complete file path to a suitable directory.)

Example 2-10. Creating a new EventList with Eln and command-line arguments

```
pi{y0}ti{a2} :: eln /Volumes/xdisc/_scratch/test02.xml
      EventList test02 complete:
/Volumes/xdisc/_scratch/test02.bat
/Volumes/xdisc/_scratch/test02.csd
/Volumes/xdisc/_scratch/test02.mid
/Volumes/xdisc/_scratch/test02.xml
```

Using the ELh command to listen to this EventList, the user should identify that although "test01" and "test02" are closely related, each musical fragment, due to algorithmic variation, has differences.

2.6. Configuring and Using Csound

Although Csound files were created in the above examples, only the resulting MIDI files were auditioned. To produce audio files with Csound, some additional configuration may be necessary.

To create an audio file with Csound, two files are required: a score (.sco) and an orchestra (.orc); alternatively, both files can be combined into a single XML file called (within athenaCL) a csoundData file (.csd). With the csoundNative instruments and EventMode, all necessary Csound files are created by athenaCL. To activate csoundData file production, the EventOutput csoundData must be selected. Alternatively, users can create only a Csound score (with EventModes csoundExternal or csoundSilence), and apply this score to any desired external Csound orchestra.

The Csound audio rendering software must be installed separately. Csound is an open source, free, cross platform program available for all major operating systems.

Once configured properly, athenaCL provides commands to control Csound rendering. The user may be required to provide the location of (file path to) the Csound program; the location of the Csound program is set with the APea command, or Athena Preferences external applications command. Each platform has a different default Csound application specified. Unix: default position is /usr/local/bin/csound; MacOS X: default Csound is the same as Unix; Windows: users must select the Csound executable, "winsound.exe," with the APea command. The user can select a different Csound with the APea command; this selection is stored in the user preferences and is maintained between athenaCL sessions.

Assuming that the necessary Csound files were created with ELn as demonstrated above, the user may view the Csound score file created with the command ELv, or EventList view. Depending on operating system configuration, this command will open the score file with a platform-specific text reader. Alternatively, the .sco file can be manually selected and opened by the user.

Whenever athenaCL creates Csound files under EventMode csoundNative, a script file (.bat) is created to automate rendering of the audio file from the Csound score and orchestra (or .csd file). The script instructs Csound to create an audio file with the same name as the score in the same directory as the score, orchestra, and batch file.

Prior to writing files with the ELn command, the desired audio file format can be specified from within athenaCL using the command APa. The user will be prompted to select a file format from

the options given. Note: the user must set Csound options before executing ELn; otherwise, they will have no effect until a new EventList is created.

Example 2-11. Changing the Csound audio file format with CPff

```
pi{y0}ti{a2} :: apa
select format, channels, or rate. (f,c,r): f
current audio file format: aif.
select aif, wav, or sd2. (a, w, or s): a
audio format set to 'aif'.
```

Assuming correct Csound installation and configuration within athenaCL, the user can enter ELr to automatically initiate Csound rendering of the last Csound score created with ELn. ELr, using the operating system, calls the athenaCL-created script. For ELr to function, and thus the ELn-created script to function, the Csound score and orchestra files (or .csd file) must remain in their original locations.

Example 2-12. Rendering a Csound score

```
pi{y0}ti{a2} :: elr
audio rendering initiated: /Volumes/xdisc/_scratch/test02.bat
```

Alternatively, users can render Csound files created in athenaCL within any Csound application, just as they would for any other Csound score and orchestra, manually setting file Paths, file formats, and Csound option flags. See Csound documentation for more information on using Csound.

As demonstrated above with MIDI files, the user can open the Csound-rendered audio file with the ELh command. This command opens the audio file with a platform-specific media player.

Example 2-13. Opening Csound-generated audio files with ELh

```
pi{y0}ti{a2} :: elh
EventList hear initiated: /Volumes/xdisc/_scratch/test02.aif
EventList hear initiated: /Volumes/xdisc/_scratch/test02.mid
```

To summarize, there are three athenaCL commands needed to create, render, and hear a Csound score, and they must be executed in order: ELn, ELr, ELh. To link these three commands, the user can set a automation preference with the ELauto command. When this option is toggled, the single command ELn will create an EventList, render it in Csound, and open the Csound-created audio file with a platform-specific media player.

2.7. Saving and Merging AthenaObjects

Loading a new AthenaObject will completely replace the current AthenaObject contents. For this reason, users should always save their work before loading a new AthenaObject. The user can,

alternatively, merge AthenaObjects. Merging is a powerful tool: the user can combine many AthenaObjects that have been saved separately, or combine an AthenaObject numerous times with itself. In the example below, the user merges "demo01.xml", loaded above, with another of the same AthenaObject "demo01.xml". The file paths for athenaCL demonstration files are known to athenaCL, and thus the user can simply provide the name of the demonstration file as a command-line argument.

Example 2-14. Merging AthenaObjects with AOmg

```
pi{y0}ti{a2} :: aomg demo01.xml
      1.3.1 xml AthenaObject merged (00:01):
/Volumes/xdisc/_sync/_x/src/athenacl/athenaCL/demo/legacy/demo01.xml

pi{y0}ti{a2} ::
```

The command TIls can be used to confirm that the AthenaObjects have been merged. The AOmg command, in the case that two Paths or Textures have the same name, automatically alters the name by appending an underscore ("_"). In the case where an AthenaObject is merged with itself as in this example, each Texture and Path is duplicated.

Example 2-15. Listing TextureInstances

```
pi{y0}ti{a2} :: tils
TextureInstances available:
{name,status,TM,PI,instrument,time,TC}
   _space           + MonophonicOrnament x0          62  39.0--40.0   0
   _space_          + MonophonicOrnament x0_         62  39.0--40.0   0
   a0               + MonophonicOrnament y0          50  01.0--41.0   0
   a0_              + MonophonicOrnament y0_         50  01.0--41.0   0
   a1               + MonophonicOrnament y0          50  01.0--41.0   0
   a1_              + MonophonicOrnament y0_         50  01.0--41.0   0
 + a2               + MonophonicOrnament y0          50  01.0--41.0   0
   a2_              + MonophonicOrnament y0_         50  01.0--41.0   0
```

As shown above, the user may create a new MIDI or Csound EventList of this new AthenaObject and audition the results. As should be clear, the resulting musical structure will sound more dense due to the additional Textures. Due to algorithmic variation, each Texture will remain relatively independent.

To save the current AthenaObject, the user may create an XML AthenaObject file. Although AthenaObject files may be created with the proper EventOutput selection and by use of the ELn command, in same cases the user my want to create the XML AthenaObject file alone. The command AOw, for AthenaObject Write, provides this functionality. The user must name the AthenaObject with a ".xml" extension. In the example below the user saves the merged files as a new AthenaObject named "merged.xml" using a command-line argument. If desired, the AOw command can be used without command-line arguments to select the location of the file with an interactive file dialog. (Replace "/Volumes/xdisc/_scratch/" with a complete file path to a suitable directory.)

Example 2-16. Creating a new AthenaObject with AOw

```
pi{y0}ti{a2} :: aow /Volumes/xdisc/_scratch/merged.xml
      AthenaObject saved:
/Volumes/xdisc/_scratch/merged.xml
```

Saving your work in athenaCL is very important, and should be done often. The athenaCL system can not reconstruct an AthenaObject from an EventList or an audio file; an athenaCL session can only be reconstructed by loading an AthenaObject XML file.

Chapter 3. Tutorial 3: Creating and Editing Paths

This tutorial demonstrates the basic features of the Path, including creating, storing, examining, and editing Paths.

3.1. Introduction to Paths

A PathInstance (or a Path or PI) is an ordered collection of pitch groups. A pitch group, or a Multiset, is the simultaneous representation of pitch-space, pitch-class space, and set-class information for a collection of microtonally-specified pitches. This collection can be treated as an ordered or unordered collection, can be edited by transposition, replacement, or serial re-ordering, and can be used by one or more Textures to provide pitch materials that are then independently transposed and interpreted by the Texture and its ParameterObjects.

A PathInstance allows the representation of ordered content groups, and presents this representation as a multifaceted object. Paths can be of any length, from one to many Multisets long. A Multiset can be specified in terms of pitch class (excluding octave information with integers from 0 to 11), or in terms of pitch-space (including octave information with integers below 0 or above 11, or with register-specific note names such as C3 and G#12). A Multiset can also be specified as a group, set, or scale sequence such as a Forte set-class (Forte 1973) or a Xenakis sieve (Ariza 2005c). Finally, Multisets can be derived from spectrums and frequency analysis information provided from the cross-platform audio editor Audacity (enter "help audacity" for more information).

A Path can be developed as a network of intervallic and motivic associations. The interpretation of a Path by a Texture provides access to diverse pitch representations for a variety of musical contexts, and permits numerous Textures to share identical or related pitch information. The use of a Path in a Texture, however, is optional: a Path can function, at a minimum, simply as a referential point in Pitch space from which subsequent Texture transpositions are referenced.

3.2. Creating, Selecting, and Viewing PathInstances

To create a PathInstance, enter PIn (for PathInstance new) at the athenaCL prompt. You must name the new Path, and then supply a pitch group, Forte-number, Xenakis sieve, or alternative pitch representation (enter "help pitch" for more information on pitch representations).

Example 3-1. Creating a new PathInstance with PIn

```
pi{}ti{} :: pin
name this PathInstance: pathA
enter a pitch set, sieve, spectrum, or set-class: e$, e, c#
      SC 3-2B as (D#4,E4,C#4)? (y, n, or cancel): y
      add another set? (y, n, or cancel): y
enter a pitch set, sieve, spectrum, or set-class: 0,1,6,7
      SC 4-9 as (C4,C#4,F#4,G4)? (y, n, or cancel): y
      add another set? (y, n, or cancel): y
enter a pitch set, sieve, spectrum, or set-class: 3-11
      SC 3-11A as (C4,D#4,G4)? (y, n, or cancel): y
      add another set? (y, n, or cancel): y
```

```
enter a pitch set, sieve, spectrum, or set-class: 7@3|6@4, g2, c4
      SC 4-6 as (A#2,B2,F3,B3,C4)? (y, n, or cancel): y
      add another set? (y, n, or cancel): n
PI pathA added to PathInstances.

pi{pathA}ti{} ::
```

Note that after successfully creating a Path, the athenaCL cursor tool changes to reflect the active Path: the name in parenthesis following "pi" designates the active Path ("pathA"). The same information is provided for a TextureInstance following the "ti" prefix. To view the active PI, enter PIv at the athenaCL prompt:

Example 3-2. Viewing a Path with PIv

```
pi{pathA}ti{} :: piv
PI: pathA
psPath              3,4,1       0,1,6,7         0,3,7       -14,-13,-7,-1,0
                    D#4,E4,C#4  C4,C#4,F#4,G4   C4,D#4,G4   A#2,B2,F3,B3,C4
pcsPath             3,4,1       0,1,6,7         0,3,7       10,11,5,11,0
scPath              3-2B        4-9             3-11A       4-6
durFraction         1(25%)      1(25%)          1(25%)      1(25%)
TI References: none.
```

This display provides all essential information about a Path. The header contains the name of the Path ("pathA"). The parallel presentation of psPath, pcsPath, and scPath illustrates the simultaneous availability of pitch space, pitch class space, and set class representations. The label "TI references", when needed, provides information on which TextureInstances link to this PathInstance.

In order to hear a possible interpretation of this Path, the command PIh generates a MIDI file based on a simple interpretation of the Path with the active TextureModule. The resulting musical structure is only provided to audition the Path, and uses default values for all musical parameters. The MIDI file is written in the user-specified scratch directory (see Example 1-9) and is opened via the operating system.

Example 3-3. Creating a MIDI file with PIh

```
pi{pathA}ti{} :: pih
PI pathA hear with TM LineGroove complete.
(/Volumes/xdisc/_scratch/ath2010.07.02.16.29.39.mid)
```

A second Path can be created exclusively with Forte set class numbers. In this example, all arguments are provided via the command line:

Example 3-4. Creating a Path with Forte numbers

```
pi{pathA}ti{} :: pin pathB 5-3 6-4 7-34 4-14
PI pathB added to PathInstances.
```

A newly-created Path always becomes the active Path. Entering PIv will display the details of the newly created Path:

Example 3-5. Displaying a Path

```
pi{pathB}ti{} :: piv
PI: pathB
psPath              0,1,2,4,5         0,1,2,4,5,6           0,1,3,4,6,8,10
                    C4,C#4,D4,E4,F4   C4,C#4,D4,E4,F4,F#4   C4,C#4,D#4,E4,F#4,G#4,
pcsPath             0,1,2,4,5         0,1,2,4,5,6           0,1,3,4,6,8,10
scPath              5-3A              6-4                   7-34
durFraction         1(25%)            1(25%)                1(25%)
                    ..........................................................
                         0,2,3,7
                    A#4  C4,D4,D#4,G4
                         0,2,3,7
                         4-14A
                         1(25%)
TI References: none.
```

As is clear from the PIv display above, when a Multiset in a Path is entered as a Set class, a pitch space and a pitch class space representation (psPath, pcsPath) are created from the normal-form of the desired SetClass.

In order to display the complete collection of Paths available in the AthenaObject, the user enters PIls, for PathInstance list:

Example 3-6. Listing Paths

```
pi{pathB}ti{} :: pils
PathInstances available:
{name,TIrefs,scPath}
   pathA            0  3-2B,4-9,3-11A,4-6
 + pathB            0  5-3A,6-4,7-34,4-14A
```

Many displays provided by athenaCL are given in columns of data. After whatever header information is give, a key, in braces ("{}"), is provided to define the data provided in each column. In the example above, the key shows that each row contains the name of the PI, the number of TI references, the number of PathVoices, and an scPath representation of the Path. The "+" next to "pathB" illustrates that this PI is currently active. All "ls" commands use a similar designation.

Many commands in athenaCL function by using an "active" object. The active PI defines which Path is used in many different commands. For example, the PIv command, when used without an argument for which Path to display, displays the active Path.

To select a different PI as the active PI, simply enter PIo. The user is prompted to either enter the name of the Path to select, or its order number from the "ls" view (where 1 is pathA, 2 is pathB). Displaying the list of all PathInstances will confirm that pathA is now the selected PI.

Example 3-7. Selecting Paths

```
pi{pathB}ti{} :: pio
select a path to activate: (name or number 1-2): pathA
PI pathA now active.

pi{pathA}ti{} :: pils
PathInstances available:
{name,TIrefs,scPath}
 + pathA            0  3-2B,4-9,3-11A,4-6
   pathB            0  5-3A,6-4,7-34,4-14A
```

Alternatively the user can enter the name of the Path to be selected as a command-line argument with the PIo command. After making pathA active, the user can make pathB active again by entering the following:

Example 3-8. Selecting a Path with an argument

```
pi{pathA}ti{} :: pio pathB
PI pathB now active.
```

3.3. Copying and Removing PathInstances

In order to manage the collection of Paths in the AthenaObject, the user can copy and remove Paths. In all cases of copying and removing user-defined objects in athenaCL, the active object is never assumed to be the object that the command should be performed upon. Said another way, the user must always specify which object(s) to copy or remove.

To copy a Path instance, enter PIcp and select a Path to copy:

Example 3-9. Copying a Path with PIcp

```
pi{pathB}ti{} :: picp
select a path to copy: (name or number 1-2): pathB
name the copy of path pathB: pathC
PI pathC added to PathInstances.

pi{pathC}ti{} :: pils
PathInstances available:
{name,TIrefs,scPath}
   pathA            0  3-2B,4-9,3-11A,4-6
   pathB            0  5-3A,6-4,7-34,4-14A
 + pathC            0  5-3A,6-4,7-34,4-14A
```

To delete a Path, enter PIrm and select a Path to delete as above. In the example below, the Path to delete is given with a command line argument:

Example 3-10. Removing a Path with PIrm

```
pi{pathC}ti{} :: pirm pathB
```

```
PI pathB destroyed.
```

3.4. Editing PathInstances

A Path can be edited as a serial succession of Multisets with the standard assortment of serial operations: retrograde, rotation, and slice. Additionally, each Multiset in a Path can be changed, either by transposition or replacement.

Whenever a serial edit is performed on a Path, the edited Path becomes a new, distinct Path and the original Path is left unchanged. For example, to create the retrograde of the active Path, enter PIret. The user must provide the name of the new Path:

Example 3-11. Creating a retrograde of a Path with PIret

```
pi{pathC}ti{} :: piret
name this PathInstance: pathCret
retrograde PI pathCret added to PathInstances.

pi{pathCret}ti{} :: pils
PathInstances available:
{name,TIrefs,scPath}
   pathA            0  3-2B,4-9,3-11A,4-6
   pathC            0  5-3A,6-4,7-34,4-14A
 + pathCret         0  4-14A,7-34,6-4,5-3A
```

To create a rotation, the user, after entering PIrot, must enter the number of the Multiset to occupy the new first position. If the new first position is to be the second Multiset, the user would enter 2:

Example 3-12. Creating a rotation of a Path with PIrot

```
pi{pathCret}ti{} :: pirot
name this PathInstance: pathCretRot
which chord should start the rotation? (positions 2-4): 2
rotation PI pathCretRot added to PathInstances.

pi{pathCretRot}ti{} :: pils
PathInstances available:
{name,TIrefs,scPath}
   pathA            0  3-2B,4-9,3-11A,4-6
   pathC            0  5-3A,6-4,7-34,4-14A
   pathCret         0  4-14A,7-34,6-4,5-3A
 + pathCretRot      0  7-34,6-4,5-3A,4-14A
```

A slice will extract a segment from a Path. To create a slice, enter PIslc. The user is prompted for the name of the new Path, and the start and end Multiset positions. If the slice is to only contain the last two chords of a four chord Path, for example, the start and end positions would be 3,4:

Example 3-13. Creating a slice of a Path with PIslc

```
pi{pathCretRot}ti{} :: pislc
```

```
name this slice of path pathCretRot: pathD
which chords should bound the slice? (positions 1 - 4): 3,4
slice PI pathD added to PathInstances.

pi{pathD}ti{} :: pils
PathInstances available:
{name,TIrefs,scPath}
   pathA            0  3-2B,4-9,3-11A,4-6
   pathC            0  5-3A,6-4,7-34,4-14A
   pathCret         0  4-14A,7-34,6-4,5-3A
   pathCretRot      0  7-34,6-4,5-3A,4-14A
 + pathD            0  5-3A,4-14A
```

There are three ways to edit a single Multiset within a Path using the PIe command: by replacement, by transposition, or by inversion. In all cases, the number of elements in the Multiset must be maintained.

To edit a single Multiset in a Path enter PIe:

Example 3-14. Transposing a set within a Path

```
pi{pathD}ti{} :: pie
edit PI pathD
enter position to edit (positions 1-2): 2
replace, transpose, or invert set (0,2,3,7): (r, t, or i): t
enter a transposition method: literal or modulus? (l or m): l
enter a positive or negative transposition: 8
PI pathD edited.

pi{pathD}ti{} :: piv
PI: pathD
psPath              0,1,2,4,5        8,10,11,15
                    C4,C#4,D4,E4,F4  G#4,A#4,B4,D#5
pcsPath             0,1,2,4,5        8,10,11,3
scPath              5-3A             4-14A
durFraction         1(50%)           1(50%)
TI References: none.
```

Here the user has selected the Multiset in position "2" of PI "pathD" to edit. The user next selects to edit the set by transposition, entering "t". There are two methods of transposition available: a "literal" transposition is done in pitch space, creating a new set in the range of all positive and negative integers; a "modulus" transposition is done in pitch-class space, creating a new set in the range of pitch-classes 0 through 11. In the example above the user has selected a literal ("l") transposition and enters "8" as the transposition value. This shifts each pitch in the Multiset up 8 half-steps. Since this is a literal and not a modulus transposition, pitch 5 becomes pitch 15, or D#5.

Any Multiset in a Path can be replaced with a Multiset of equal size. For example, the same Multiset edited above can be replaced with any four-element Multiset:

Example 3-15. Replacing a Multiset with a new Multiset

```
pi{pathD}ti{} :: pie
edit PI pathD
```

```
enter position to edit (positions 1-2): 2
replace, transpose, or invert set (8,10,11,15): (r, t, or i): r
enter a pitch set, sieve, spectrum, or set-class: 2,2,4,4
      SC 2-2 as (D4,D4,E4,E4)? (y, n, or cancel): y
PI pathD edited.

pi{pathD}ti{} :: piv
PI: pathD
psPath              0,1,2,4,5        2,2,4,4
                    C4,C#4,D4,E4,F4  D4,D4,E4,E4
pcsPath             0,1,2,4,5        2,2,4,4
scPath              5-3A             2-2
durFraction         1(50%)           1(50%)
TI References: none.
```

Chapter 4. Tutorial 4: Creating and Editing Textures

This tutorial demonstrates basic Texture creation, configuration, and deployment in musical structures. This chapter is essential for using athenaCL for algorithmic music production.

4.1. Introduction to Textures and ParameterObjects

A TextureInstance (or a Texture or TI) is an algorithmic music layer. Like a track or a part, a Texture represents a single musical line somewhat analogous to the role of a single instrument in an ensemble. The music of a Texture need not be a single monophonic line: it may consist of chords and melody, multiple independent lines, or any combination or mixture. The general generative shape and potential of a Texture is defined by the TextureModule. A Texture is an instance of a TextureModule: a single TextureModule type can be used to create many independent instances of that type; each of these instances can be customized and edited independently. Collections of TextureInstances are used to create an EventList, or the musical output of all Textures.

A TextureInstance consists of many configurable slots, or attributes. These attributes allow the user to customize each Texture. Attributes include such properties as timbre (instrument and parametric timbre specifications), rhythm (duration and tempo), frequency materials (Path, transposition, and octave position), and mixing (amplitude and panning). Other attributes may control particular features of the Texture, like the number of voices, position of chords, or formal properties.

Most attributes of a TextureInstance are not fixed values. Unlike a track or a part, a Texture often does not have a fixed sequence of values for attributes like amplitude, or even fixed note-sequences. Rather, attributes of a Texture are algorithmic objects, or ParameterObjects. Rather than enter a value for amplitude, the user chooses a ParameterObject to produce values for the desired attribute, and enters settings to specialize the ParameterObject's behavior. Rather than enter note-sequences, the Texture selects and combines pitches from a Path, or a user-supplied sequence of pitch groups. In this way each attribute of a Texture can be given a range of motion and a degree of indeterminacy within user-composed boundaries.

A TextureInstance is not a fixed entity: it is a collection of instructions on how to create events for a certain duration. Every time an EventList is created, each Texture is "performed," or called into motion to produce events. Depending on the TextureModule and the Texture's configuration, the events produced may be different each time the EventList is created.

athenaCL is designed to allow users work with broad outlines of musical parameters and materials, and from this higher level organize and control combinations of Textures. This should not be confused with a much higher level of algorithmic composition, where an algorithm is responsible for creating an entire composition: its style, form, parts, and surface. athenaCL is unlikely to produce such "complete" musical structures. Rather, athenaCL is designed to produce complex, detailed, and diverse musical structures and surfaces. Combinations of parts and construction of form are left to the user, and can be composed either in athenaCL or in a Digital Audio Workstation where athenaCL EventOutput formats, such as MIDI files or Csound-rendered audio files, can be mixed, processed, and combined in whatever desired fashion. Alternatively, MIDI files produced with athenaCL can be modified or combined in traditional sequencers and notation editors.

4.2. Introduction Instrument Models

athenaCL features numerous integrated instrument models. In some cases these instrument models are references to external specifications; in other cases these instrument models contain complete source code necessary for instantiating synthesis systems. Textures are assigned an instrument from an Orchestra upon creation, and are able to control a wide variety of instrument-specific parameters.

athenaCL features an integrated library of Csound instruments, providing automated control of both Csound score and orchestra generation and control. For details on installing and using Csound within athenaCL, see Section 2.6. Csound instruments are signal processing and synthesis instructions. These instructions designates a certain number of parameters to expose to the user of the instrument. These parameters allow events in the score to communicate information and settings to the instrument. athenaCL's integrated library of Csound instruments permits dynamically constructed orchestra files to be used with athenaCL-generated Csound scores. Alternatively, users can use external, custom orchestras with athenaCL-written score files. EventModes csoundNative, csoundExternal, and csoundSilence support diverse ways of working with Csound within athenaCL.

athenaCL provides instrument collections (Orchestras) for working with other EventOutput formats. For working with MIDI systems, General MIDI (GM) instrument definitions are provided with the generalMidi and generalMidiPercussion EventModes.

Whenever a Texture is created, an instrument must be specified by number. This is necessary because the Texture must be configured with additional ParameterObjects for instrument-specific parameter control. Instruments are always identified by a number, though within athenaCL descriptive names are provided when available.

The instruments available during Texture creation are dependent on the active EventMode: that is, for any active EventMode, one Orchestra is available from which a Texture's instrument must be selected. In the following example, the user lists available EventModes to check that csoundNative is active, and then views the available instruments with the EMi command.

Example 4-1. Listing available Instruments with EMi

```
pi{}ti{} :: emls
EventMode modes available:
{name}
   csoundExternal
 + csoundNative
   csoundSilence
   midi
   midiPercussion
   superColliderNative

pi{}ti{} :: 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
```

Other EventModes provide other Orchestras for use in Textures. In the example below, the user selects the EventMode midiPercussion with the EMo command and examines the available instruments with the EMi command:

Example 4-2. Examining additional Instruments with EMi

```
pi{}ti{} :: emo mp
EventMode mode set to: midiPercussion.

pi{}ti{} :: emi
generalMidiPercussion instruments:
{number,name}
   35     acousticBassDrum
   36     bassDrum1
   37     sideStick
   38     acousticSnare
   39     handClap
   40     electricSnare
   41     lowFloorTom
   42     closedHiHat
   43     highFloorTom
   44     pedalHiHat
   45     lowTom
   46     openHiHat
   47     lowMidTom
   48     hiMidTom
   49     crashCymbal1
   50     highTom
   51     rideCymbal1
   52     chineseCymbal
   53     rideBell
   54     tambourine
   55     splashCymbal
   56     cowBell
   57     crashCymbal2
   58     vibraSlap
   59     rideCymbal2
   60     hiBongo
   61     lowBongo
   62     muteHiConga
   63     openHiConga
   64     lowConga
   65     highTimbale
   66     lowTimbale
   67     highAgogo
   68     lowAgogo
   69     cabasa
   70     maracas
   71     shortWhistle
   72     longWhistle
   73     shortGuiro
   74     longGuiro
   75     claves
   76     hiWoodBlock
   77     lowWoodBlock
   78     muteCuica
   79     openCuica
   80     muteTriangle
   81     openTriangle
```

4.3. Selecting and Viewing TextureModules

A Texture is an instance of a TextureModule. Every time a Texture is created, athenaCL creates an independent instance of the active TextureModule. To display a complete list of all available TextureModules, enter the command TMls:

Example 4-3. Listing TextureModules with TMls

```
pi{}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
```

As in other athenaCL displays, the first line of the display is a key, telling the user that the list consists of a name followed by the number of TI references. This number displays the count of TextureInstances referenced from a parent TextureModule. The "+" designates the active TextureModule. When creating a new TextureInstance, athenaCL uses the active TextureModule.

To select a different TextureModule, the user enters TMo. The user is prompted to enter the name or number (as represented in the list order) of the desired TextureModule. The TMls command can be used to confirm the change.

Example 4-4. Selecting the active TextureModule with TMo

```
pi{}ti{} :: tmo
which TextureModule to activate? (name or number 1-14): da
TextureModule DroneArticulate now active.

pi{}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
```

Here the user has entered "da", to select the TextureModule DroneArticulate. Whenever selecting objects in athenaCL the user may enter the acronym (formed from the leading character and all

following capitals), the literal name ("dronearticulate"), or the ordinal number as displayed in the corresponding list display.

To learn what a particular TextureModule does, as well what types of Texture options are available, enter the command TMv, for TextureModule View. In this example, the user, with TIo, selects the TextureModule "LineGroove" (with a command-line argument) before entering the TMv command.

Example 4-5. Viewing details of the active TextureModule

```
pi{}ti{} :: tmo linegroove
TextureModule LineGroove now active.

pi{}ti{} :: tmv
TextureModule: LineGroove; author: athenaCL native
This TextureModule performs each set of a Path as a simple monophonic line;
pitches are chosen from sets in the Path based on the pitch selector control.
texture (s)tatic
parallelMotionList    Description: List is a collection of transpositions
                      created above every Texture-generated base note. The
                      timeDelay value determines the amount of time in seconds
                      between each successive transposition in the
                      transpositionList. Arguments: (1) name, (2)
                      transpositionList, (3) timeDelay
pitchSelectorControl  Description: Define the selector method of Path pitch
                      selection used by a Texture. Arguments: (1) name, (2)
                      selectionString {'randomChoice', 'randomWalk',
                      'randomPermutate', 'orderedCyclic',
                      'orderedCyclicRetrograde', 'orderedOscillate'}
levelFieldMonophonic  Description: Toggle between selection of local field
                      (transposition) values per set of the Texture Path, or per
                      event. Arguments: (1) name, (2) level {'set', 'event'}
levelOctaveMonophonic 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'}
texture (d)ynamic
```

The TMv command displays the name of the TextureModule along with the author of its code. Following the author designation is a description of how the module performs. Following this is documentation for all TextureStatic parameter objects, or Texture-specific options and user-configurable settings pertinent to the particular TextureModule's algorithmic design.

4.4. Creating, Selecting, and Viewing TextureInstances

A TextureInstance is always linked to a Path. If no Paths exists when the Texture is created, a default Path is automatically created consisting of a single Multiset with a single pitch (middle C, or C4). If Paths exists when the Texture is created, the active PathInstance is assigned to the Texture. A TextureInstance's Path can be later edited. For a complete introduction to Paths see Chapter 3.

A new TextureInstance is always created from the active TextureModule; the user must then always select the desired TextureModule before creating a Texture of the desired type. A TextureInstance's type, or TextureModule, cannot be changed after the Texture is created.

A new Texture is created with the TIn command, for TextureInstance New. The user is prompted to name the new Texture and select an instrument by number. If the number of the desired instrument is not known, a "?" can be entered to display a list of instruments. In the example below the user selects TextureMode LineGroove, EventMode midiPercussion, and then creates a texture named "a1" with instrument 64 ("lowConga").

Example 4-6. Creating a new TextureInstance with TIn

```
pi{}ti{} :: tmo linegroove
TextureModule LineGroove now active.

pi{}ti{} :: emo mp
EventMode mode set to: midiPercussion.

pi{}ti{} :: tin
name this texture: a1
enter instrument number:
(35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,6
1,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81)
or "?" for instrument help: 64
TI a1 created.
```

To hear the resulting musical structure, enter the command ELn. (For more information on using ELn, see Section 2.5. The resulting MIDI file may be opened with the ELh command.

Example 4-7. Creating a new EventList with ELn

```
pi{auto-lowConga}ti{a1} :: eln
command.py: temporary file: /Volumes/xdisc/_scratch/ath2010.07.02.17.51.42.xml
      EventList ath2010.07.02.17.51.42 complete:
/Volumes/xdisc/_scratch/ath2010.07.02.17.51.42.mid
/Volumes/xdisc/_scratch/ath2010.07.02.17.51.42.xml
```

After creating a Texture, the TIv command can be used to view the active Texture:

Example 4-8. Viewing a TextureInstance

```
pi{auto-lowConga}ti{a1} :: tiv
TI: a1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, duration: 000.0--20.06
(i)nstrument        64 (generalMidiPercussion: lowConga)
(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              auto-lowConga
                    (E4)
                    20.00(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            parallelMotionList, (), 0.0
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

The TIv command displays all essential attributes of a Texture. Each label in the display corresponds to an attribute in the TextureInstance. The TIv display is in two-blocks. The first block gives parameters that are constant. The first line displays the name of the TextureInstance (a1), the name of the parent TextureModule (LineGroove), the number of TextureClones (0), and the active TextureTemperament (TwelveEqual). The second line displays the PitchMode (pitchSpace), the silenceMode (off), and the postMapMode (on). The third line provides the GM MIDI program name (piano1). The fourth, indented line displays the TextureInstance's mute status (where a "o" is muted and a "+" is non-muted) and the absolute duration the Texture's events.

The second block lists the primary algorithmic controls of the Texture. The names of these attributes use parenthesis to designate a single-letter abbreviation. The instrument attribute is displayed first, with the value following the label: instrument number (64), the name of the orchestra (generalMidiPercussion) and the name of the instrument (lowConga). The next attribute is time-range, the start and end time in seconds from the beginning of the EventList. A new Texture is given a default time-range of twenty seconds (00.0--20.0). New Textures, when created, get their time-range from the active Texture.

The bpm attribute is the tempo in beats per minute. The value is set with the ParameterObject "constant" to produce a tempo of 120 BPM. In most cases, the bpm control is used to calculate the duration of rhythms and pulses used in a Texture.

The rhythm attribute designates a Rhythm ParameterObject to control the generation of event durations. Rhythm ParameterObjects often notate rhythms as lists of Pulses. A Pulse is designated as a list of three elements: (divisor, multiplier, accent). The duration of a rhythm is calculated by dividing the time of a beat (from the bpm parameter) by the Pulse divisor, then multiplying the result by the Pulse multiplier. The value of the "accent" determines if a duration is a note or a rest, where 1 or a "+" designates a note, 0 or a "o" designates a rest. Thus an eighth note is given as (2,1,1), a dotted-quarter note as (2,3,1), and dotted-eighth rest as (4,3,0). In the example above, the ParameterObject "loop" is used with three Pulses: two sixteenth notes (4,1,1) and a duration equal to a quarter-note tied to a sixteenth note (4,5,1).

The Path attribute gives the name of the PathInstance used by this Texture, followed on the next line by the Multiset pitches that will be used. PathInstances are linked to the Texture. Thus, if a change is made to a Path (with PIe, for example), all Textures that use that Path will reflect the change. Each TextureInstance, however, can control the interpretation of a Path in numerous ways. The Texture PitchMode setting, for example, determines if pitches are derived from a Path in pitchSpace, pitchClassSpace, or as a setClass. The local field and local octave attributes permit each Texture to transpose pitches from the Path independently.

The attribute "local field" stores a ParameterObject that controls local transposition of Path pitches. Values are given in semitones, and can include microtonal transpositions as floating-point values following the semitone integer. Thus, a transposition of five half-steps and a quarter-tone would be equal to 5.5. A transposition of a major tenth would be 16. In the example above the attribute value instructs the Texture to use a ParameterObject called "constant." Note: some EventOutput formats do not support microtonal pitch specification. In such cases microtones are rounded to the nearest semitone. The attribute "local octave," similar to local field, controls the octave position of Path pitches. Each integer represents an octave shift, where 0 is no octave shift, each Path pitch retaining its original octave register.

The amplitude attribute designates a ParameterObject to control the amplitude of the Texture, measured in a symbolic range from 0 to 1. The panning attribute designates the ParameterObject used to control spatial location in stereo or quadraphonic space. Values are along the unit interval, from 0 to 1.

The attributes that make up the "auxiliary" listing provide any number of additional ParameterObjects to control instrument specific parameter fields. The number of parameter fields is determined by the instrument definition.

The last attributes, "texture static" and "texture dynamic," designate controls specific to particular TextureModules. The values here can be edited like other attributes.

A second Texture will be created with TIn named "b1" and using instrument 62. The Texture, after creation, is displayed with the TIv command.

Example 4-9. Creating and viewing a TextureInstance

```
pi{auto-lowConga}ti{a1} :: tin
name this texture: b1
enter instrument number:
(35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,6
1,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81)
or "?" for instrument help: 62
TI b1 created.

pi{auto-muteHiConga}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, duration: 000.0--20.06
(i)nstrument        62 (generalMidiPercussion: muteHiConga)
(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              auto-muteHiConga
                    (D4)
                    20.00(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            parallelMotionList, (), 0.0
```

```
      s1            pitchSelectorControl, randomPermutate
      s2            levelFieldMonophonic, event
      s3            levelOctaveMonophonic, event
texture (d)ynamic   none
```

This new Texture, though created with the same TextureModule, is a completely autonomous object. No changes to "a1" will have any effect on "b1".

During an athenaCL session a user can create any number of TextureInstances and save this collection in an AthenaObject file for latter use. For more information on saving, loading, and merging AthenaObjects see Chapter 2. To view a list of all current Textures, enter the command TIls, for TextureInstance List.

Example 4-10. Listing all TextureInstances

```
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               + LineGroove  auto-muteHiConga 62  00.0--20.0   0
```

This display shows a list of all Textures, each Texture on a single line. The information given, in order from left to right, is the name, the mute-status, the parent TM, the PathInstance, the instrument number, the time-range, and the number of TextureClones. Notice the "+" in front of Texture "b1": this designates that this Texture is active. To change the active Texture, enter the command TIo either with a command-line argument or alone:

Example 4-11. Selecting the active TextureInstance

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

pi{auto-muteHiConga}ti{a1} ::
```

In order to compare a single attribute of all Textures, the user can enter the command TEv, for TextureEnsemble View. TextureEnsemble refers to the collection of all Textures, and all TE commands process all Textures simultaneously. The user will be prompted to enter an abbreviation for the desired attribute. Attribute abbreviations are notated in the TIv display labels. Thus the attribute abbreviation for "(a)mplitude" is "a"; the attribute abbreviation for "pan(n)ing" is "n." As with other commands, use of command-line arguments provides flexible control:

Example 4-12. Viewing parameter values for all Textures

```
pi{auto-muteHiConga}ti{a1} :: tev
compare texture parameters: which parameter? a
compare parameters: amplitude
{name,value,}
a1                  randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
```

```
b1                  randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)

pi{auto-muteHiConga}ti{a1} :: tev i
compare parameters: instrument
{name,value,}
a1                  64 (generalMidiPercussion: lowConga)
b1                  62 (generalMidiPercussion: muteHiConga)
```

4.5. Copying and Removing Texture Instances

TextureInstances can be duplicated with the command TIcp. The user is prompted to enter the name of the Texture to be copied, and then the name of the copy. The copy can be confirmed by listing all Textures with the command TIls.

Example 4-13. Copying a TextureInstance

```
pi{auto-muteHiConga}ti{a1} :: ticp
which TextureInstnace to copy? (name or number 1-2): b1
name this copy of TI 'b1': b2
TextureInstance b2 created.

pi{auto-muteHiConga}ti{b2} :: tils
TextureInstances available:
{name,status,TM,PI,instrument,time,TC}
   a1               + LineGroove  auto-lowConga    64  00.0--20.0   0
   b1               + LineGroove  auto-muteHiConga 62  00.0--20.0   0
 + b2               + LineGroove  auto-muteHiConga 62  00.0--20.0   0
```

Textures can be deleted with the command TIrm, for TextureInstance Remove. The user is prompted to enter the name of the Texture to be deleted. The removal can be confirmed by listing all Textures with the command TIls.

Example 4-14. Removing a TextureInstance

```
pi{auto-muteHiConga}ti{b2} :: tirm
which TextureInstnace to delete? (name or number 1-3): b2
are you sure you want to delete texture b2? (y, n, or cancel): y
TI b2 destroyed.

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               + LineGroove  auto-muteHiConga 62  00.0--20.0   0

pi{auto-muteHiConga}ti{b1} ::
```

When the active Texture is deleted, as it is above, athenaCL chooses a new Texture to activate, here choosing "b1." To select a different Texture, use the command TIo.

4.6. Editing TextureInstance Attributes

Each attribute of a Texture can be edited to specialize its performance. Some attributes such as instrument, time-range, and Path are static: they do not change over the duration of a Texture. Other attributes are dynamic, such as bpm, rhythm, local field, local octave, amplitude and panning, and can be configured with a wide range of ParameterObjects.

Texture attributes are edited with the TIe command. The command first prompts the user to select which attribute to edit. Attributes are named by a single-letter abbreviation, as notated in the TIv display with parenthesis. Next, the current value of the attribute is displayed, followed by a prompt for the new value. In the following example the time range of Texture "a1" is edited:

Example 4-15. Editing 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): t
current time range: 0.0, 20.0
new value: 5, 20
TI b1: parameter time range updated.

pi{auto-muteHiConga}ti{b1} :: tiv
TI: b1, TM: LineGroove, TC: 0, TT: TwelveEqual
pitchMode: pitchSpace, silenceMode: off, postMapMode: on
midiProgram: piano1
      status: +, 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         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
```

In the example above the user select "t" to edit the active Texture's time-range attribute. In general, new values for attributes must be entered with the same syntax with which they are displayed. In this example, time-range values are given as two numbers separated by a comma. Deviation from this syntax will return an error. The user enters 5, 20 to set the time-range attribute to the duration from 5 to 20 seconds.

The command TEe, for TextureEnsemble Edit can be used to edit the entire collection of Textures with one command. In the following example the user selects the amplitude attribute with "a" and then enters a new ParameterObject: randomUniform. The randomUniform parameterObject

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

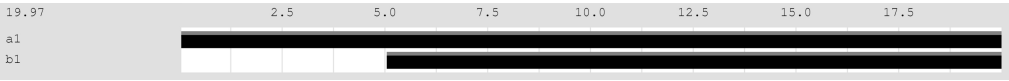
```
pi{auto-muteHiConga}ti{b1} :: tee
edit all TextureInstances
which parameter? (i,t,b,r,p,f,o,a,n,x): a
sample amplitude: randomBeta, 0.4, 0.4, (constant, 0.7), (constant, 0.9)
new value: ru, .6, 1
TI a1: parameter amplitude updated.
TI b1: parameter amplitude updated.

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                                 ____________________________________________
```

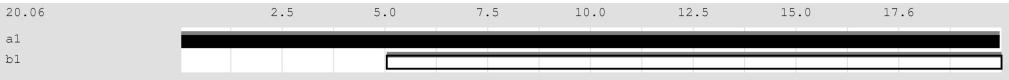


4.7. Muting Textures

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:

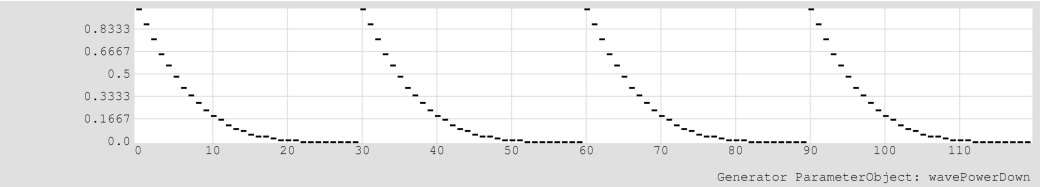
Example 4-21. Viewing ParameterObject reference information

```
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

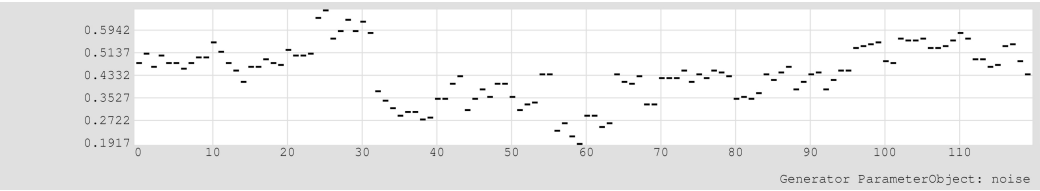
```
pi{auto-muteHiConga}ti{b1} :: tpmap
select a library: Generator, Rhythm, or Filter. (g, r, f): g
number of events: 120
enter a Generator ParameterObject argument: wpd, e, 30, 0, 2

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
```

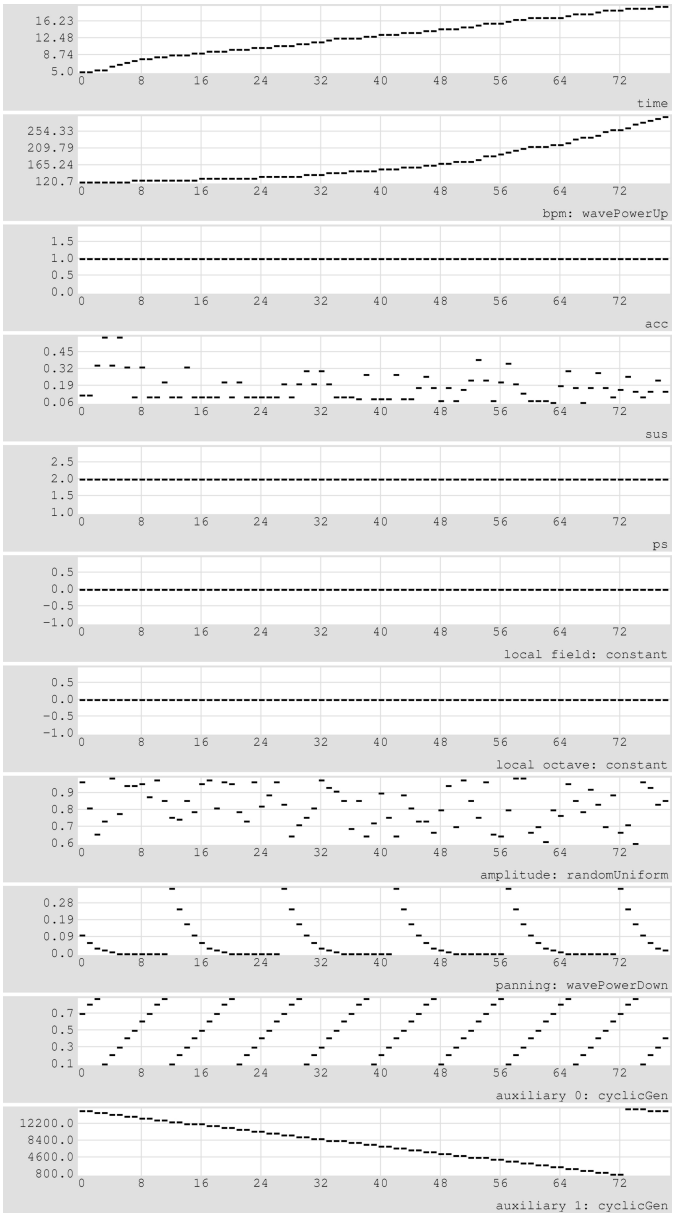
4.12. Displaying Texture Parameter Values

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

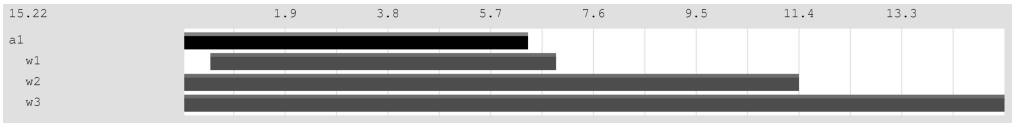
```
pi{auto}ti{a1} :: tpv retrograde
Clone Static ParameterObject
{name,documentation}
retrogradeMethodToggle retrogradeMethodToggle, name
                       Description: Selects type of retrograde transformation
                       applied to Texture events. Arguments: (1) name, (2) name
   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

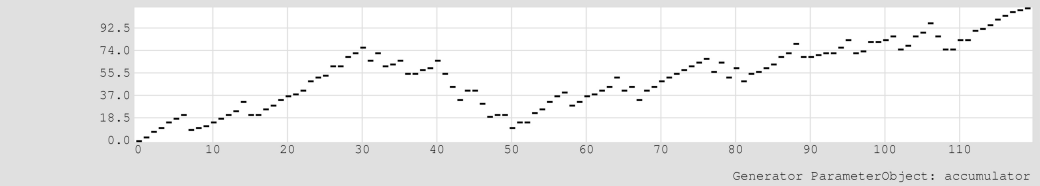
accumulator, initValue, parameterObject

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}

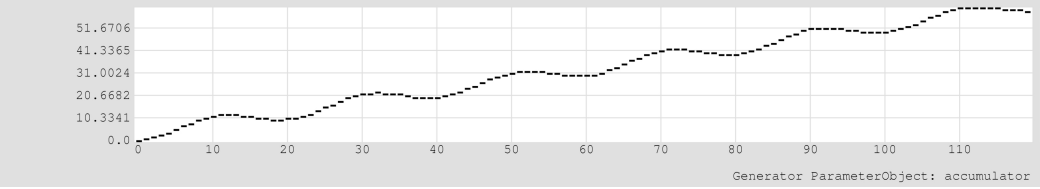
Sample Arguments: `a, 0, (bg,rc,(1,3,4,7,-11))`

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)

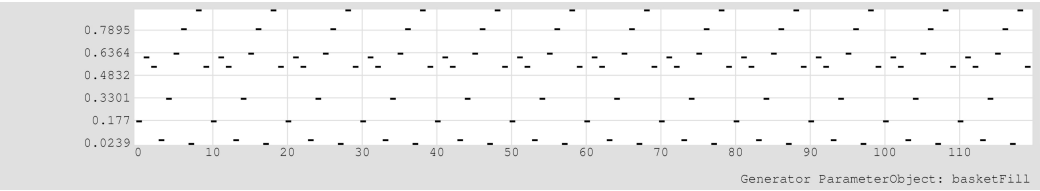
basketFill, selectionString, parameterObject, valueCount

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)

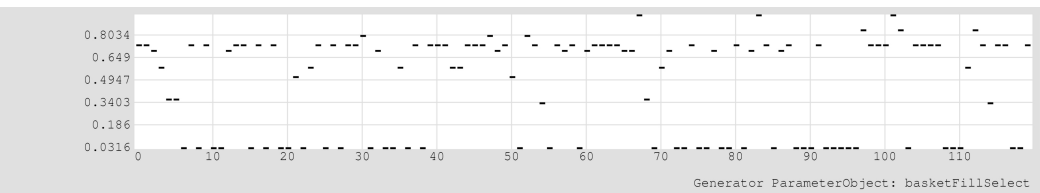
basketFillSelect, parameterObject, valueCount, parameterObject

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)

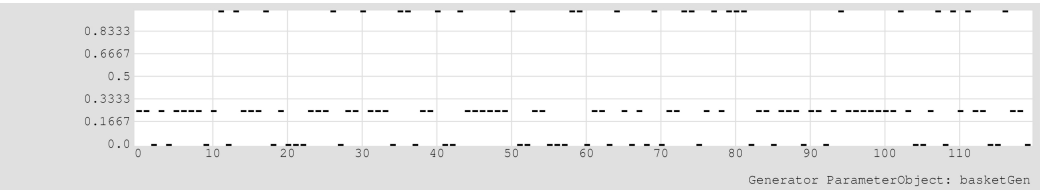
basketGen, selectionString, valueList

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

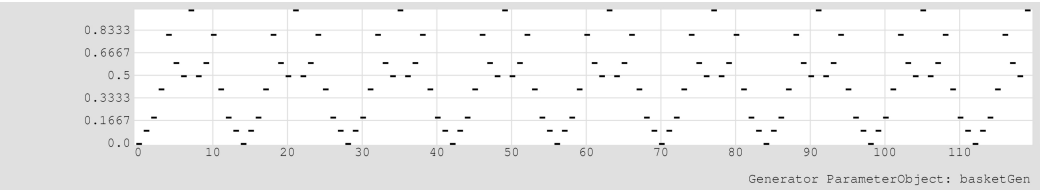
Sample Arguments: `bg, rc, (0,0.25,0.25,1)`

Example C-5. basketGen Demonstration 1



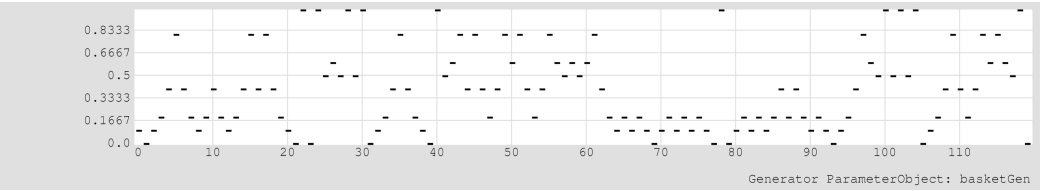

```
basketGen, randomChoice, (0,0.25,0.25,1)
```

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)

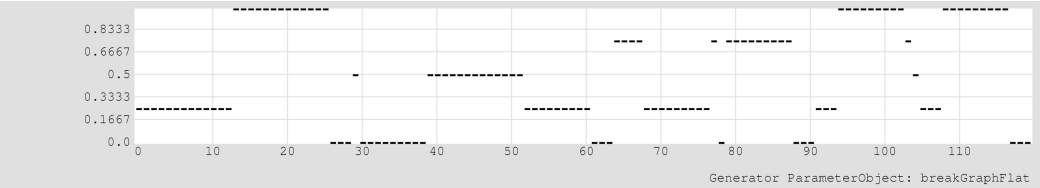
breakGraphFlat, stepString, edgeString, parameterObject, parameterObject, pointCount

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)

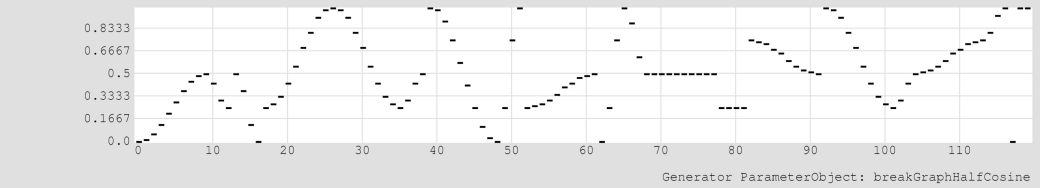
breakGraphHalfCosine, stepString, edgeString, parameterObject, parameterObject, pointCount

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)

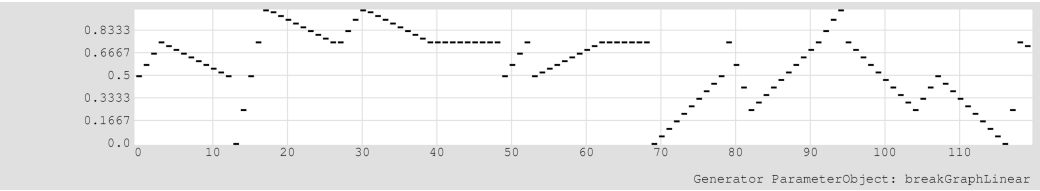
breakGraphLinear, stepString, edgeString, parameterObject, parameterObject, pointCount

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)

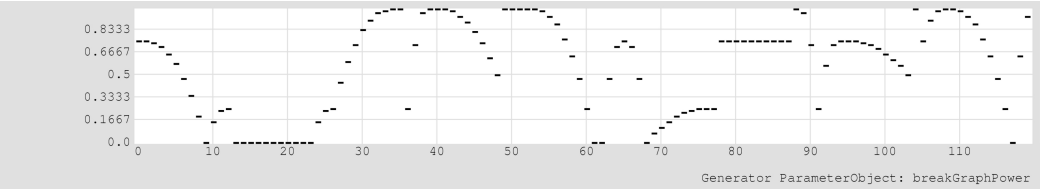
breakGraphPower, stepString, edgeString, parameterObject, parameterObject, pointCount, exponent

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)

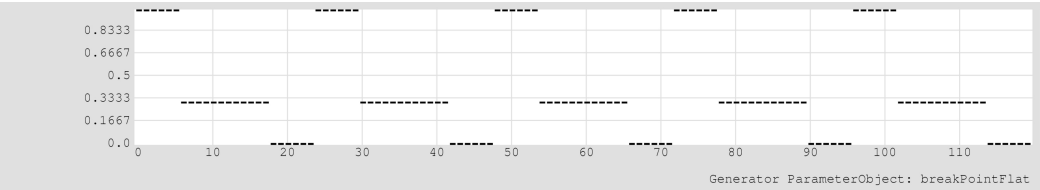
breakPointFlat, stepString, edgeString, pointList

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

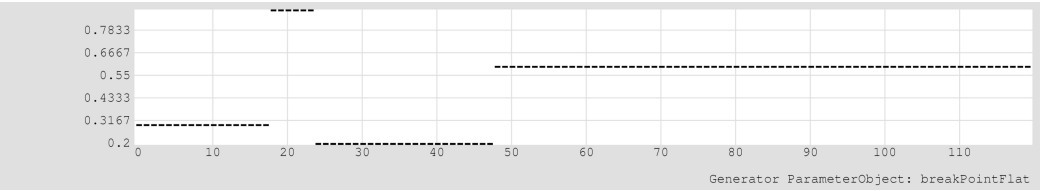
Sample Arguments: `bpf, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))`

Example C-12. breakPointFlat Demonstration 1



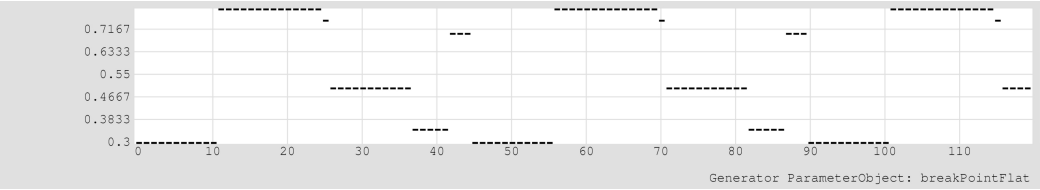

```
breakPointFlat, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))
```

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)

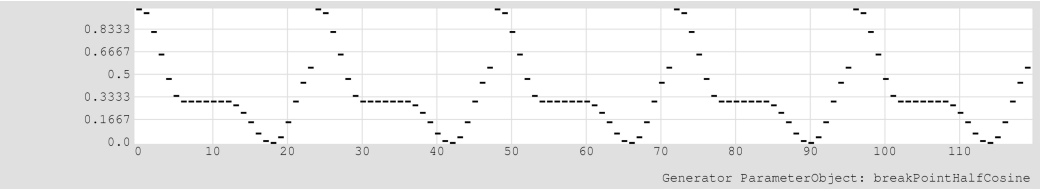
breakPointHalfCosine, stepString, edgeString, pointList

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

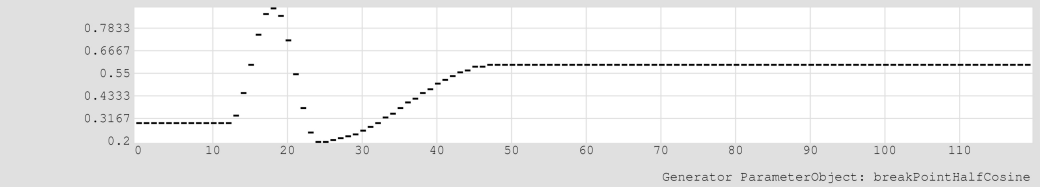
Sample Arguments: `bphc, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))`

Example C-15. breakPointHalfCosine Demonstration 1



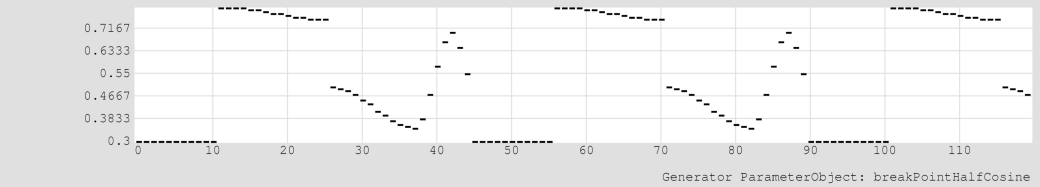

```
breakPointHalfCosine, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))
```

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

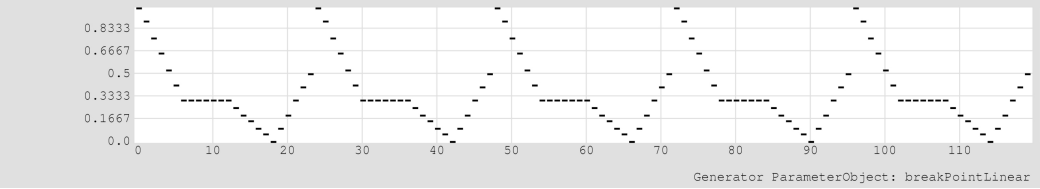
Description: Provides a break-point function with linear interpolation from a list of (x,y) coordinate pairs (pointList). A step type (stepString) determines if x values in the pointList refer to events or

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

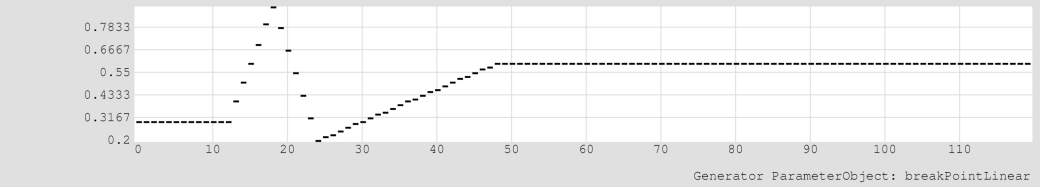
Sample Arguments: `bpl, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))`

Example C-18. breakPointLinear Demonstration 1



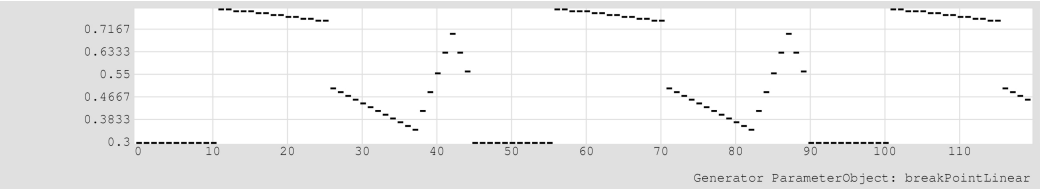

```
breakPointLinear, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6))
```

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)

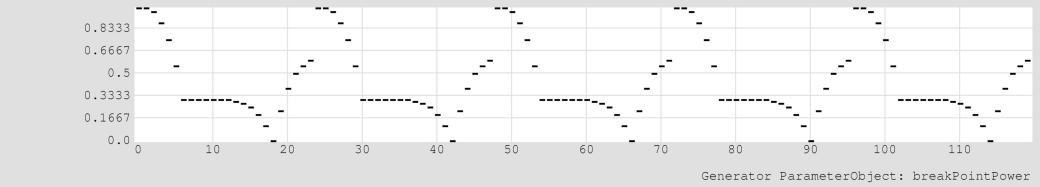
breakPointPower, stepString, edgeString, pointList, exponent

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

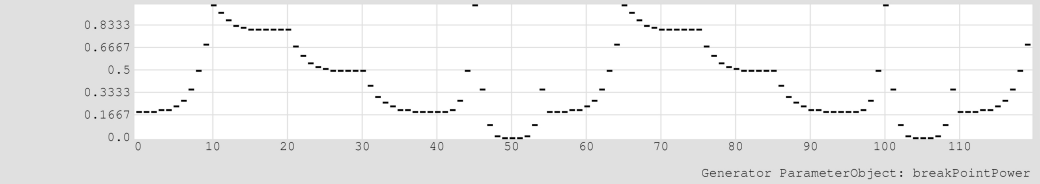
Sample Arguments: `bpp, e, l, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6)), -1.5`

Example C-21. breakPointPower Demonstration 1



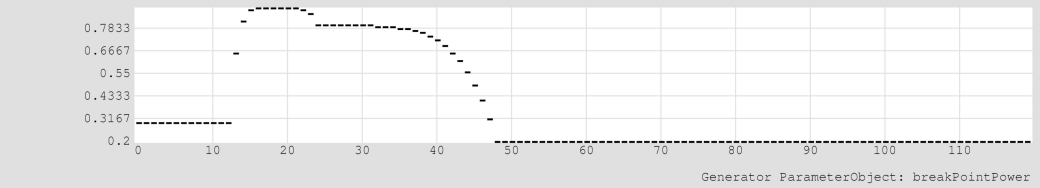

```
breakPointPower, event, loop, ((0,1),(6,0.3),(12,0.3),(18,0),(24,0.6)), -1.5
```

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)

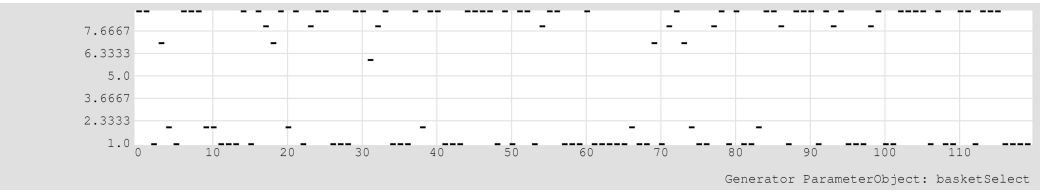
basketSelect, valueList, parameterObject

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)

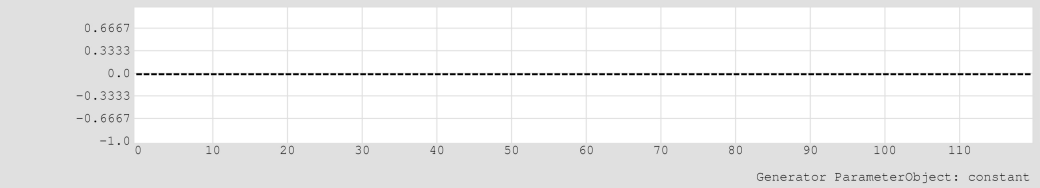
constant, value

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)

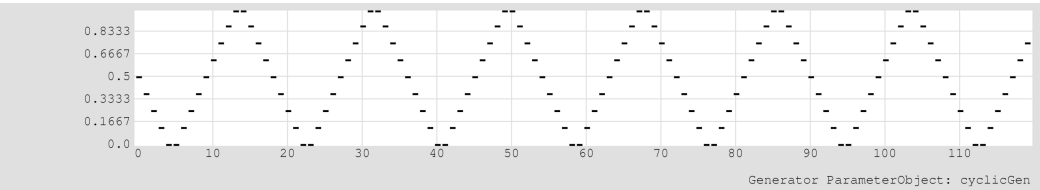
cyclicGen, directionString, min, max, increment

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

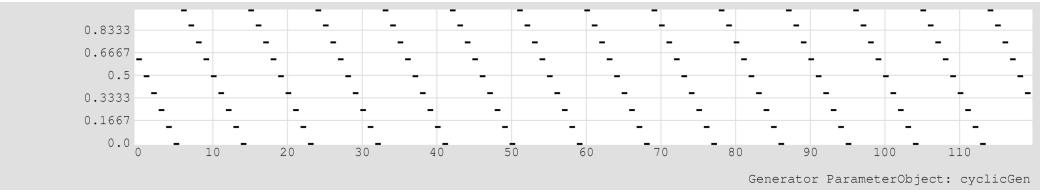
Sample Arguments: `cg, ud, 0, 1, 0.13`

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)

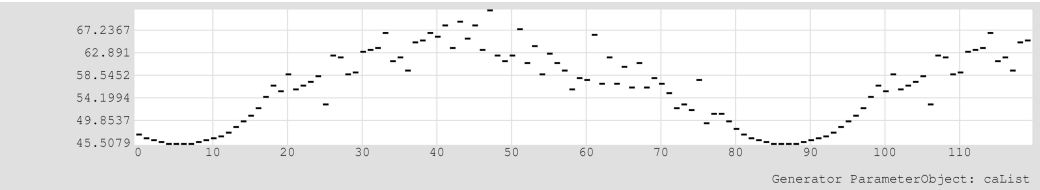
caList, caSpec, parameterObject, parameterObject, tableExtractionString, selectionString

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'}

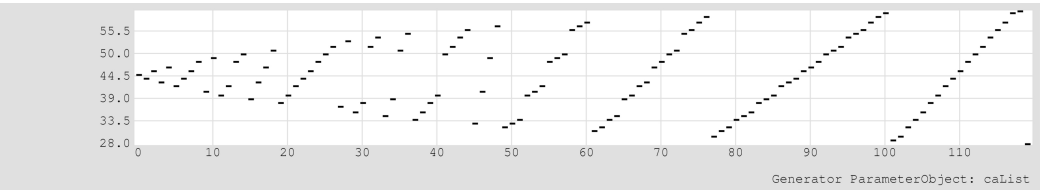
Sample Arguments: `cl, f{f}i{c}x{81}y{120}, 0.25, 0.0005, sc, oc`

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

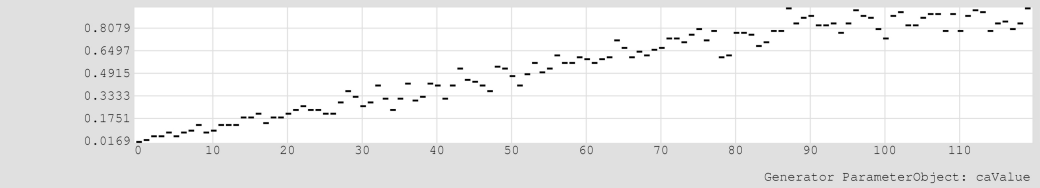
Description: Produces values from a one-dimensional cellular automata table scaled within dynamic min and max values. One dimensional cellular automata may be standard, totalistic, continuous, or float formats, and are defined with a caSpec string. The caSpec string may contain one or more CA parameters 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'}

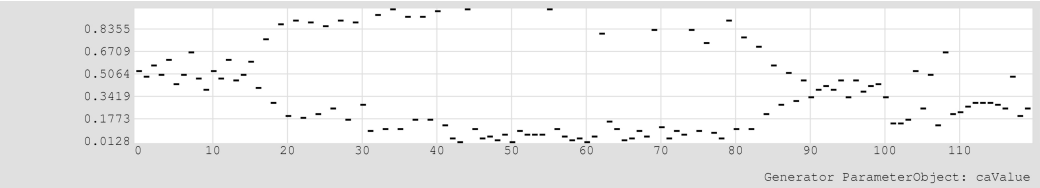
Sample Arguments: `cv, f{s}, (c,110), (c,0), sr, 0, 1, oc`

Example C-30. caValue Demonstration 1



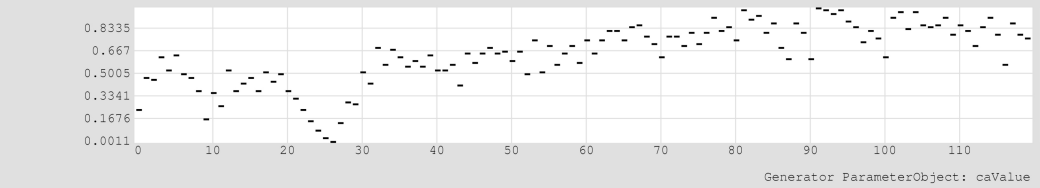

```
caValue, f{s}k{2}r{1}i{center}x{91}y{135}w{91}c{0}s{0}, (constant, 110),
(constant, 0), sumRow, (constant, 0), (constant, 1), orderedCyclic
```

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

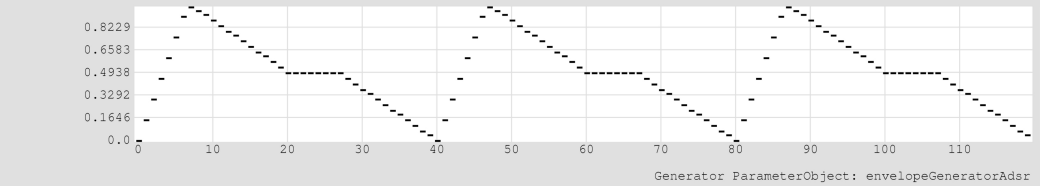
Description: Generates a sequence of dynamic envelopes with durations controlled by a Generator Parameter Object. Envelope duration is specified by the duration ParameterObject; all values are interpreted in seconds. The scaleString parameter determines if shape values are interpreted as proportional 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

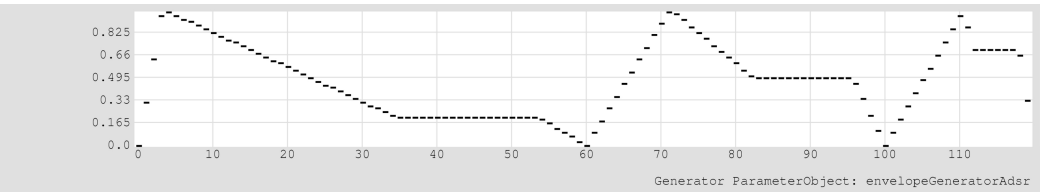
Sample Arguments: `ega, proportional, 1, 100, (c,40), (c,2), (c,4), (c,2), (c,4), (c,0.5), 0, 1`

Example C-33. envelopeGeneratorAdsr Demonstration 1



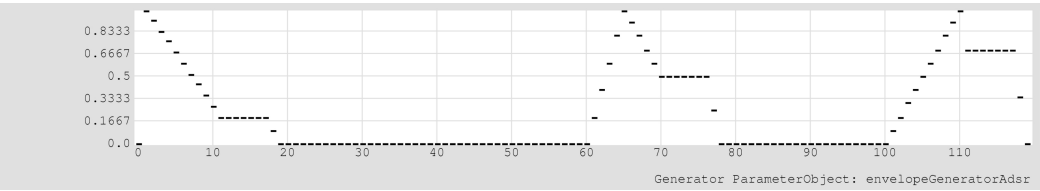

```
envelopeGeneratorAdsr, proportional, loop, 100, (constant, 40), (constant, 2),
(constant, 4), (constant, 2), (constant, 4), (constant, 0.5), (constant, 0),
(constant, 1)
```

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)

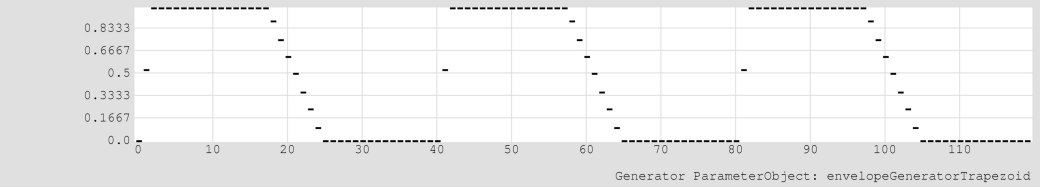
envelopeGeneratorTrapezoid, scaleString, edgeString, eventCount, parameterObject, parameterObject, parameterObject, parameterObject, parameterObject, min, max

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

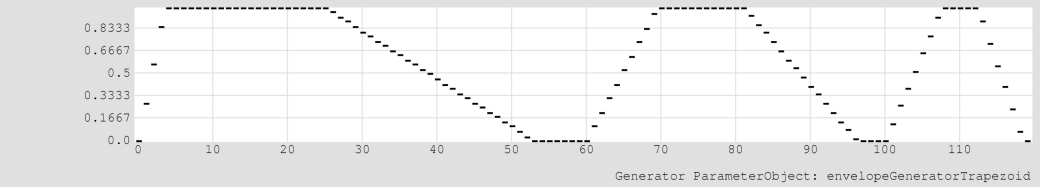
Sample Arguments: `egt, proportional, 1, 100, (c,40), (c,0.5), (c,4), (c,2), (c,4), 0, 1`

Example C-36. envelopeGeneratorTrapezoid Demonstration 1



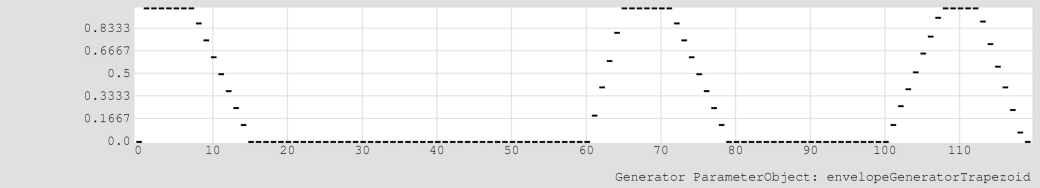

```
envelopeGeneratorTrapezoid, proportional, loop, 100, (constant, 40),
(constant, 0.5), (constant, 4), (constant, 2), (constant, 4), (constant, 0),
(constant, 1)
```

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)

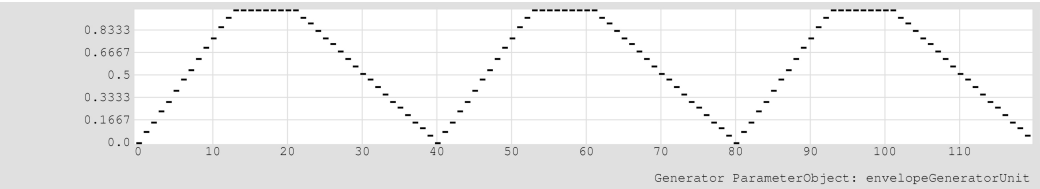
envelopeGeneratorUnit, edgeString, eventCount, parameterObject, parameterObject, parameterObject, min, max

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

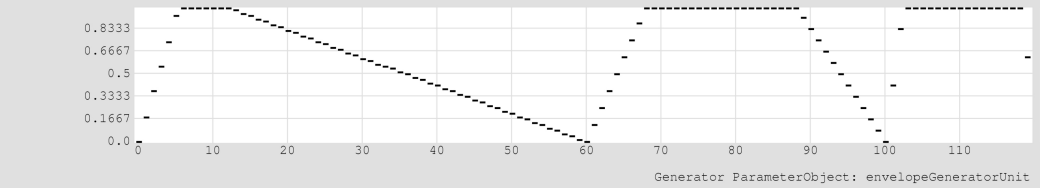
Sample Arguments: `egu, 1, 100, (c,40), (c,0.4), (c,0.2), 0, 1`

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)

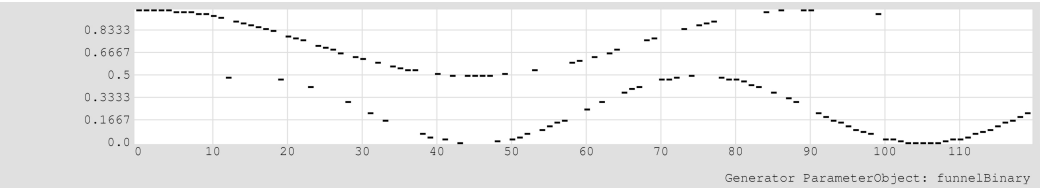
funnelBinary, thresholdMatchString, parameterObject, parameterObject, parameterObject, parameterObject

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}

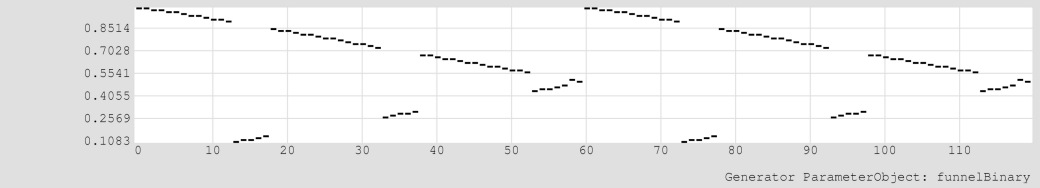
Sample Arguments: `fb, u, (bpl,e,s,((0,0),(120,1))), (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), (ru,0,1)`

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)

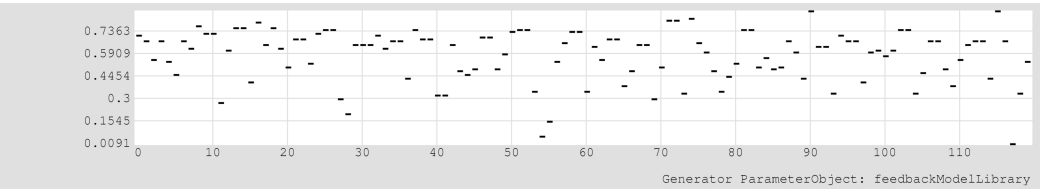
feedbackModelLibrary, feedbackModelName, parameterObject, parameterObject, min, max

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

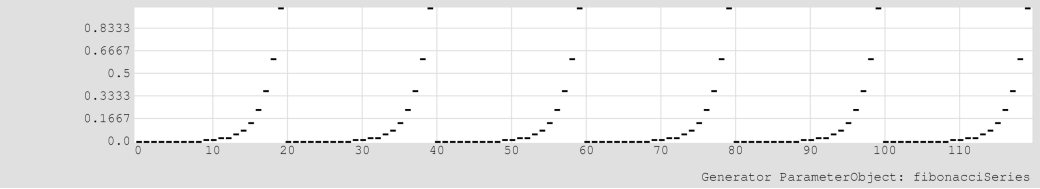
Description: Provides values derived from a contigous section of the Fibonacci series. A section is built from an initial value (start) and as many additional values as specified by the length argument. Negative length values reverse the direction of the series. The resulting list of values is normalized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

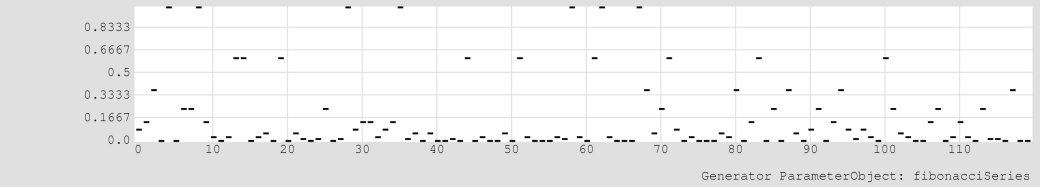
Sample Arguments: `fs, 200, 20, 0, 1, oc`

Example C-44. fibonacciSeries Demonstration 1



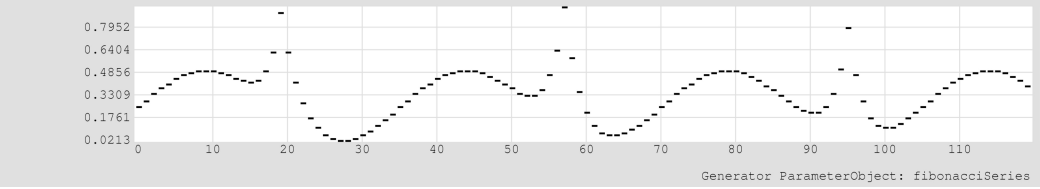

```
fibonacciSeries, 200, 20, (constant, 0), (constant, 1), orderedCyclic
```

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)

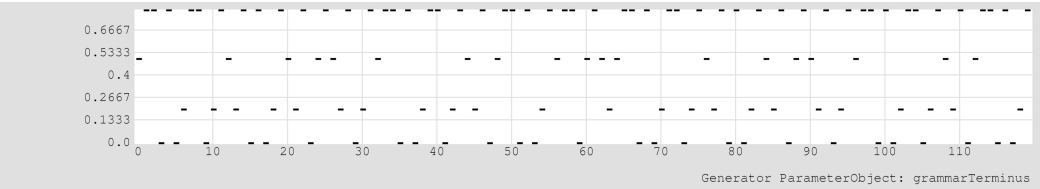
grammarTerminus, grammarString, stepCount, selectionString

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)

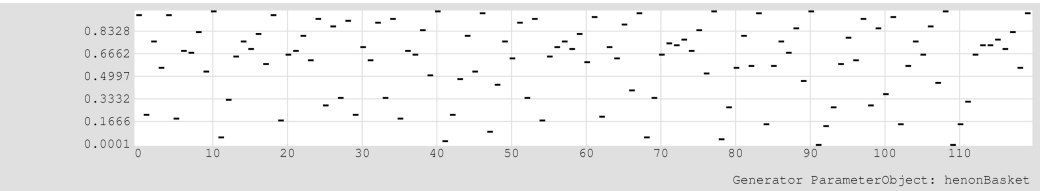
henonBasket, xInit, yInit, parameterObject, parameterObject, valueCount, valueSelect, min, max, selectionString

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'}

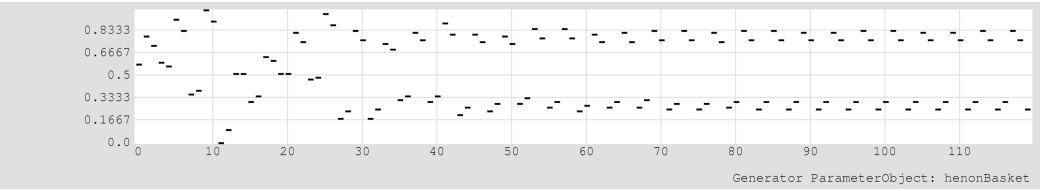
Sample Arguments: `hb, 0.5, 0.5, 1.4, 0.3, 1000, x, 0, 1, oc`

Example C-48. henonBasket Demonstration 1



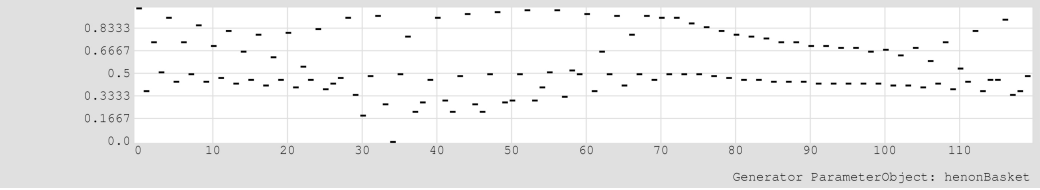

```
henonBasket, 0.5, 0.5, (constant, 1.4), (constant, 0.3), 1000, x, (constant,
0), (constant, 1), orderedCyclic
```

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

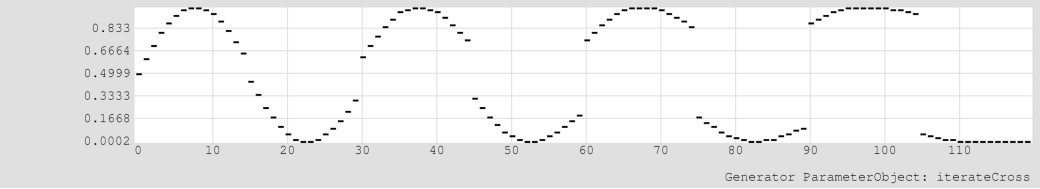
Description: Produces a single value cross faded between two values created by two Generator ParameterObjects in parallel. The cross fade is expressed as a number within the unit interval, where

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}

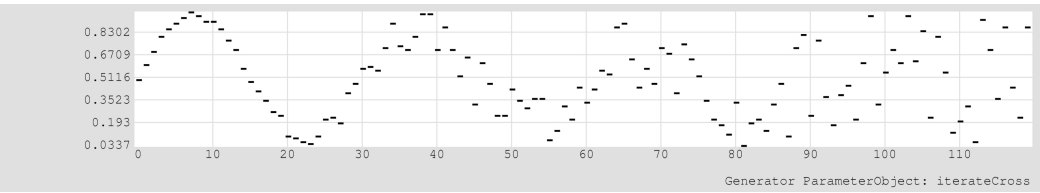
Sample Arguments: `ic, (ws,e,30,0,0,1), (wp,e,30,0,0,1), (bpl,e,l,((0,0),(120,1)))`

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

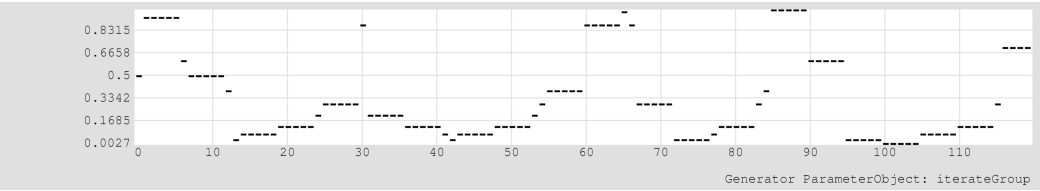
Description: Allows the output of a source ParameterObject to be grouped (a value is held and repeated a certain number of times), to be skipped (a number of values are generated and discarded), or to be bypassed. A numeric value from a control ParameterObject is used to determine the source ParameterObject 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}

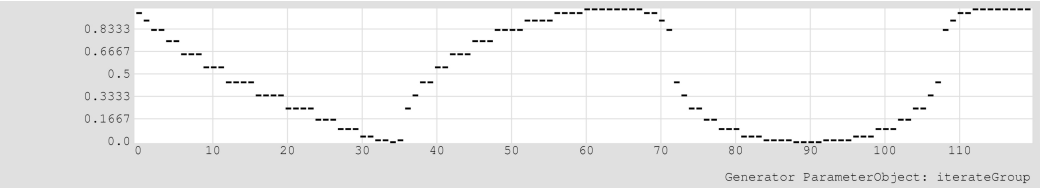
Sample Arguments: `ig, (ws,e,30,0,0,1), (bg,rc,(-3,1,-1,5))`

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)

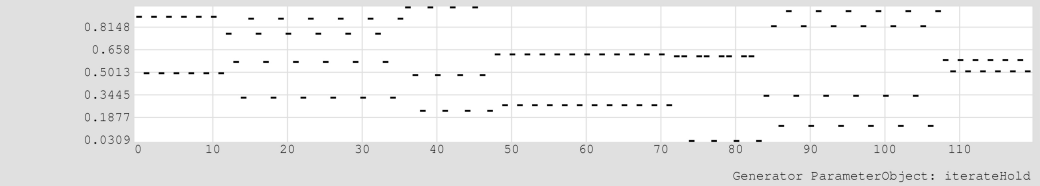
iterateHold, parameterObject, parameterObject, parameterObject, selectionString

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'}

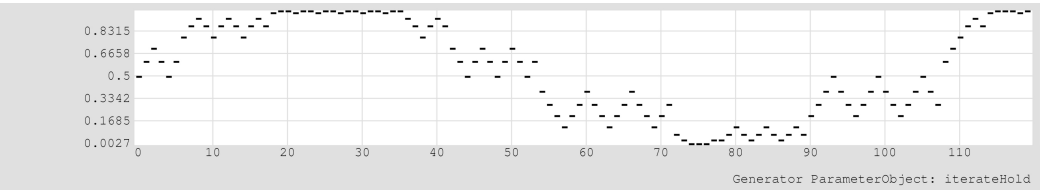
Sample Arguments: `ih, (ru,0,1), (bg,rc,(2,3,4)), (bg,oc,(12,24)), oc`

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)

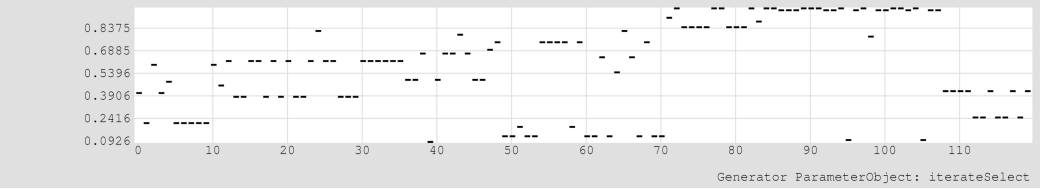
iterateSelect, parameterObject, parameterObject, parameterObject, parameterObject

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}

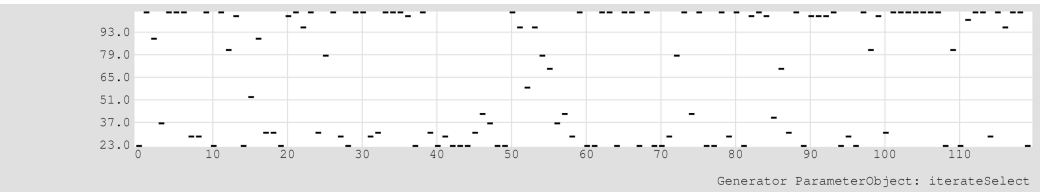
Sample Arguments: `is, (ru,0,1), (bg,rc,(10,11,12)), (bg,oc,(12,24)), (rb,0.15,0.15,0,1)`

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

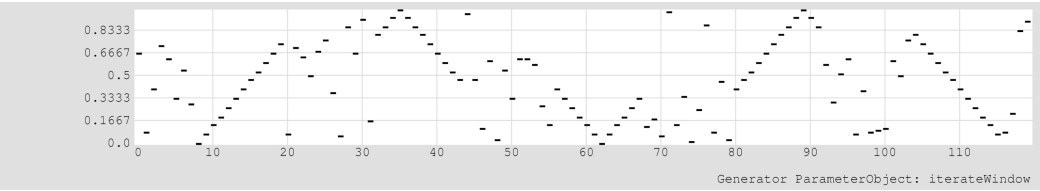
Description: Allows a ParameterObject, selected from a list of ParameterObjects, to generate values, to skip values (a number of values are generated and discarded), or to bypass value generation. A numeric value from a control ParameterObject is used to determine the selected ParameterObject behavior. 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'}

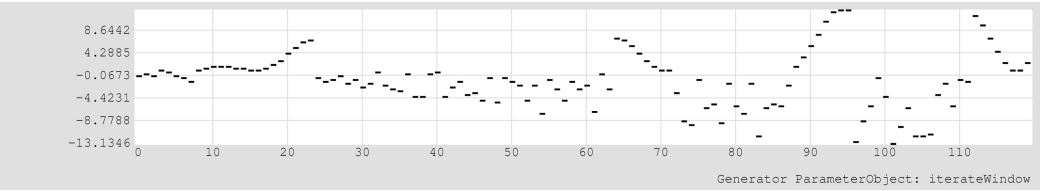
Sample Arguments: `iw, ((ru,0,1),(wt,e,30,0,0,1)), (bg,oc,(8,4,-2)), oc`

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

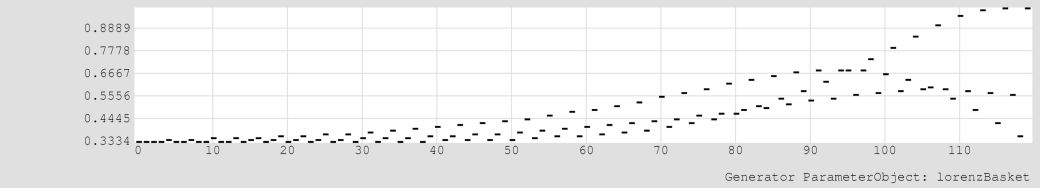
Description: Performs the Lorenz attractor, a non-linear three-dimensional discrete deterministic dynamical system. The equations are derived from a simplified model of atmospheric convection rolls. For some parameter settings the system exhibits chaotic behavior, for others, periodic behavior; small 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'}

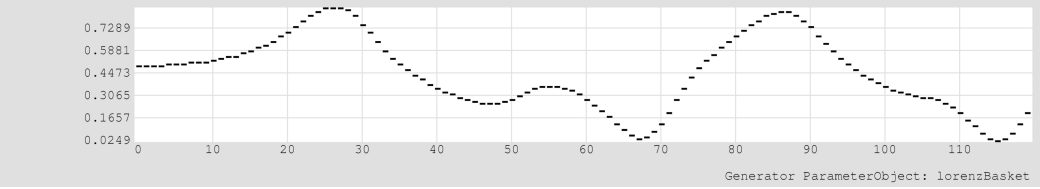
Sample Arguments: `lb, 1.0, 1.0, 1.0, 28, 10, 2.67, 1000, xyz, 0, 1, oc`

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)

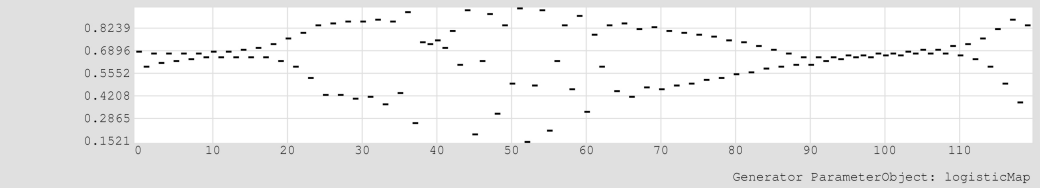
logisticMap, initValue, parameterObject, min, max

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

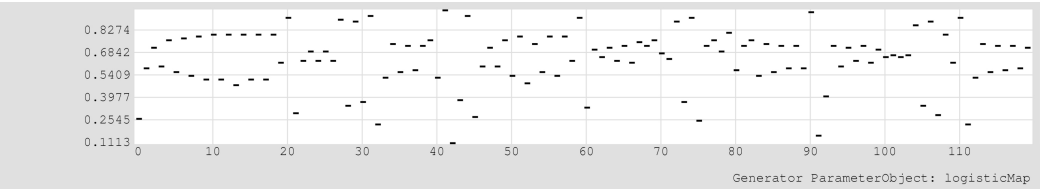
Sample Arguments: `lm, 0.5, (wt,e,90,0,2.75,4), 0, 1`

Example C-63. logisticMap Demonstration 1



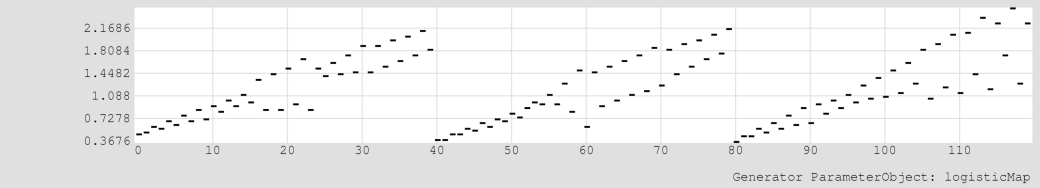

```
logisticMap, 0.5, (waveTriangle, event, (constant, 90), 0, (constant, 2.75),
(constant, 4)), (constant, 0), (constant, 1)
```

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)

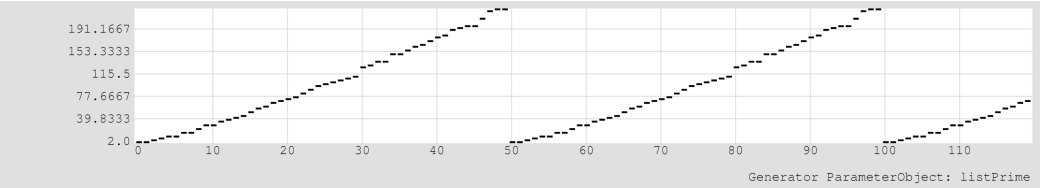
listPrime, start, length, format, selectionString

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'}

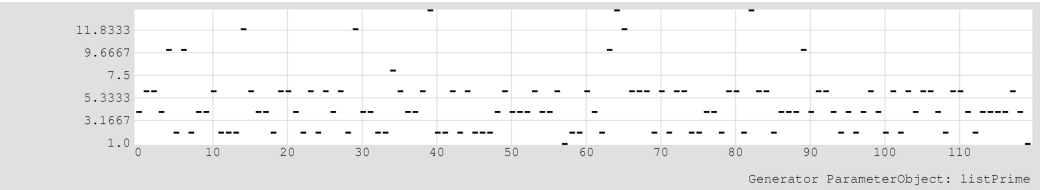
Sample Arguments: `lp, 2, 50, int, oc`

Example C-66. listPrime Demonstration 1



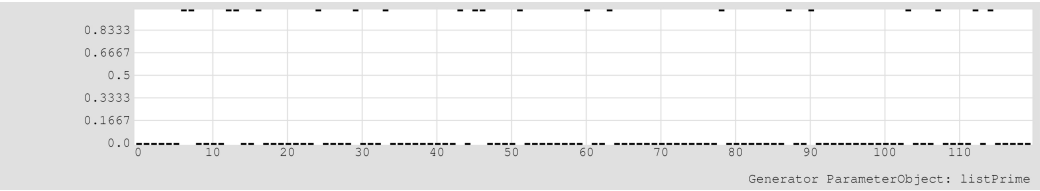

```
listPrime, 2, 50, integer, orderedCyclic
```

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)

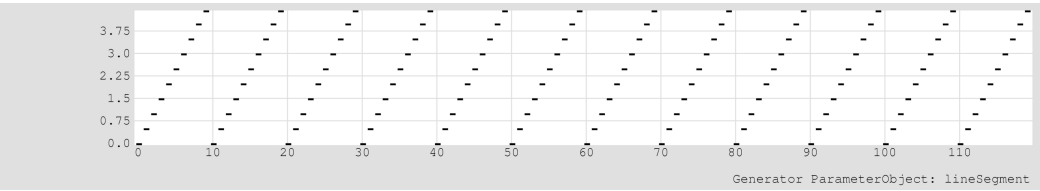
lineSegment, stepString, parameterObject, min, max

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

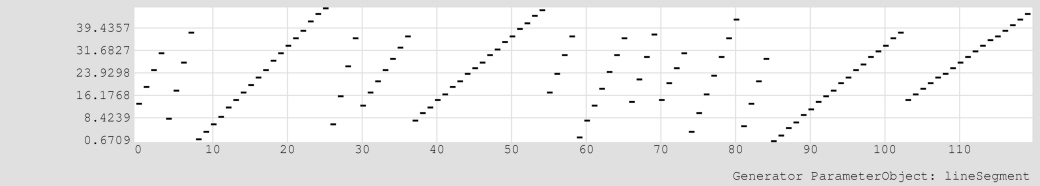
Sample Arguments: `ls, e, 10, 0, 5`

Example C-69. lineSegment Demonstration 1



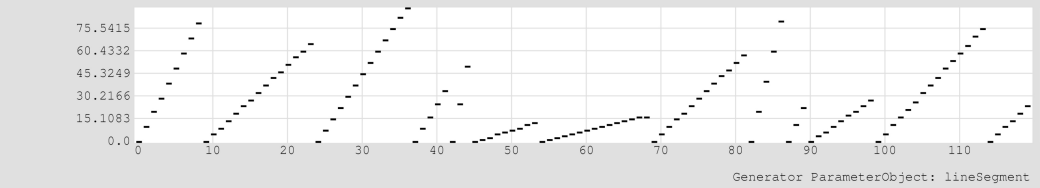
```
lineSegment, (constant, 10), (constant, 0), (constant, 5)
```

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)

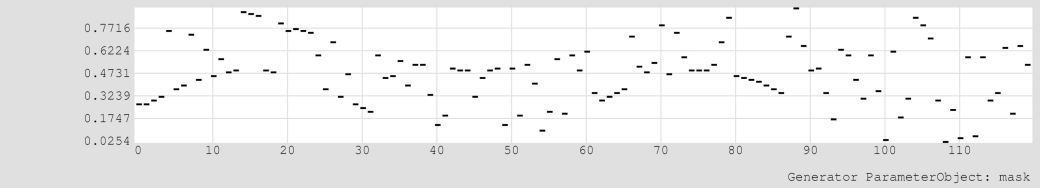
mask, boundaryString, parameterObject, parameterObject, parameterObject

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}

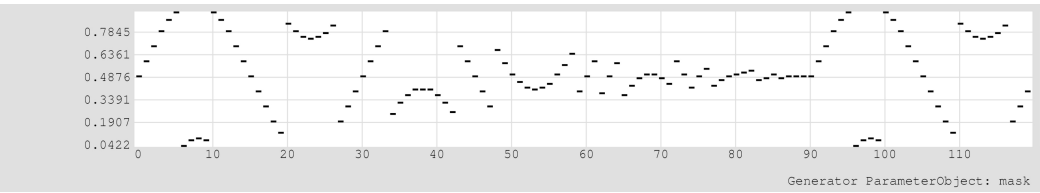
Sample Arguments: `m, l, (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), (ru,0,1)`

Example C-72. mask Demonstration 1



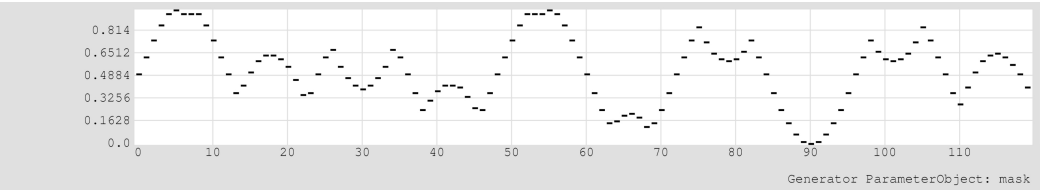

```
mask, limit, (waveSine, event, (constant, 60), 0, (constant, 0.5), (constant,
0)), (waveCosine, event, (constant, 90), 0, (constant, 0.5), (constant, 1)),
(randomUniform, (constant, 0), (constant, 1))
```

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

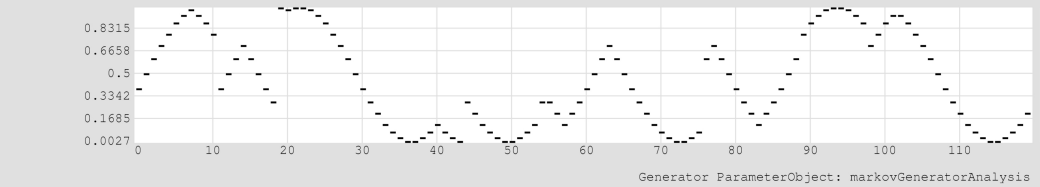
Description: Produces values by means of a Markov analysis of values provided by a source Generator ParameterObject; the analysis of these values is used with a dynamic transition order

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}

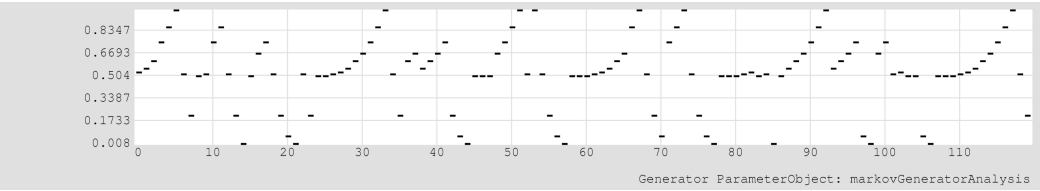
Sample Arguments: `mga, (ws,e,30,0,0,1), 30, 2, (mv,a{1}b{0}c{2}:{a=10|b=1|c=2},(c,0))`

Example C-75. markovGeneratorAnalysis Demonstration 1



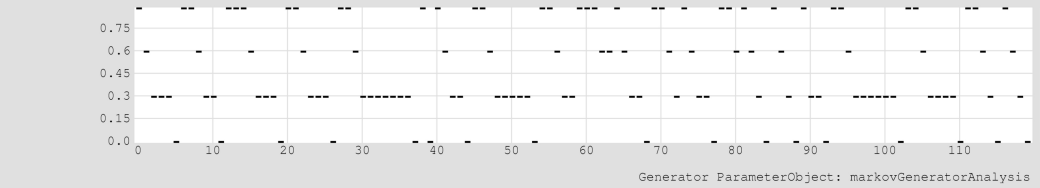

```
markovGeneratorAnalysis, (waveSine, event, (constant, 30), 0, (constant, 0),
(constant, 1)), 30, 2, (markovValue, a{1}b{0}c{2}:{a=10|b=1|c=2}, (constant,
0))
```

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)

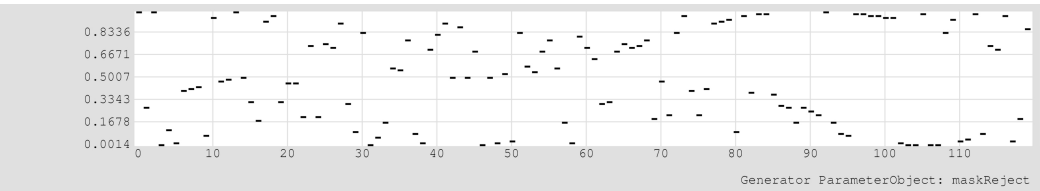
maskReject, boundaryString, parameterObject, parameterObject, parameterObject

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}

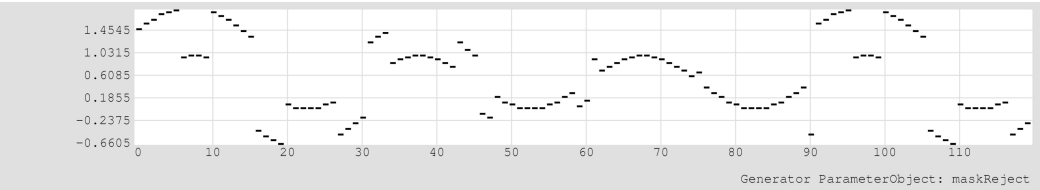
Sample Arguments: `mr, 1, (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), (ru,0,1)`

Example C-78. maskReject Demonstration 1



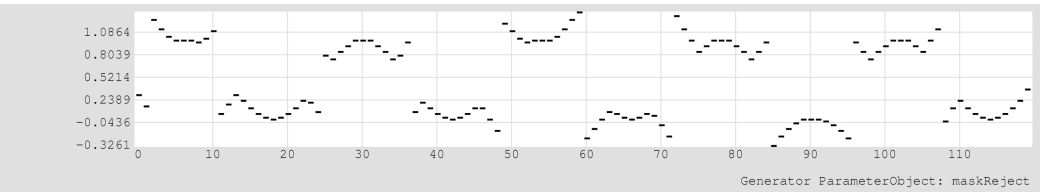

```
maskReject, limit, (waveSine, event, (constant, 60), 0, (constant, 0.5),
(constant, 0)), (waveCosine, event, (constant, 90), 0, (constant, 0.5),
(constant, 1)), (randomUniform, (constant, 0), (constant, 1))
```

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)

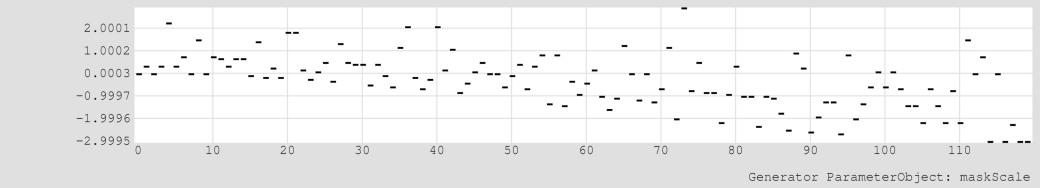
maskScale, parameterObject, valueCount, min, max, selectionString

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)

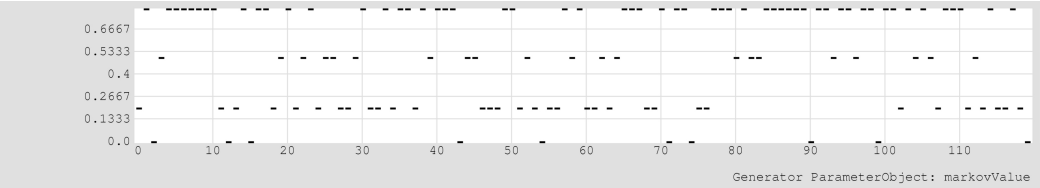
markovValue, transitionString, parameterObject

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}

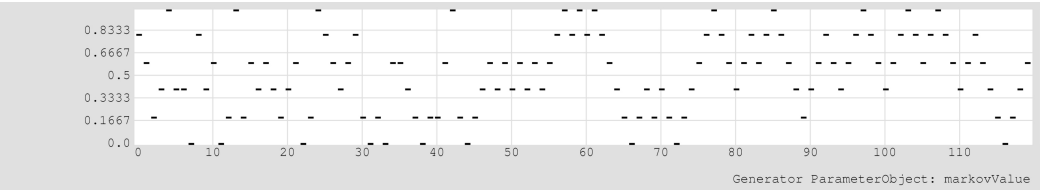
Sample Arguments: `mv, a{.2}b{.5}c{.8}d{0}:{a=5|b=4|c=7|d=1}, (c,0)`

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)

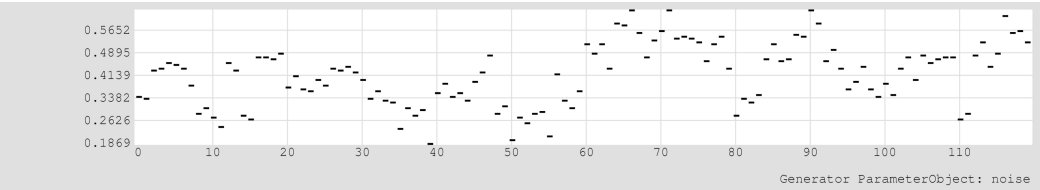
noise, resolution, parameterObject, min, max

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

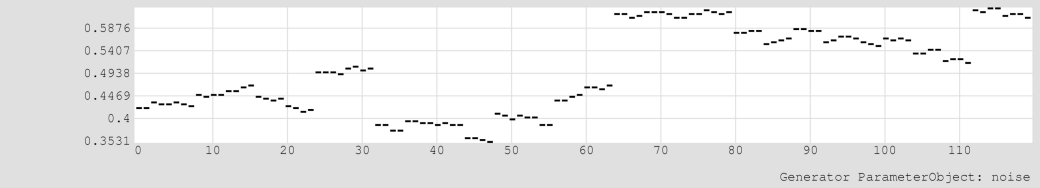
Sample Arguments: `n, 100, pink, 0, 1`

Example C-84. noise Demonstration 1



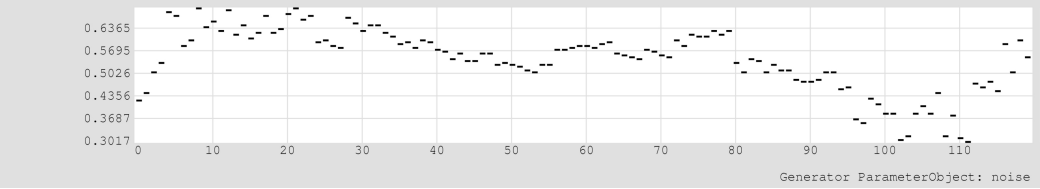

```
noise, 100, (constant, 1), (constant, 0), (constant, 1)
```

Example C-85. noise Demonstration 2



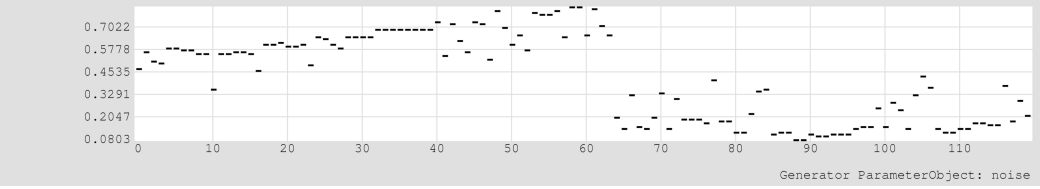

```
noise, 100, (constant, 3), (constant, 0), (constant, 1)
```

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)

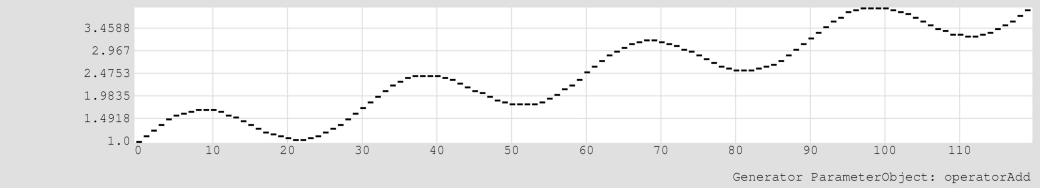
operatorAdd, parameterObject, parameterObject

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)

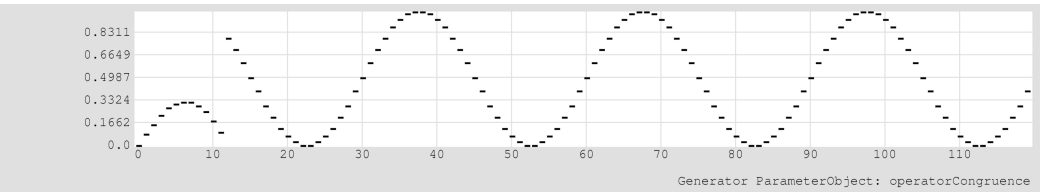
operatorCongruence, parameterObject, parameterObject

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)

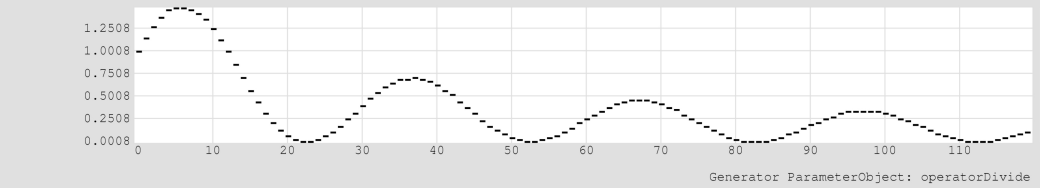
operatorDivide, parameterObject, parameterObject

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)

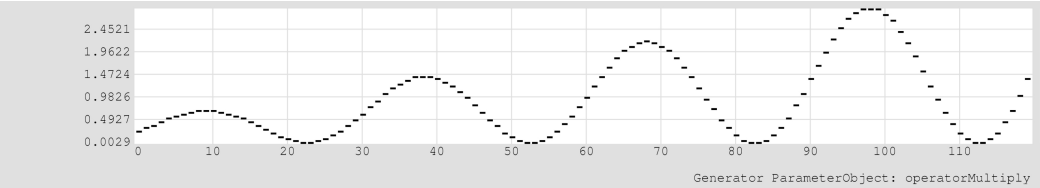
operatorMultiply, parameterObject, parameterObject

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)

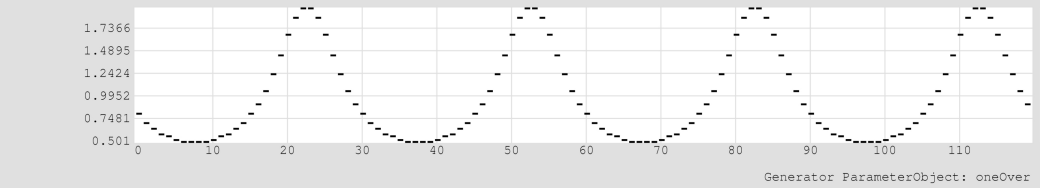
oneOver, parameterObject

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)

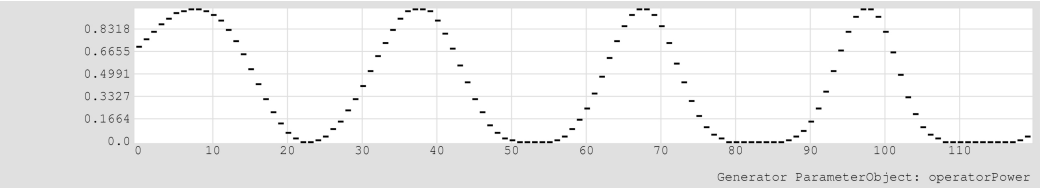
operatorPower, parameterObject, parameterObject

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)

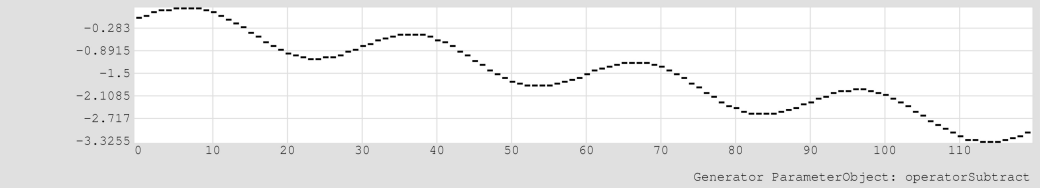
operatorSubtract, parameterObject, parameterObject

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)

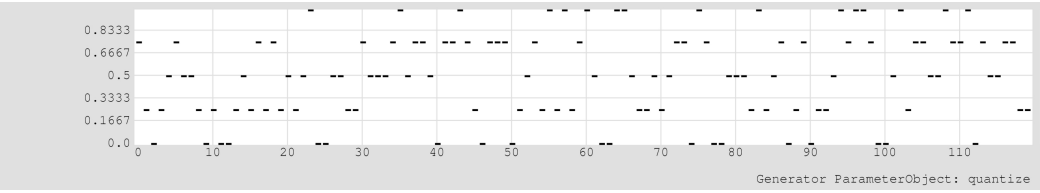
quantize, parameterObject, parameterObject, stepCount, parameterObject, parameterObject

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}

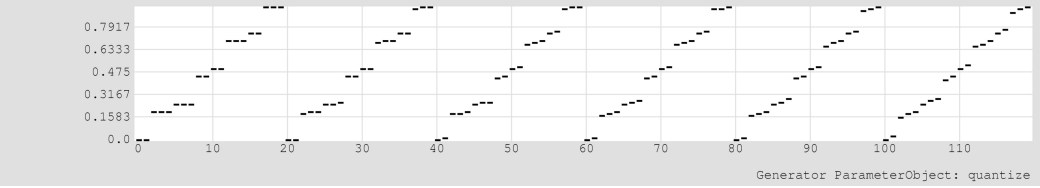
Sample Arguments: `q, (c,0), (c,0.25), 1, (c,1), (ru,0,1)`

Example C-95. quantize Demonstration 1



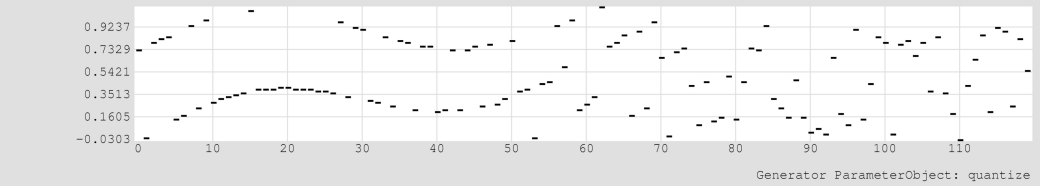

```
quantize, (constant, 0), (constant, 0.25), 1, (constant, 1), (randomUniform,
(constant, 0), (constant, 1))
```

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

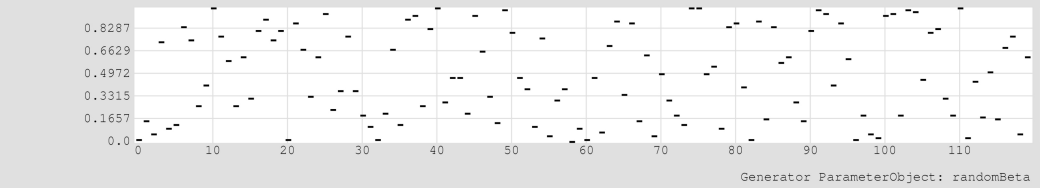
Description: Provides random numbers between 0 and 1 within a Beta distribution. This value is scaled within the range designated by min and max; min and max may be specified with

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

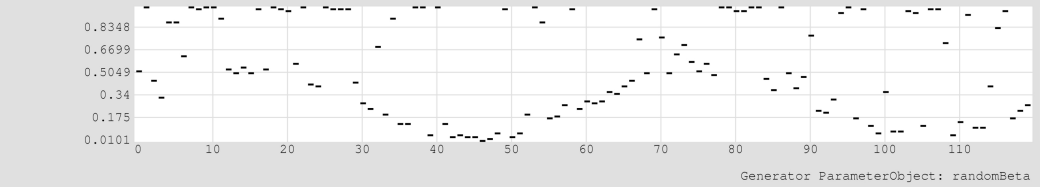
Sample Arguments: `rb, 0.5, 0.5, 0, 1`

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)

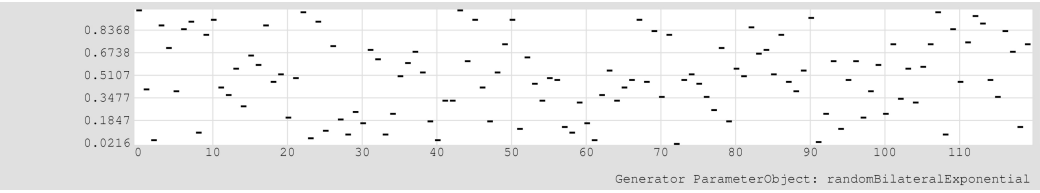
randomBilateralExponential, lambda, min, max

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

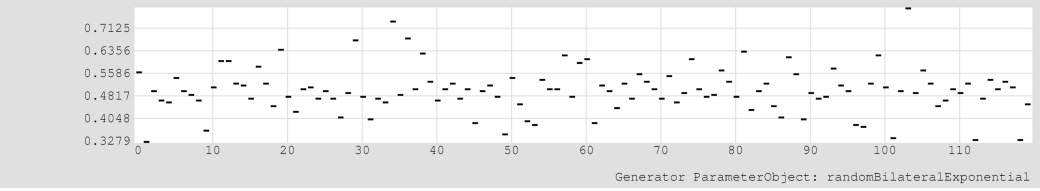
Sample Arguments: `rbe, 0.5, 0, 1`

Example C-100. randomBilateralExponential Demonstration 1



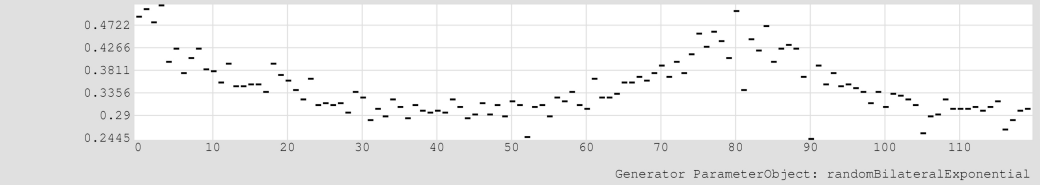

```
randomBilateralExponential, 0.5, (constant, 0), (constant, 1)
```

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)

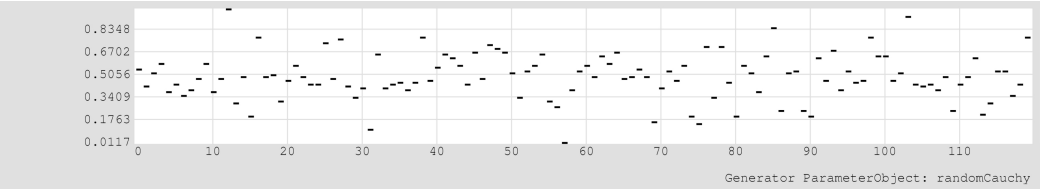
randomCauchy, alpha, mu, min, max

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

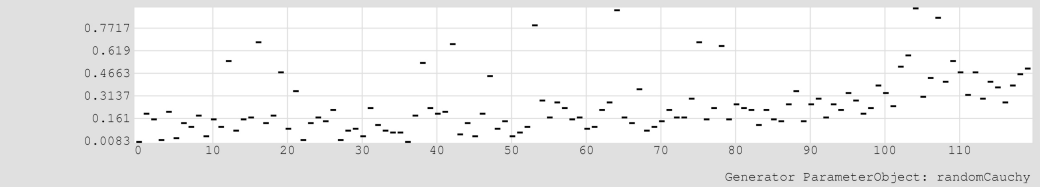
Sample Arguments: `rc, 0.1, 0.5, 0, 1`

Example C-103. randomCauchy Demonstration 1



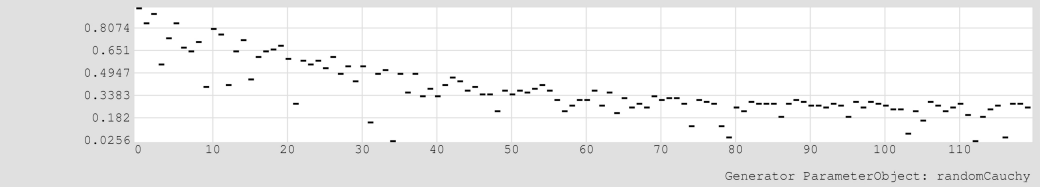

```
randomCauchy, 0.1, 0.5, (constant, 0), (constant, 1)
```

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

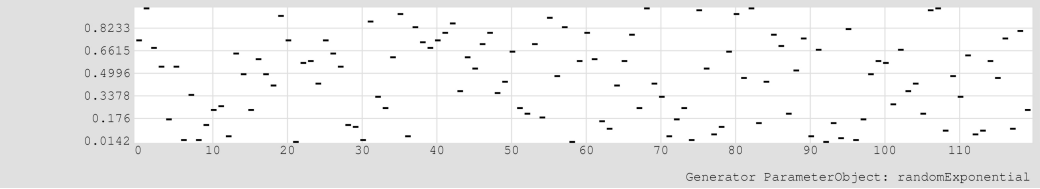
Description: Provides random numbers between 0 and 1 within an exponential distribution. This value is scaled within the range designated by min and max; min and max may be specified with

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

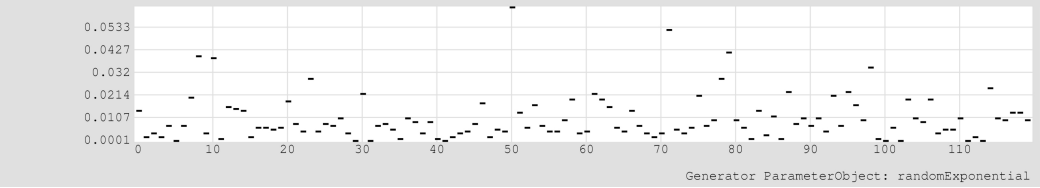
Sample Arguments: `re, 0.5, 0, 1`

Example C-106. randomExponential Demonstration 1



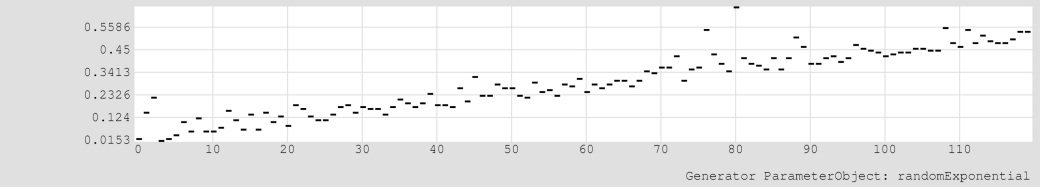

```
randomExponential, 0.5, (constant, 0), (constant, 1)
```

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)))
```

C.1.56. randomGauss (rg)

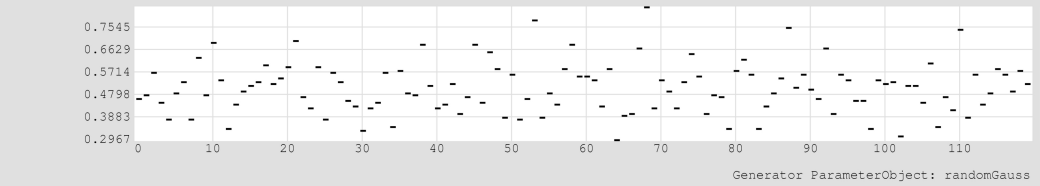
randomGauss, mu, sigma, min, max

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

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

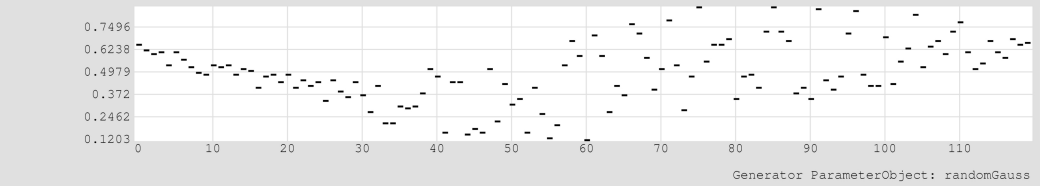
Sample Arguments: `rg, 0.5, 0.1, 0, 1`

Example C-109. randomGauss Demonstration 1




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

Example C-110. randomGauss Demonstration 2




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

C.1.57. randomInverseExponential (rie)

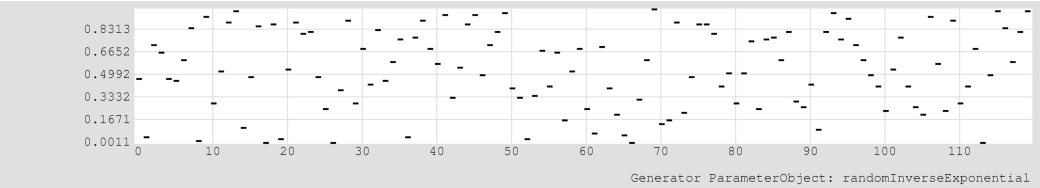
randomInverseExponential, lambda, min, max

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

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

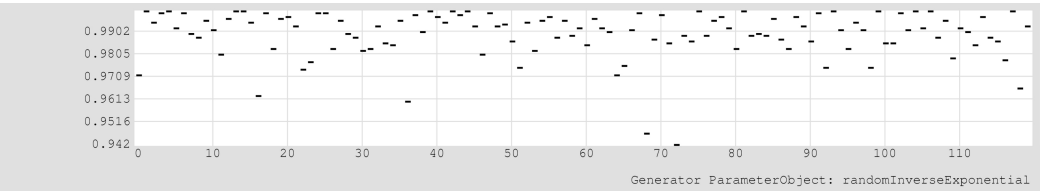
Sample Arguments: `rie, 0.5, 0, 1`

Example C-111. randomInverseExponential Demonstration 1



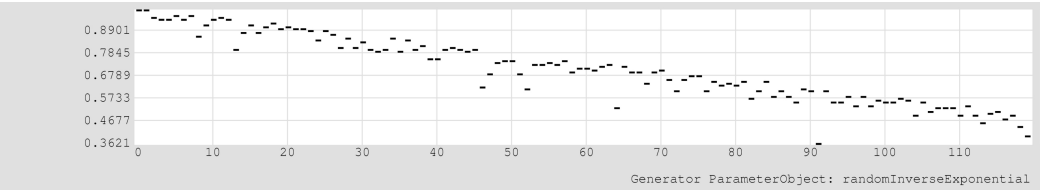

```
randomInverseExponential, 0.5, (constant, 0), (constant, 1)
```

Example C-112. randomInverseExponential Demonstration 2




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

Example C-113. randomInverseExponential Demonstration 3




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

C.1.58. randomInverseLinear (ril)

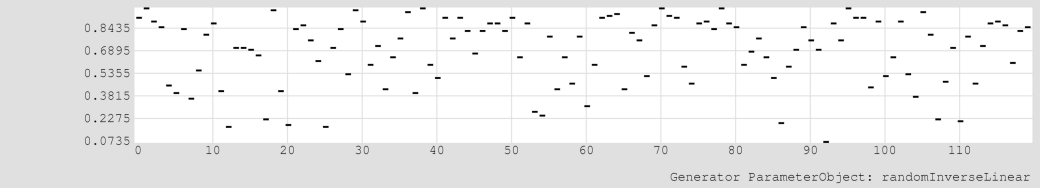
randomInverseLinear, min, max

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

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

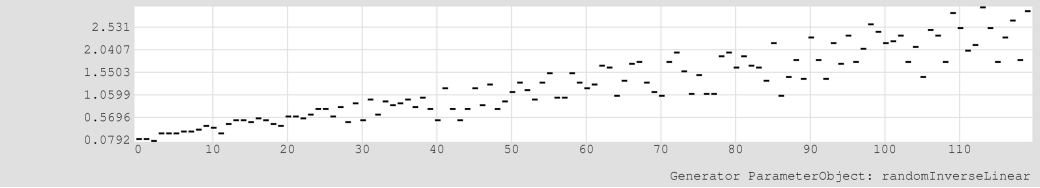
Sample Arguments: `ril, 0, 1`

Example C-114. randomInverseLinear Demonstration 1




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

Example C-115. randomInverseLinear Demonstration 2




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

C.1.59. randomInverseTriangular (rit)

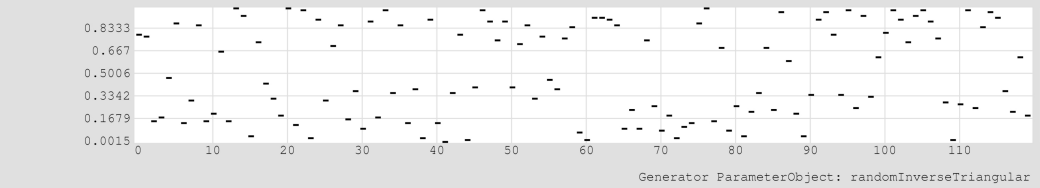
randomInverseTriangular, min, max

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

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

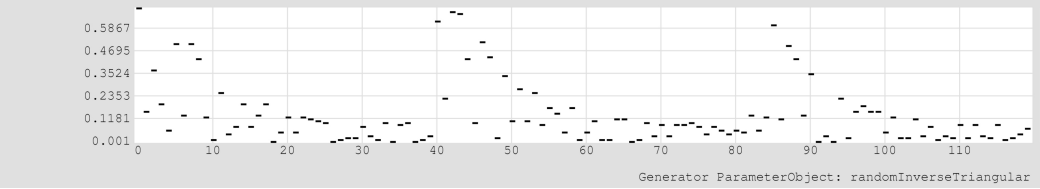
Sample Arguments: `rit, 0, 1`

Example C-116. randomInverseTriangular Demonstration 1




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

Example C-117. randomInverseTriangular Demonstration 2




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

C.1.60. randomLinear (rl)

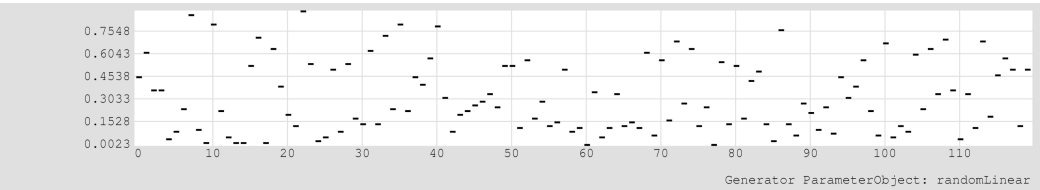
randomLinear, min, max

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

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

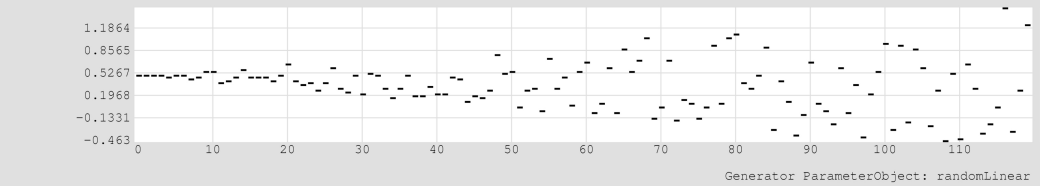
Sample Arguments: `rl, 0, 1`

Example C-118. randomLinear Demonstration 1



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

Example C-119. randomLinear Demonstration 2




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

C.1.61. randomTriangular (rt)

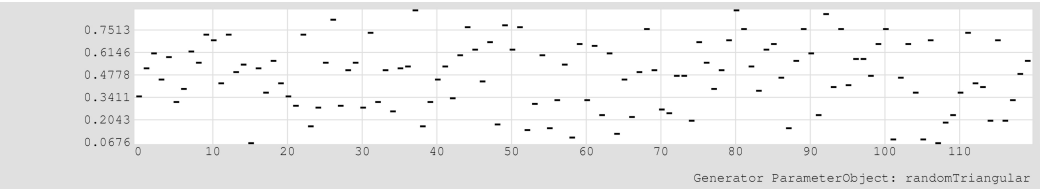
randomTriangular, min, max

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

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

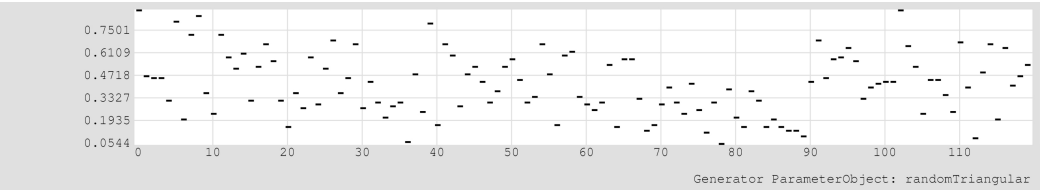
Sample Arguments: `rt, 0, 1`

Example C-120. randomTriangular Demonstration 1




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

Example C-121. randomTriangular Demonstration 2




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

C.1.62. randomUniform (ru)

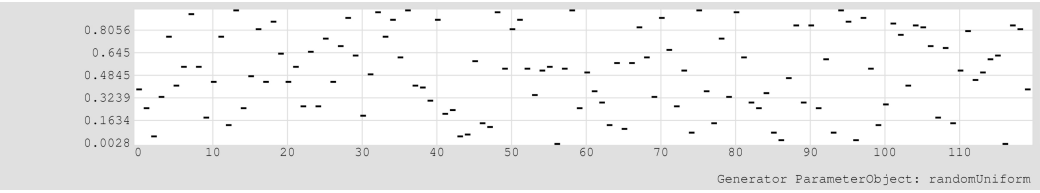
randomUniform, min, max

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

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

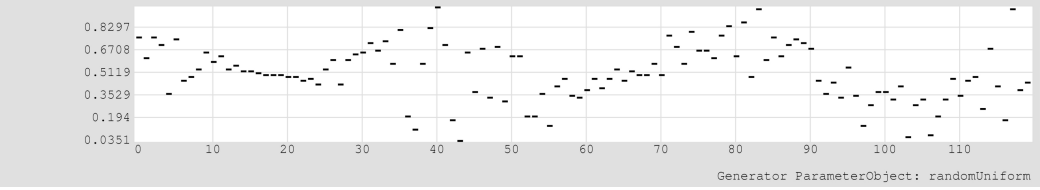
Sample Arguments: `ru, 0, 1`

Example C-122. randomUniform Demonstration 1




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

Example C-123. randomUniform Demonstration 2



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

C.1.63. randomWeibull (rw)

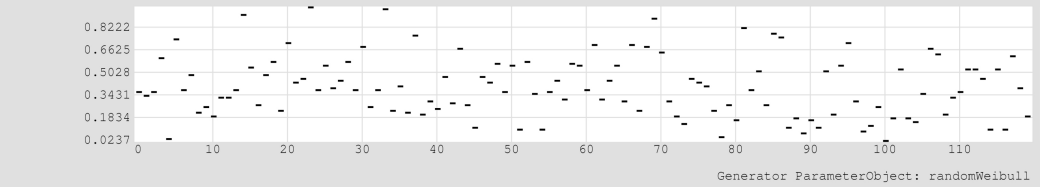
randomWeibull, alpha, beta, min, max

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

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

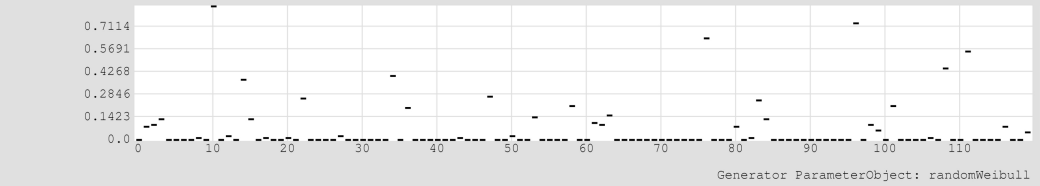
Sample Arguments: `rw, 0.5, 2.0, 0, 1`

Example C-124. randomWeibull Demonstration 1



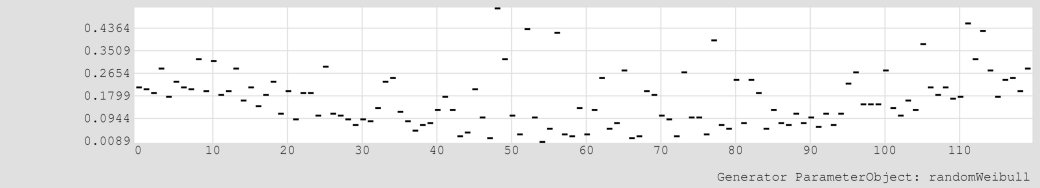

```
randomWeibull, 0.5, 2.0, (constant, 0), (constant, 1)
```

Example C-125. randomWeibull Demonstration 2




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

Example C-126. randomWeibull Demonstration 3




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

C.1.64. sampleAndHold (sah)

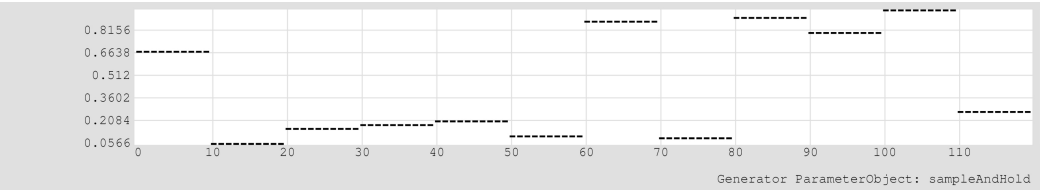
sampleAndHold, comparison, parameterObject, parameterObject, parameterObject

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

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

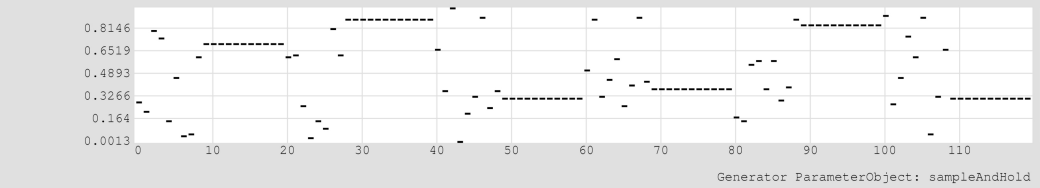
Sample Arguments: `sah, gt, (ru,0,1), (wsd,e,10,0,0,1), (c,0.5)`

Example C-127. sampleAndHold Demonstration 1



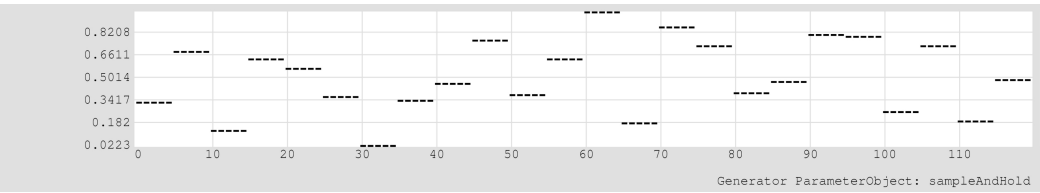

```
sampleAndHold, greaterThan, (randomUniform, (constant, 0), (constant, 1)),
(waveSawDown, event, (constant, 10), 0, (constant, 0), (constant, 1)),
(constant, 0.5)
```

Example C-128. sampleAndHold Demonstration 2




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

Example C-129. sampleAndHold Demonstration 3




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

C.1.65. sieveFunnel (sf)

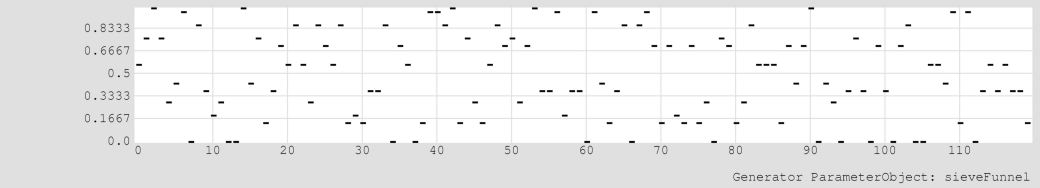
sieveFunnel, logicalString, length, min, max, parameterObject

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

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

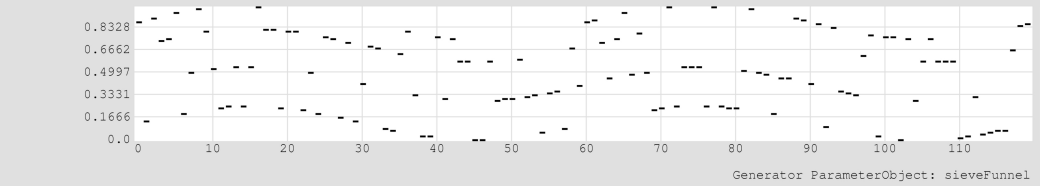
Sample Arguments: `sf, 3|4, 24, 0, 1, (ru,0,1)`

Example C-130. sieveFunnel Demonstration 1



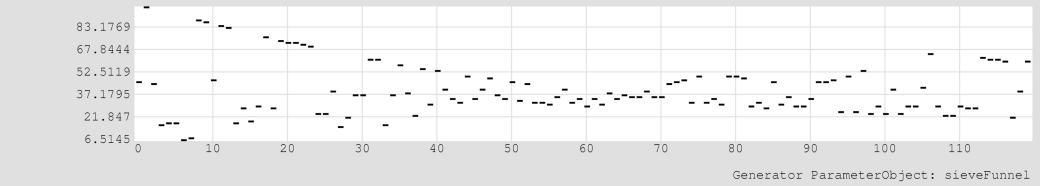

```
sieveFunnel, 3@0|4@0, 24, (constant, 0), (constant, 1), (randomUniform,
(constant, 0), (constant, 1))
```

Example C-131. sieveFunnel Demonstration 2




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

Example C-132. sieveFunnel Demonstration 3




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

C.1.66. sieveList (sl)

sieveList, logicalString, zMin, zMax, format, selectionString

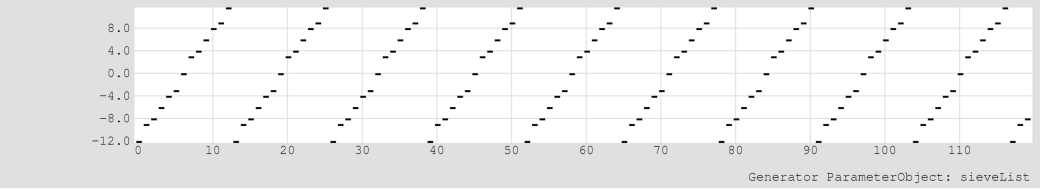
Description: Produces a Xenakis sieve as a raw, variable format sieve segment list. A z is defined by the range of integers from zMin to zMax. Depending on format type, the resulting segment can be

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

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

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

Example C-133. sieveList Demonstration 1




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

C.1.67. sampleSelect (ss)

sampleSelect, fileNameList, selectionString

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

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

Sample Arguments: `ss, (), rc`

C.1.68. typeFormat (tf)

typeFormat, typeFormatString, parameterObject

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

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

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

C.1.69. valuePrime (vp)

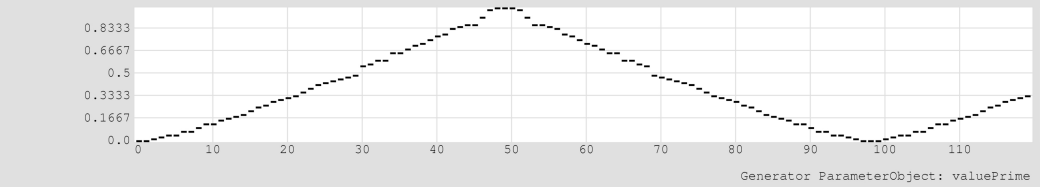
valuePrime, start, length, min, max, selectionString

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

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

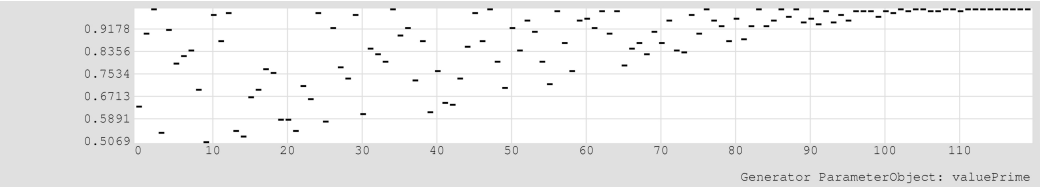
Sample Arguments: `vp, 2, 50, 0, 1, oo`

Example C-134. valuePrime Demonstration 1




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

Example C-135. valuePrime Demonstration 2




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

C.1.70. valueSieve (vs)

valueSieve, logicalString, length, min, max, selectionString

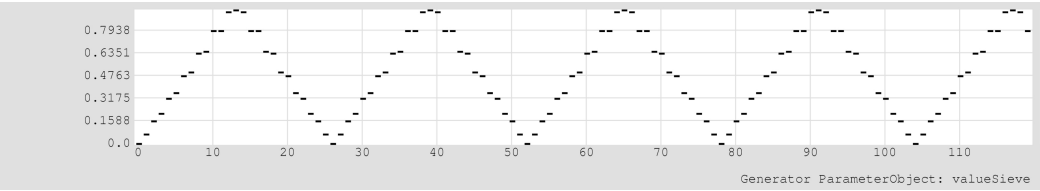
Description: Using the user-supplied logical string, this Generator produces a Xenakis sieve segment within the z range of zero to one less than the supplied length. The resulting list of values is normalized within the unit interval. Values are chosen from this list using the selector specified by

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

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

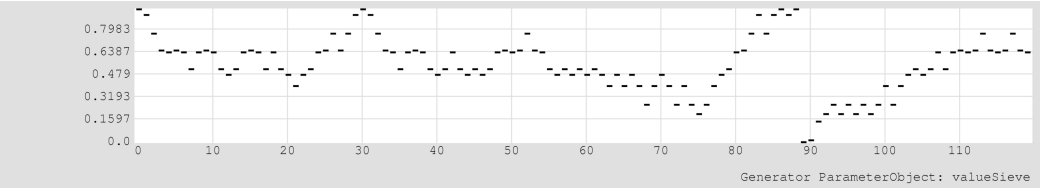
Sample Arguments: `vs, 3&19|4&13@11, 360, 0, 1, oo`

Example C-136. valueSieve Demonstration 1



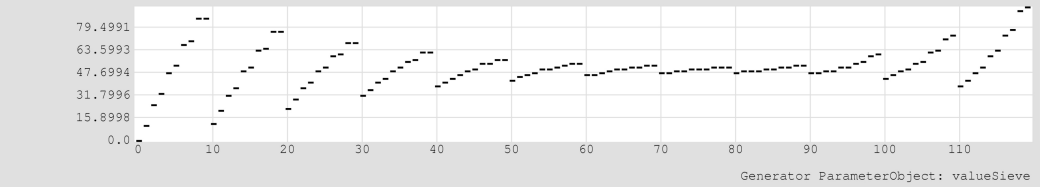

```
valueSieve, 3@0&19@0|4@0&13@11, 360, (constant, 0), (constant, 1),
orderedOscillate
```

Example C-137. valueSieve Demonstration 2



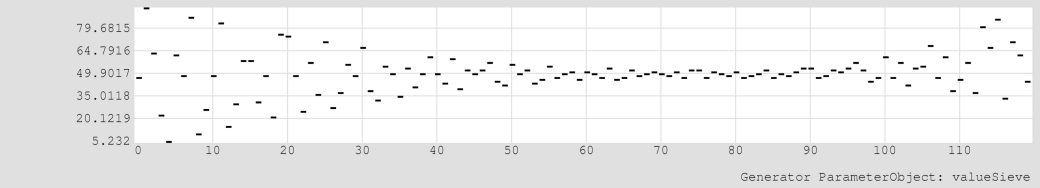

```
valueSieve, 3@0&19@0|4@0&13@11|5@2&15@2, 120, (constant, 0), (constant, 1),
randomWalk
```

Example C-138. valueSieve Demonstration 3




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

Example C-139. valueSieve Demonstration 4




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

C.1.71. waveCosine (wc)

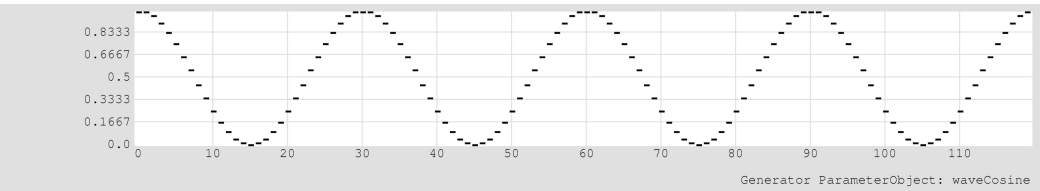
waveCosine, stepString, parameterObject, phase, min, max

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

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

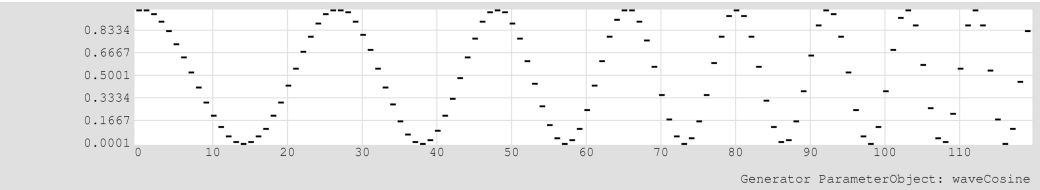
Sample Arguments: `wc, e, 30, 0, 0, 1`

Example C-140. waveCosine Demonstration 1



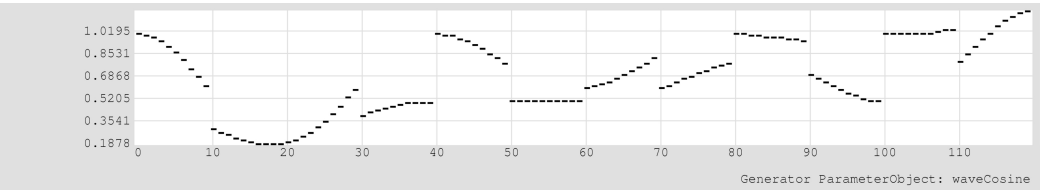

```
waveCosine, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-141. waveCosine Demonstration 2




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

Example C-142. waveCosine Demonstration 3




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

C.1.72. waveHalfPeriodCosine (whpc)

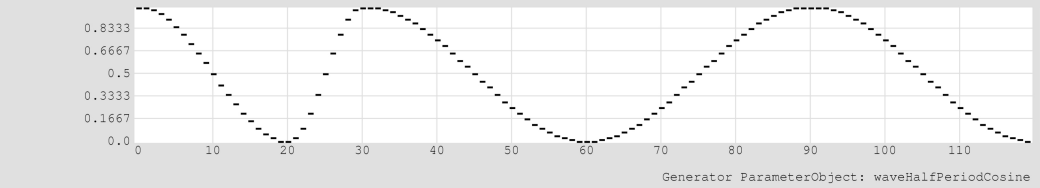
waveHalfPeriodCosine, stepString, parameterObject, phase, min, max

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

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

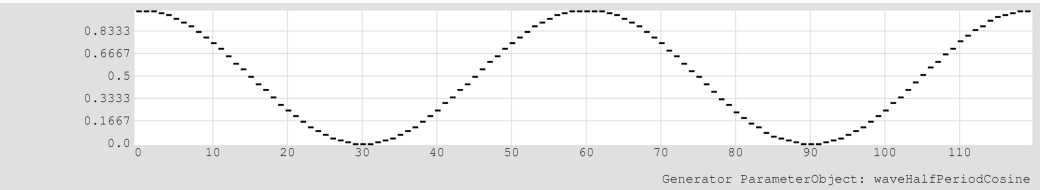
Sample Arguments: `whpc, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-143. waveHalfPeriodCosine Demonstration 1




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

Example C-144. waveHalfPeriodCosine Demonstration 2




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

C.1.73. waveHalfPeriodPulse (whpp)

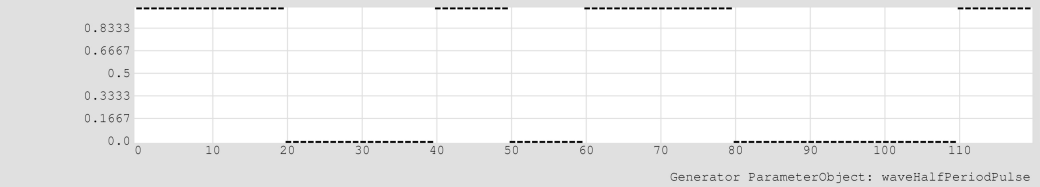
waveHalfPeriodPulse, stepString, parameterObject, phase, min, max

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

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

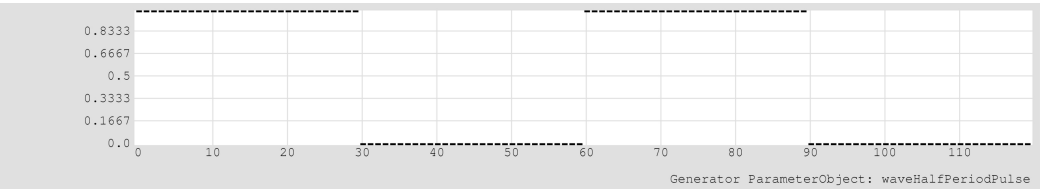
Sample Arguments: `whpp, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-145. waveHalfPeriodPulse Demonstration 1



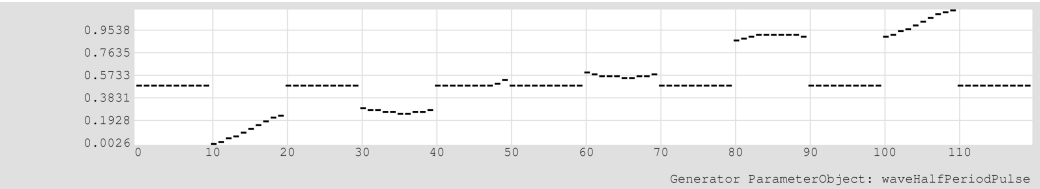

```
waveHalfPeriodPulse, event, (basketGen, randomChoice, (10,20,30)), 0,
(constant, 0), (constant, 1)
```

Example C-146. waveHalfPeriodPulse Demonstration 2



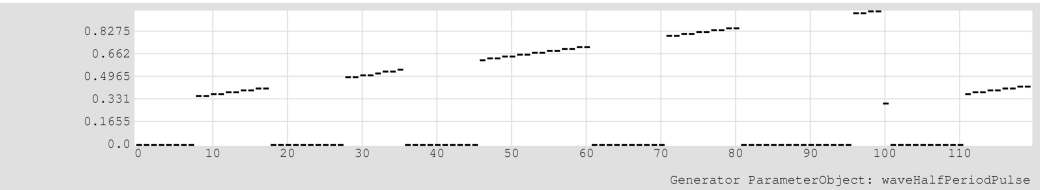

```
waveHalfPeriodPulse, event, (breakPointLinear, event, loop,
((0,30),(120,15))), 0, (constant, 0), (constant, 1)
```

Example C-147. waveHalfPeriodPulse Demonstration 3




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

Example C-148. waveHalfPeriodPulse Demonstration 4



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

C.1.74. waveHalfPeriodSine (whps)

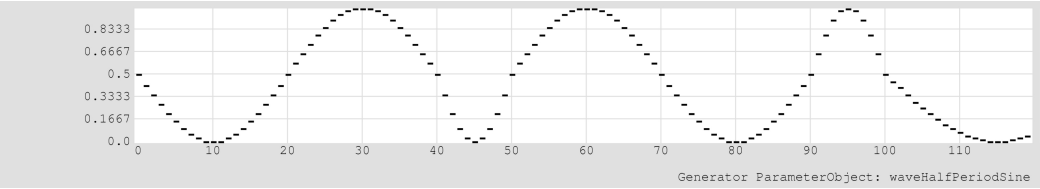
waveHalfPeriodSine, stepString, parameterObject, phase, min, max

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

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

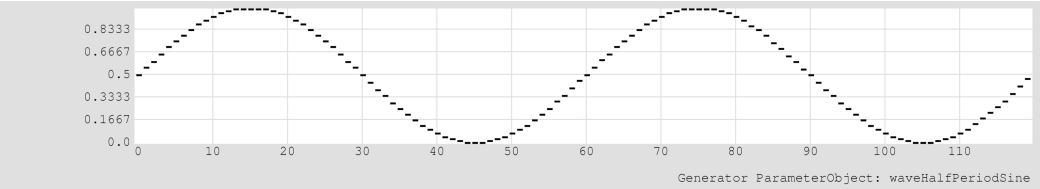
Sample Arguments: `whps, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-149. waveHalfPeriodSine Demonstration 1



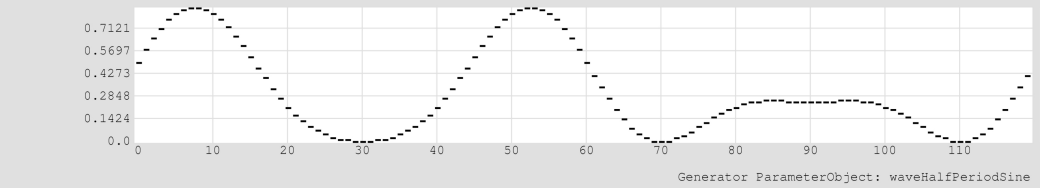

```
waveHalfPeriodSine, event, (basketGen, randomChoice, (10,20,30)), 0,
(constant, 0), (constant, 1)
```

Example C-150. waveHalfPeriodSine Demonstration 2



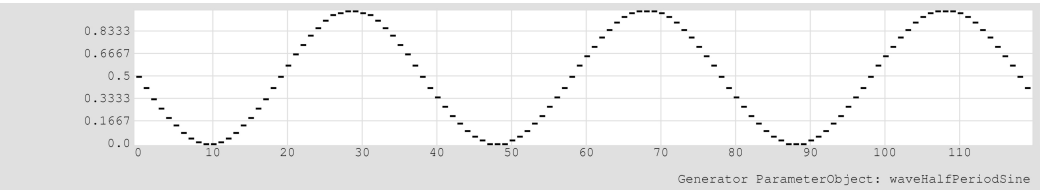

```
waveHalfPeriodSine, event, (breakPointLinear, event, loop, ((0,30),(120,15))),
0, (constant, 0), (constant, 1)
```

Example C-151. waveHalfPeriodSine Demonstration 3




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

Example C-152. waveHalfPeriodSine Demonstration 4




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

C.1.75. waveHalfPeriodTriangle (whpt)

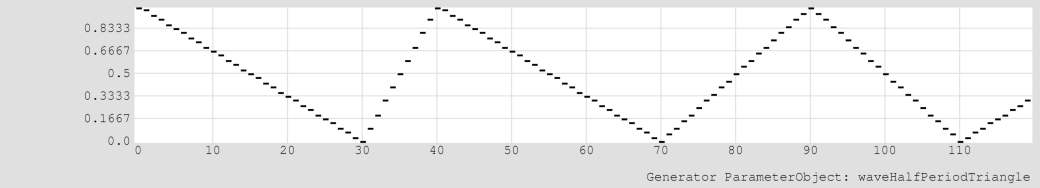
waveHalfPeriodTriangle, stepString, parameterObject, phase, min, max

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

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

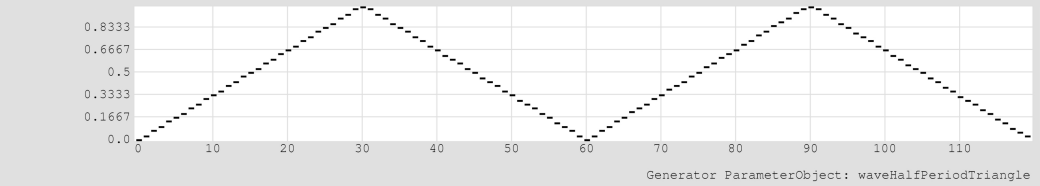
Sample Arguments: `whpt, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-153. waveHalfPeriodTriangle Demonstration 1




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

Example C-154. waveHalfPeriodTriangle Demonstration 2




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

C.1.76. wavePulse (wp)

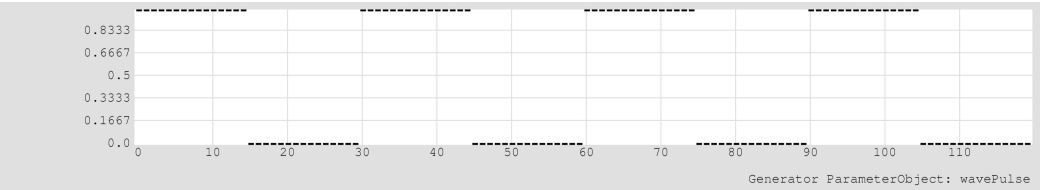
wavePulse, stepString, parameterObject, phase, min, max

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

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

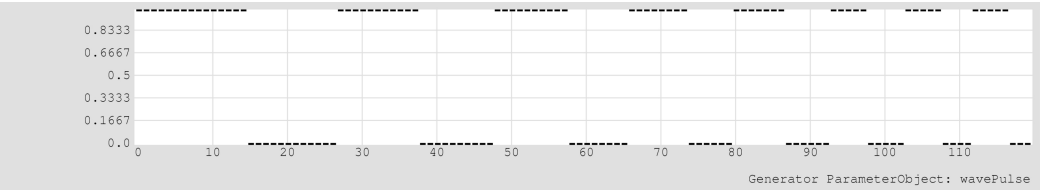
Sample Arguments: `wp, e, 30, 0, 0, 1`

Example C-155. wavePulse Demonstration 1



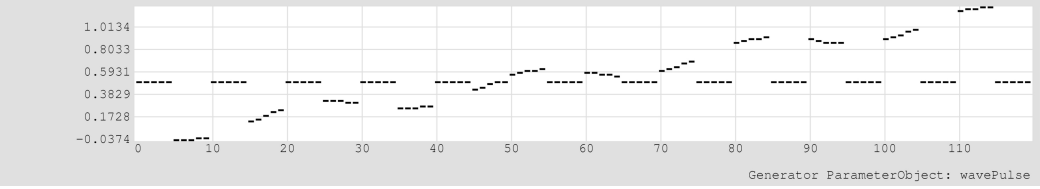

```
wavePulse, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-156. wavePulse Demonstration 2




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

Example C-157. wavePulse Demonstration 3




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

C.1.77. wavePowerDown (wpd)

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

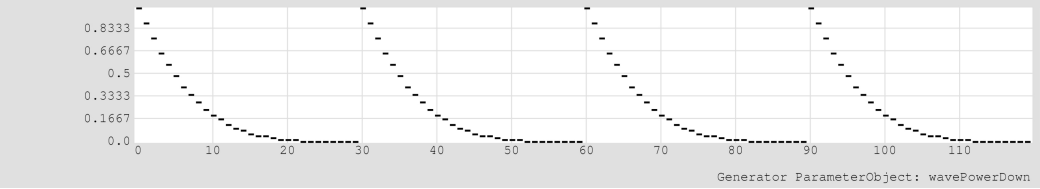
Description: Provides a power down wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min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

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

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

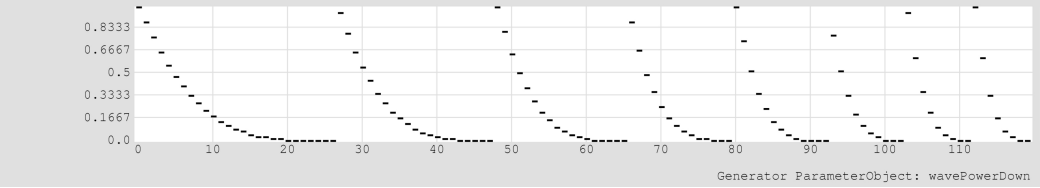
Sample Arguments: `wpd, e, 30, 0, 2, 0, 1`

Example C-158. wavePowerDown Demonstration 1



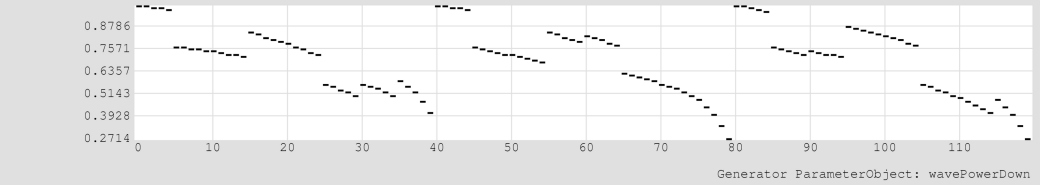

```
wavePowerDown, event, (constant, 30), 0, 2, (constant, 0), (constant, 1)
```

Example C-159. wavePowerDown Demonstration 2




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

Example C-160. wavePowerDown Demonstration 3




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

C.1.78. wavePowerUp (wpu)

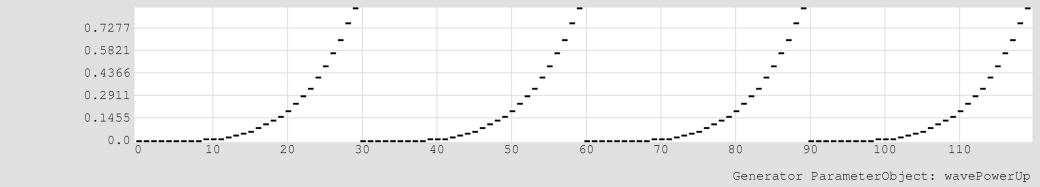
wavePowerUp, stepString, parameterObject, phase, exponent, min, max

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

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

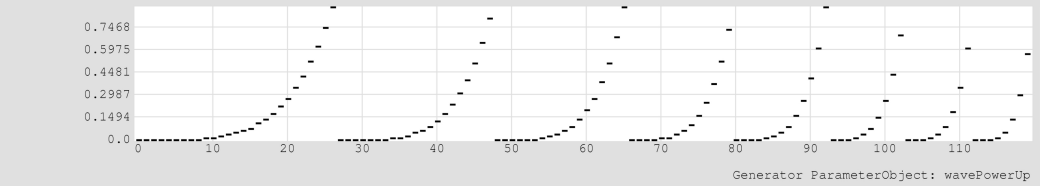
Sample Arguments: `wpu, e, 30, 0, 2, 0, 1`

Example C-161. wavePowerUp Demonstration 1



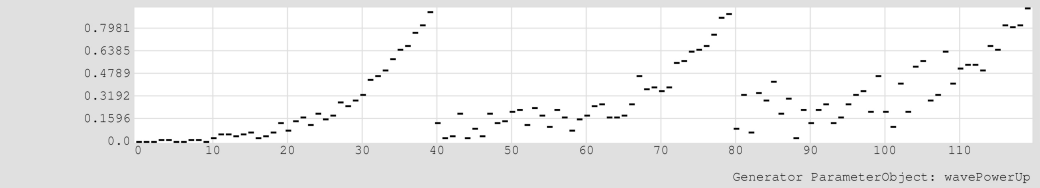

```
wavePowerUp, event, (constant, 30), 0, 2, (constant, 0), (constant, 1)
```

Example C-162. wavePowerUp Demonstration 2




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

Example C-163. wavePowerUp Demonstration 3




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

C.1.79. waveSine (ws)

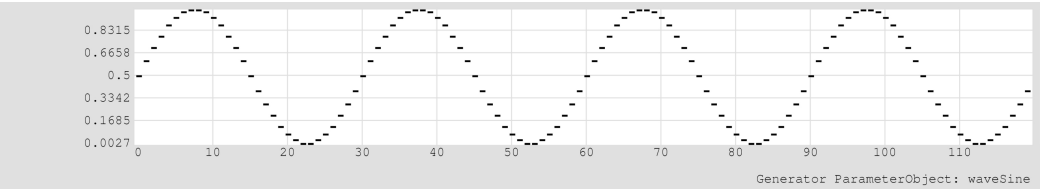
waveSine, stepString, parameterObject, phase, min, max

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

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

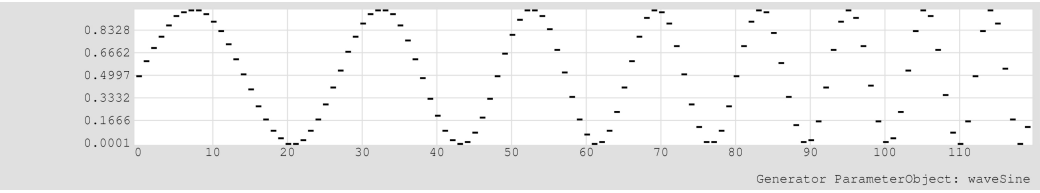
Sample Arguments: `ws, e, 30, 0, 0, 1`

Example C-164. waveSine Demonstration 1



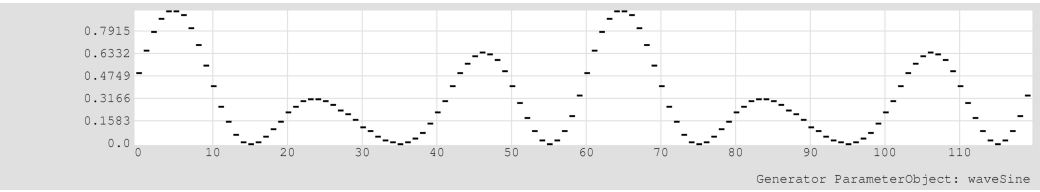

```
waveSine, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-165. waveSine Demonstration 2



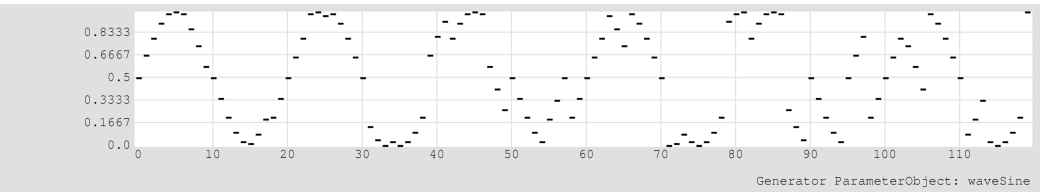

```
waveSine, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-166. waveSine Demonstration 3




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

Example C-167. waveSine Demonstration 4




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

C.1.80. waveSawDown (wsd)

waveSawDown, stepString, parameterObject, phase, min, max

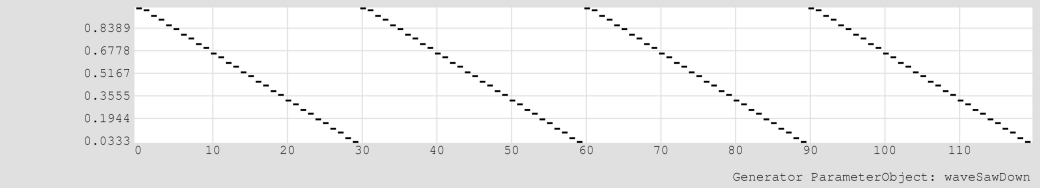
Description: Provides a saw-down wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and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

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

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

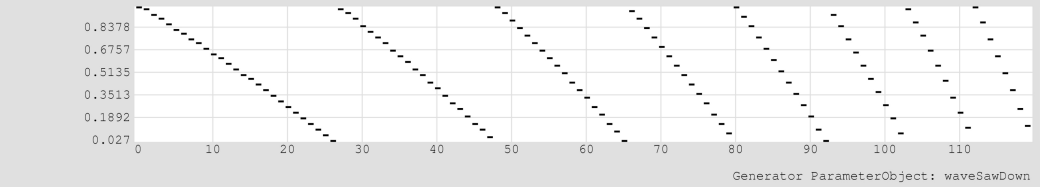
Sample Arguments: `wsd, e, 30, 0, 0, 1`

Example C-168. waveSawDown Demonstration 1



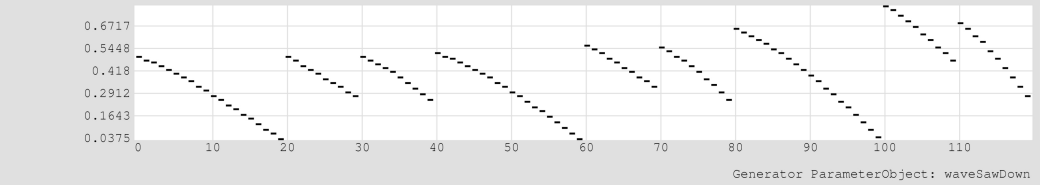

```
waveSawDown, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-169. waveSawDown Demonstration 2




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

Example C-170. waveSawDown Demonstration 3




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

C.1.81. waveSawUp (wsu)

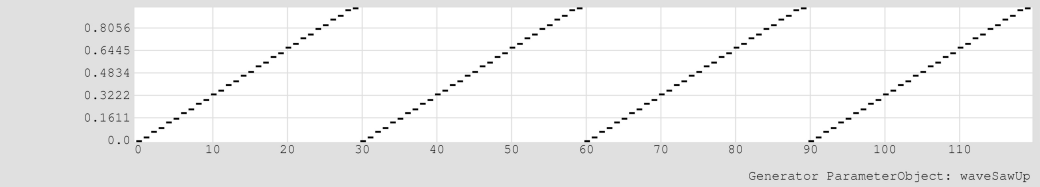
waveSawUp, stepString, parameterObject, phase, min, max

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

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

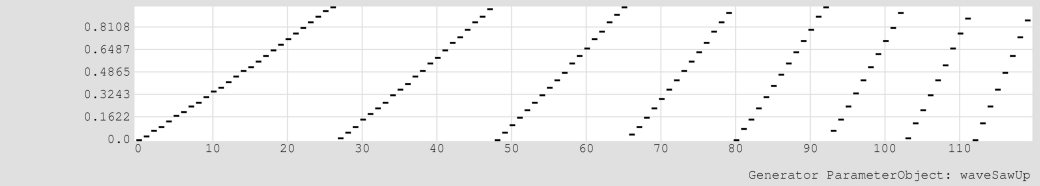
Sample Arguments: `wsu, e, 30, 0, 0, 1`

Example C-171. waveSawUp Demonstration 1



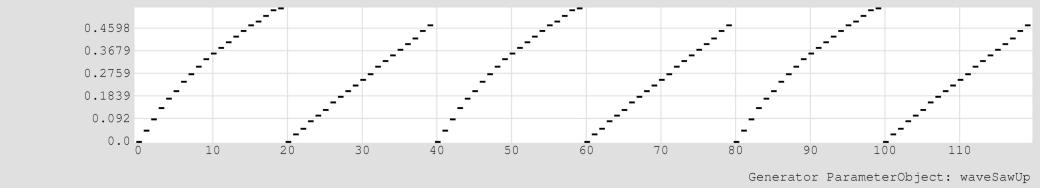

```
waveSawUp, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-172. waveSawUp Demonstration 2




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

Example C-173. waveSawUp Demonstration 3




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

C.1.82. waveTriangle (wt)

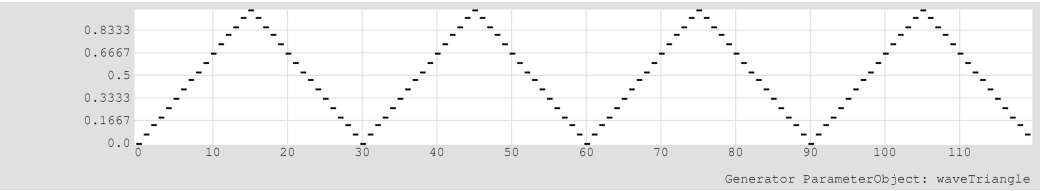
waveTriangle, stepString, parameterObject, phase, min, max

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

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

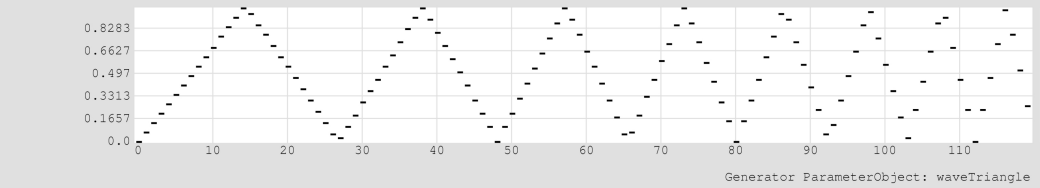
Sample Arguments: `wt, e, 30, 0, 0, 1`

Example C-174. waveTriangle Demonstration 1



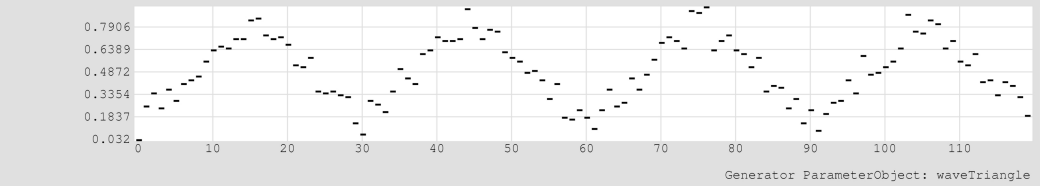

```
waveTriangle, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-175. waveTriangle Demonstration 2




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

Example C-176. waveTriangle Demonstration 3




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

C.2. Rhythm ParameterObjects

C.2.1. binaryAccent (ba)

binaryAccent, pulseList

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

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

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

C.2.2. convertSecond (cs)

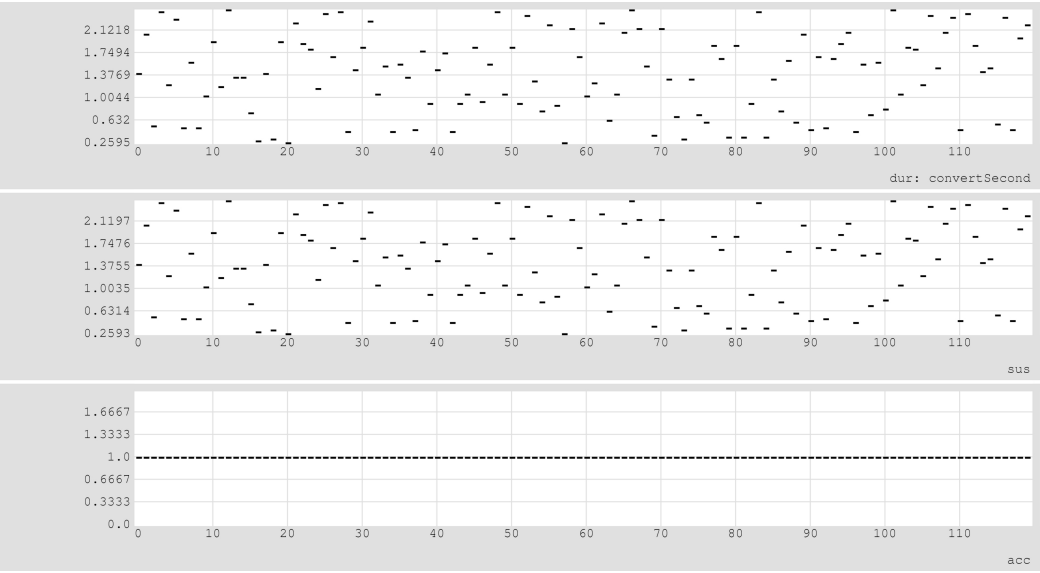
convertSecond, parameterObject

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

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

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

Example C-177. convertSecond Demonstration 1




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

C.2.3. convertSecondTriple (cst)

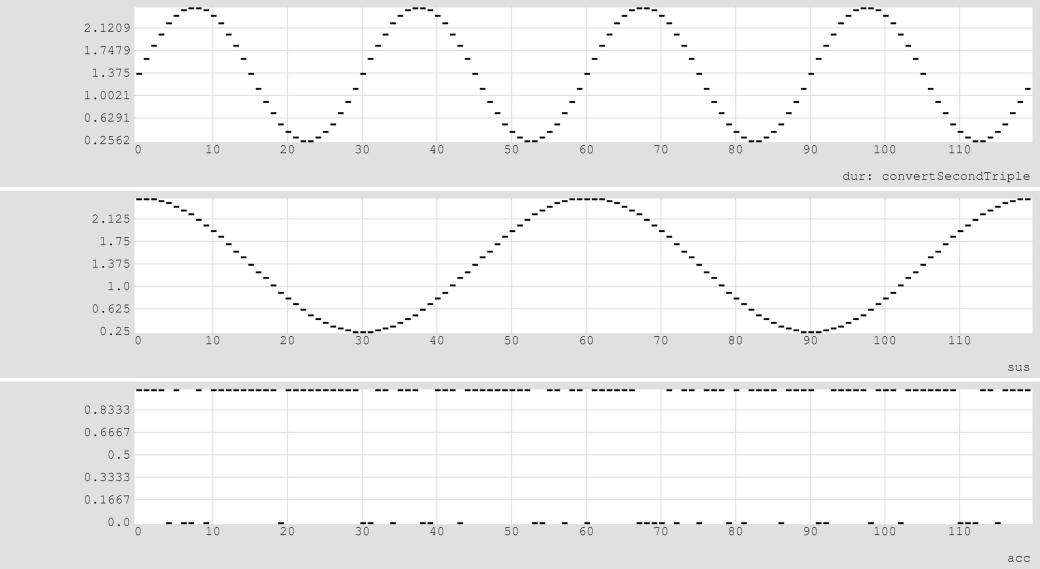
convertSecondTriple, parameterObject, parameterObject, parameterObject

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

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

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

Example C-178. convertSecondTriple Demonstration 1




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

C.2.4. gaRhythm (gr)

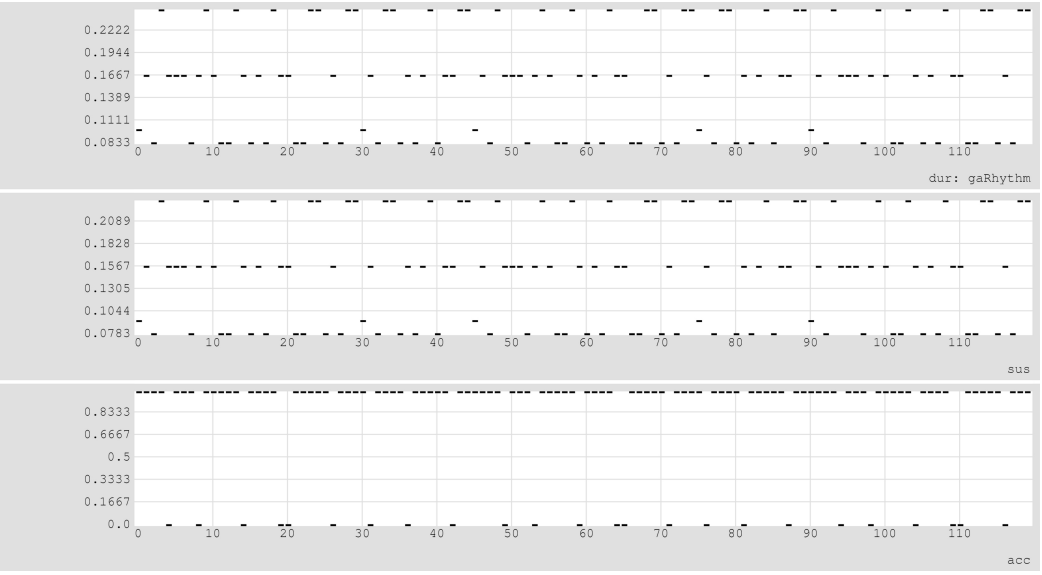
gaRhythm, pulseList, crossover, mutation, elitism, selectionString, populationSize

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

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

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

Example C-179. gaRhythm Demonstration 1




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

C.2.5. iterateRhythmGroup (irg)

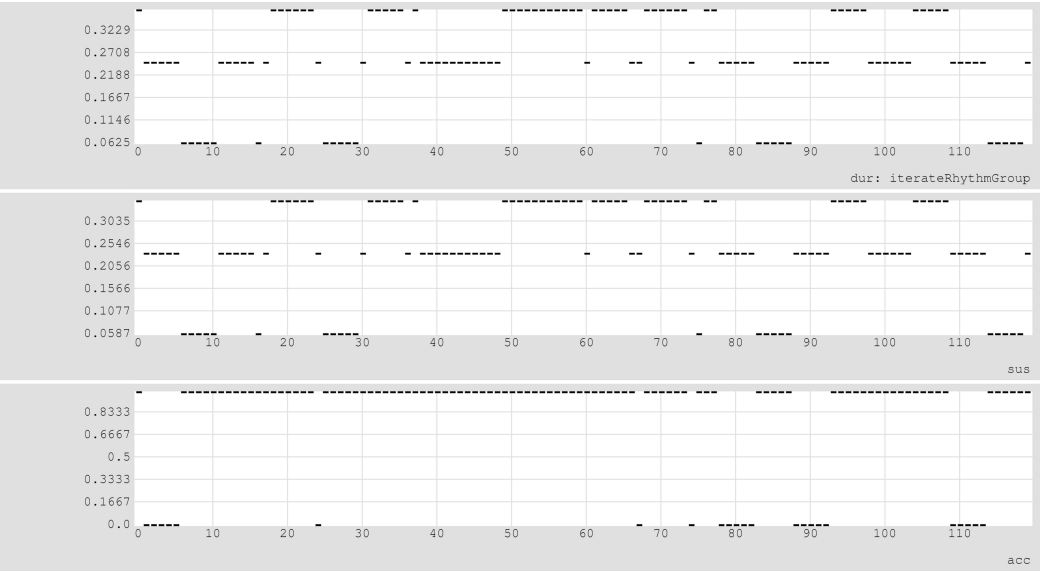
iterateRhythmGroup, parameterObject, parameterObject

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

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

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

Example C-180. iterateRhythmGroup Demonstration 1




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

C.2.6. iterateRhythmHold (irh)

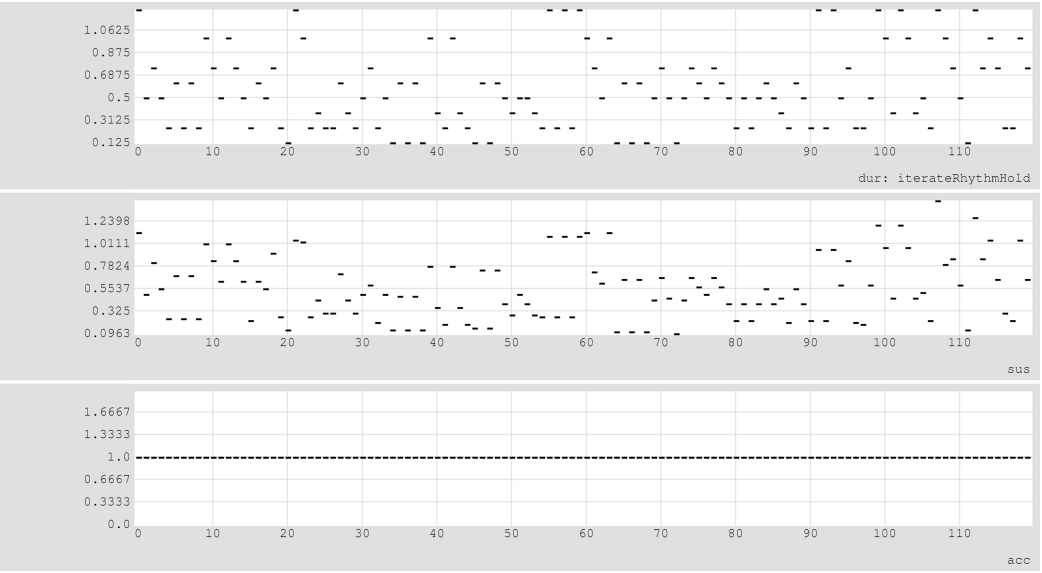
iterateRhythmHold, parameterObject, parameterObject, parameterObject, selectionString

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

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

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

Example C-181. iterateRhythmHold Demonstration 1




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

C.2.7. iterateRhythmWindow (irw)

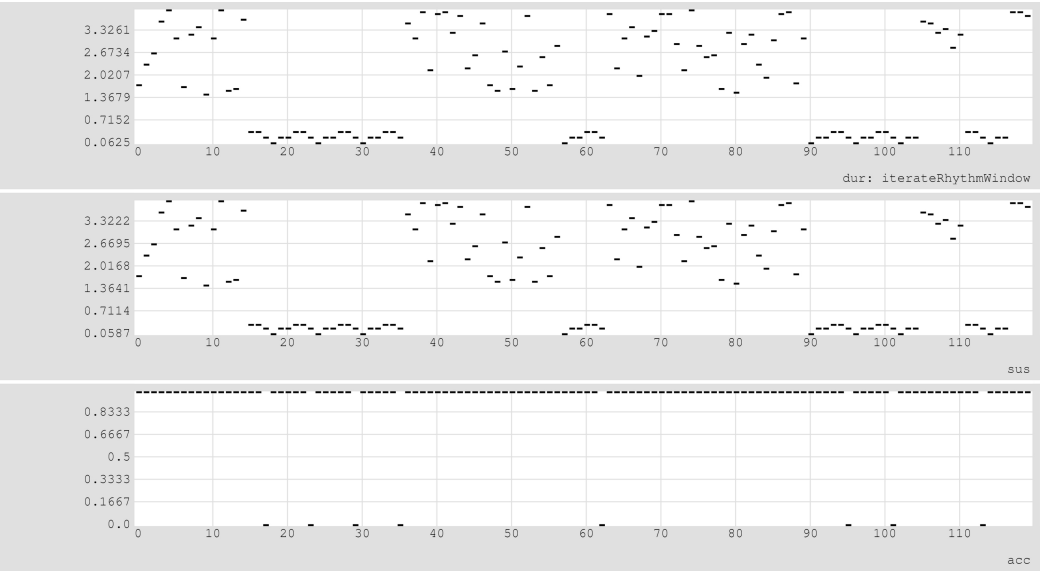
iterateRhythmWindow, parameterObjectList, parameterObject, selectionString

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

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

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

Example C-182. iterateRhythmWindow Demonstration 1




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

C.2.8. loop (l)

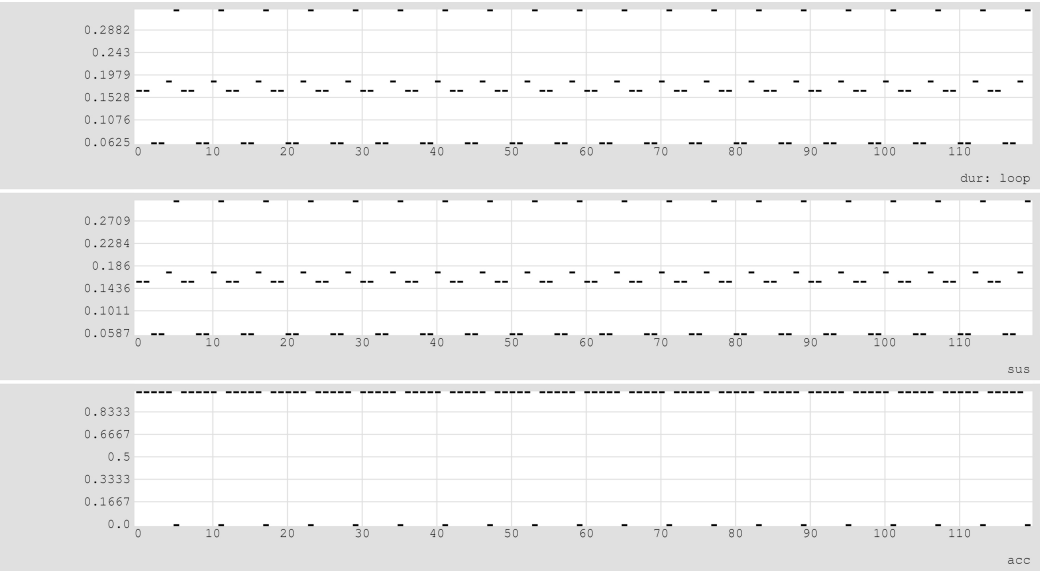
loop, pulseList, selectionString

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

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

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

Example C-183. loop Demonstration 1




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

C.2.9. markovPulse (mp)

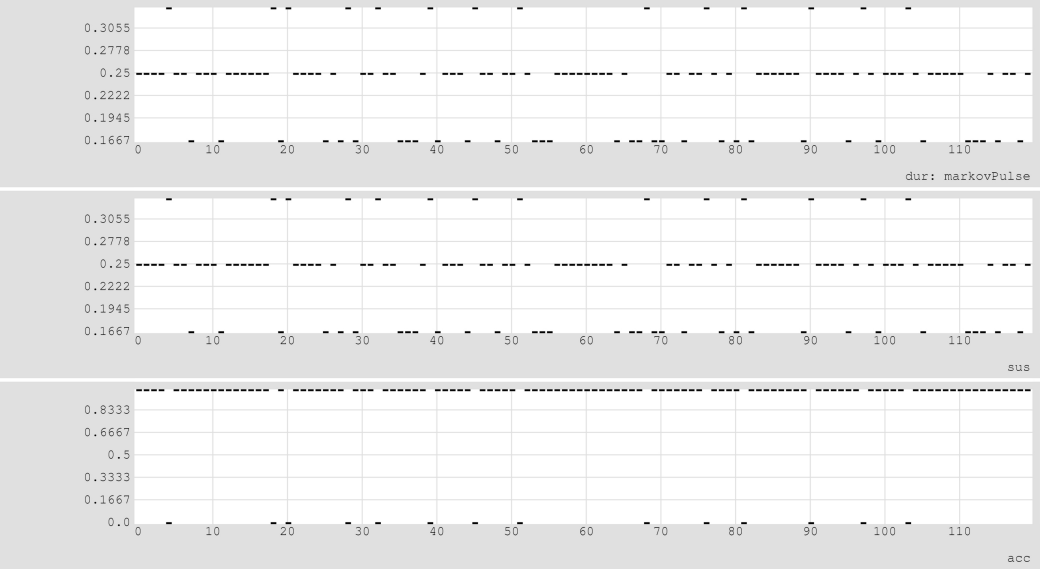
markovPulse, transitionString, parameterObject

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

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

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

Example C-184. markovPulse Demonstration 1




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

C.2.10. markovRhythmAnalysis (mra)

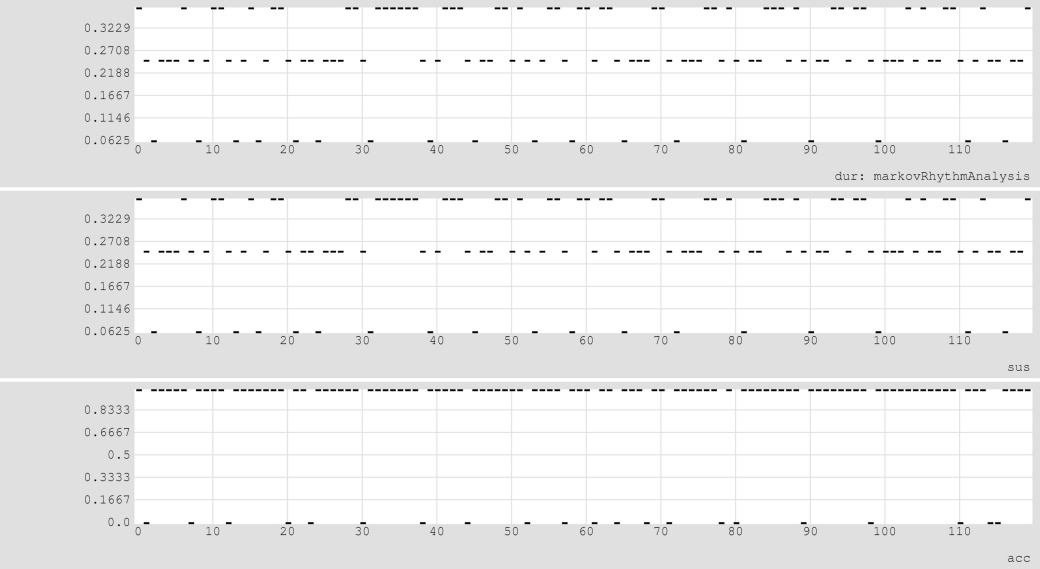
markovRhythmAnalysis, parameterObject, pulseCount, maxAnalysisOrder, parameterObject

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

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

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

Example C-185. markovRhythmAnalysis Demonstration 1




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

C.2.11. pulseSieve (ps)

pulseSieve, logicalString, sieveLength, pulse, selectionString, articulationString

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

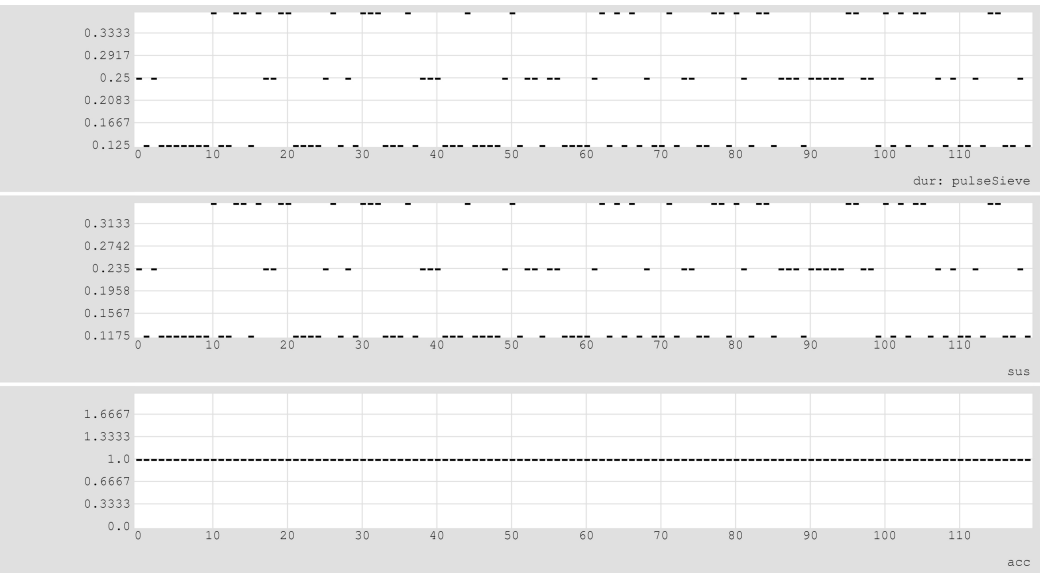
Arguments: (1) name, (2) logicalString, (3) sieveLength, (4) pulse {a single Pulse notation}, (5) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}, (6) articulationString {'attack', 'sustain'}

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

Example C-186. pulseSieve Demonstration 1


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

Example C-187. pulseSieve Demonstration 2




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

C.2.12. pulseTriple (pt)

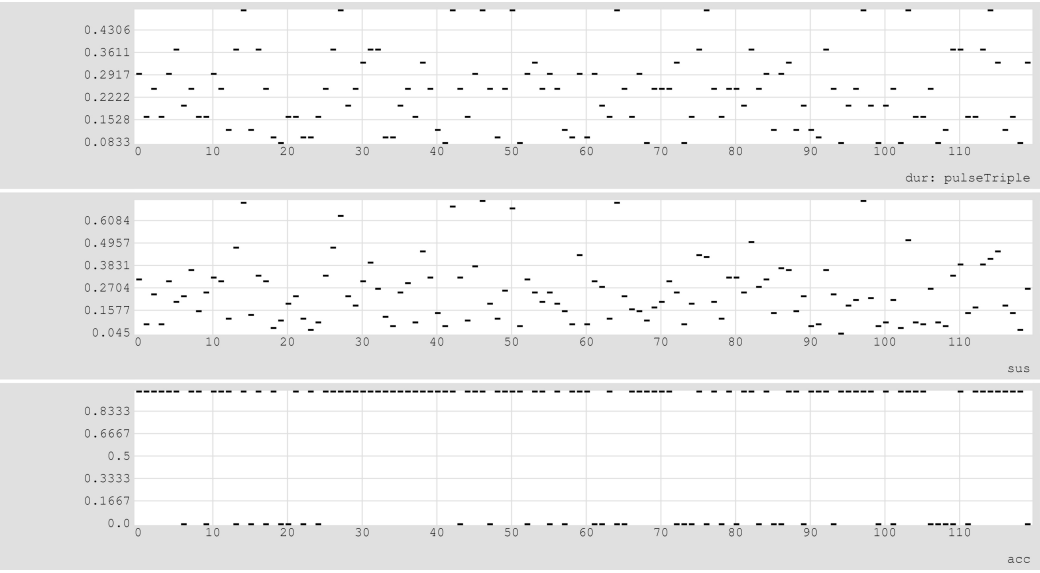
pulseTriple, parameterObject, parameterObject, parameterObject, parameterObject

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

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

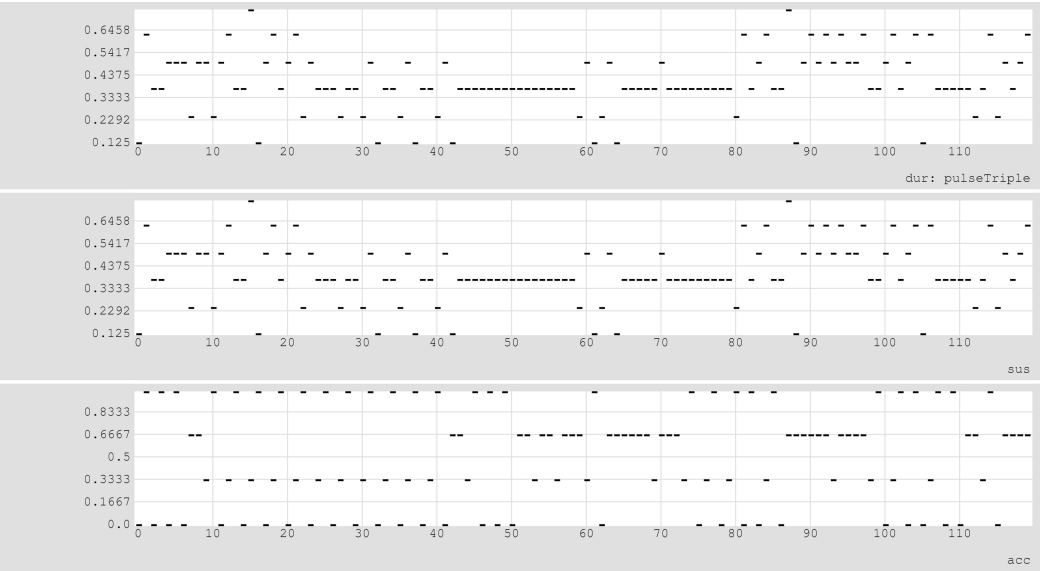
Sample Arguments: `pt, (bg,rc,(6,5,4,3)), (bg,rc,(1,2,3)), (bg,rc,(1,1,1,0)), (ru,0.5,1.5)`

Example C-188. pulseTriple Demonstration 1




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

Example C-189. pulseTriple Demonstration 2




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

C.2.13. rhythmSieve (rs)

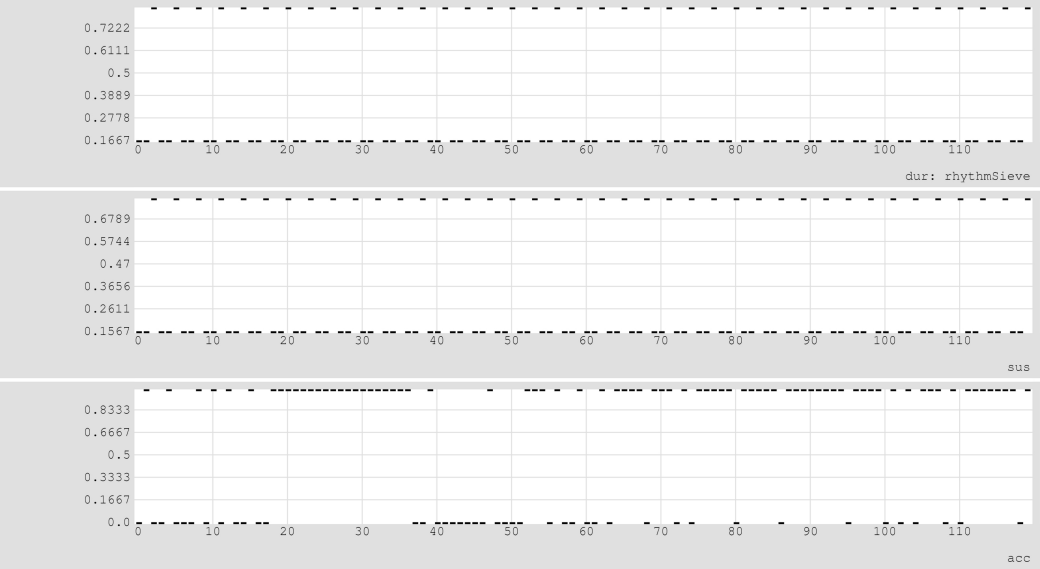
rhythmSieve, logicalString, sieveLength, selectionString, parameterObject

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

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

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

Example C-190. rhythmSieve Demonstration 1




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

C.3. Filter ParameterObjects

C.3.1. bypass (b)

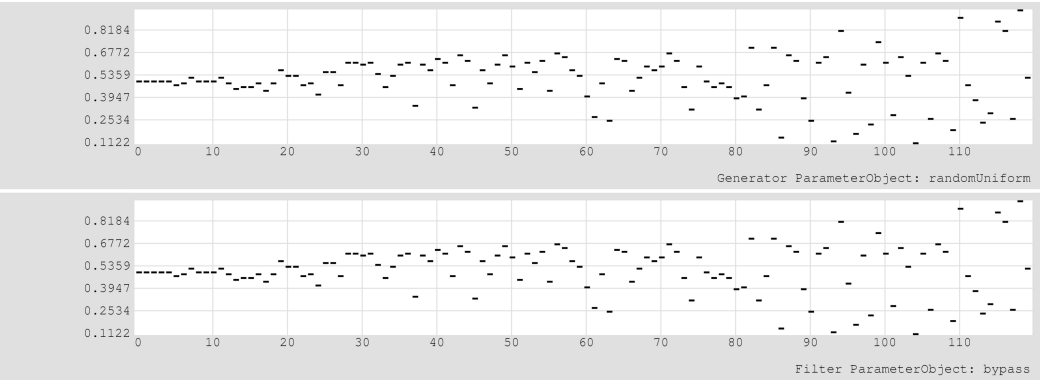
bypass

Description: Each input value is returned unaltered.

Arguments: (1) name

Sample Arguments: `b`

Example C-191. bypass Demonstration 1




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

C.3.2. filterAdd (fa)

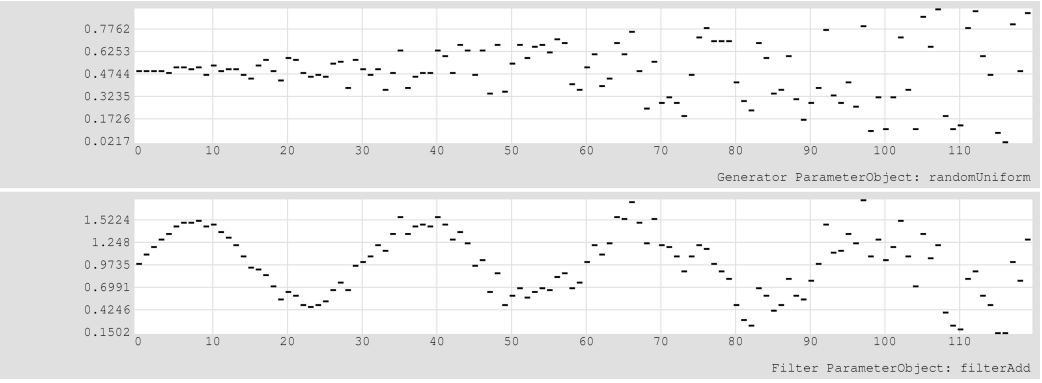
filterAdd, parameterObject

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

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

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

Example C-192. filterAdd Demonstration 1




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

C.3.3. filterDivide (fd)

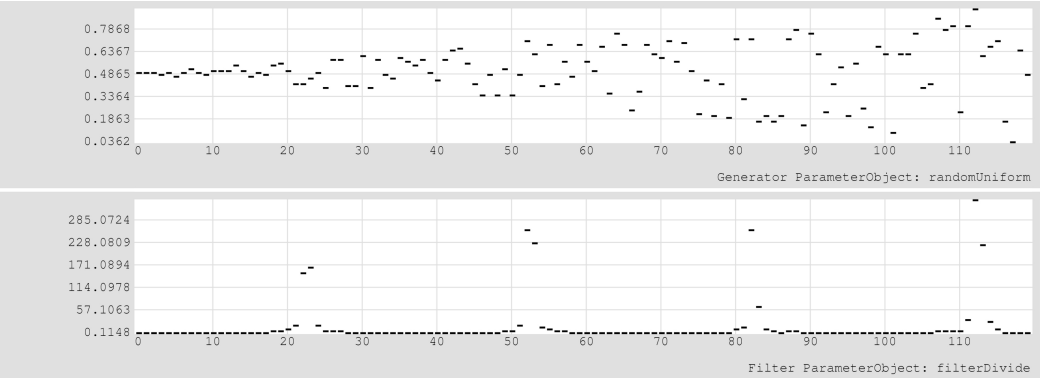
filterDivide, parameterObject

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

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

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

Example C-193. filterDivide Demonstration 1




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

C.3.4. filterDivideAnchor (fda)

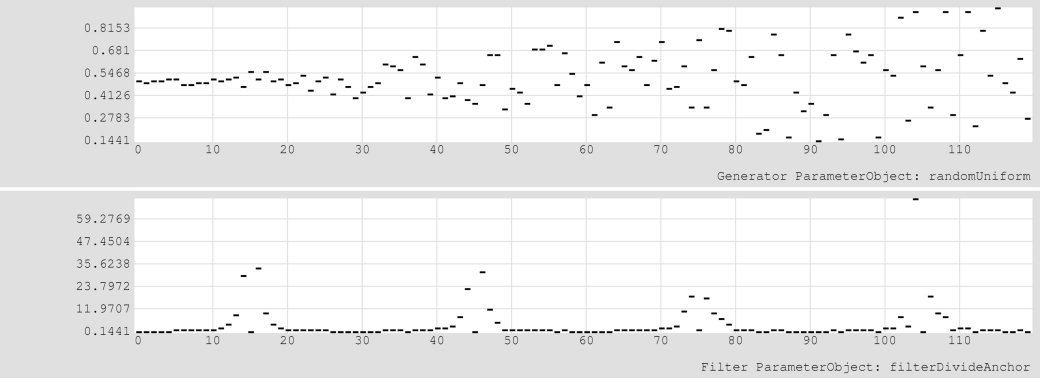
filterDivideAnchor, anchorString, parameterObject

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

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

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

Example C-194. filterDivideAnchor Demonstration 1




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

C.3.5. filterFunnelBinary (ffb)

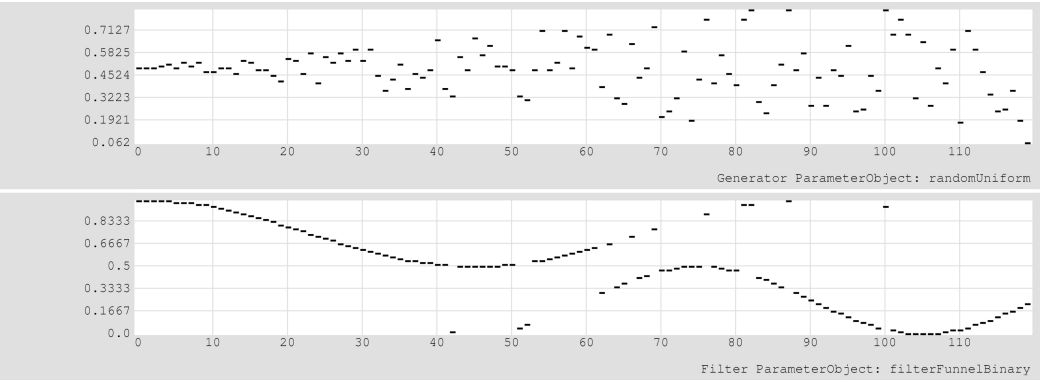
filterFunnelBinary, thresholdMatchString, parameterObject, parameterObject, parameterObject

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

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

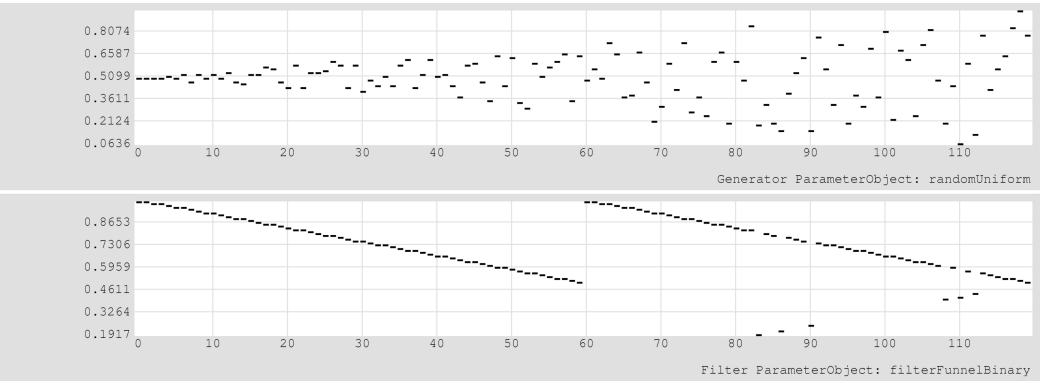
Sample Arguments: `ffb, u, (bpl,e,s,((0,0),(120,1))), (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1)`

Example C-195. filterFunnelBinary Demonstration 1




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

Example C-196. filterFunnelBinary Demonstration 2




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

C.3.6. filterMultiply (fm)

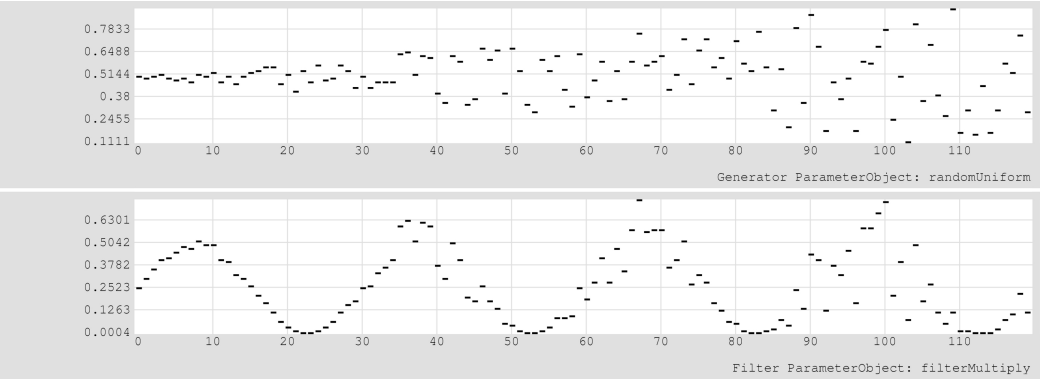
filterMultiply, parameterObject

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

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

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

Example C-197. filterMultiply Demonstration 1




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

C.3.7. filterMultiplyAnchor (fma)

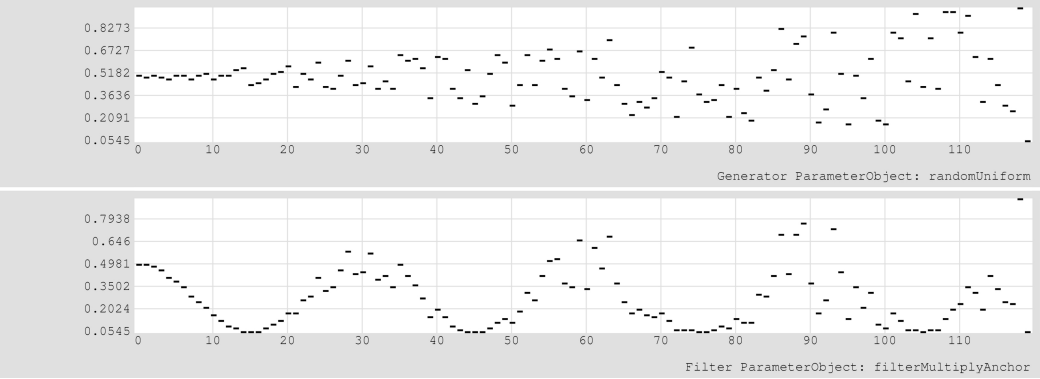
filterMultiplyAnchor, anchorString, parameterObject

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

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

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

Example C-198. filterMultiplyAnchor Demonstration 1




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

C.3.8. filterPower (fp)

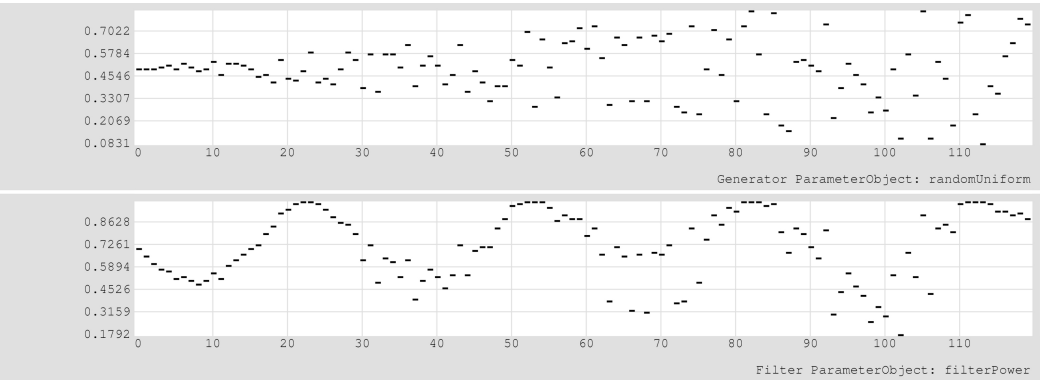
filterPower, parameterObject

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

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

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

Example C-199. filterPower Demonstration 1




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

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

C.3.9. filterQuantize (fq)

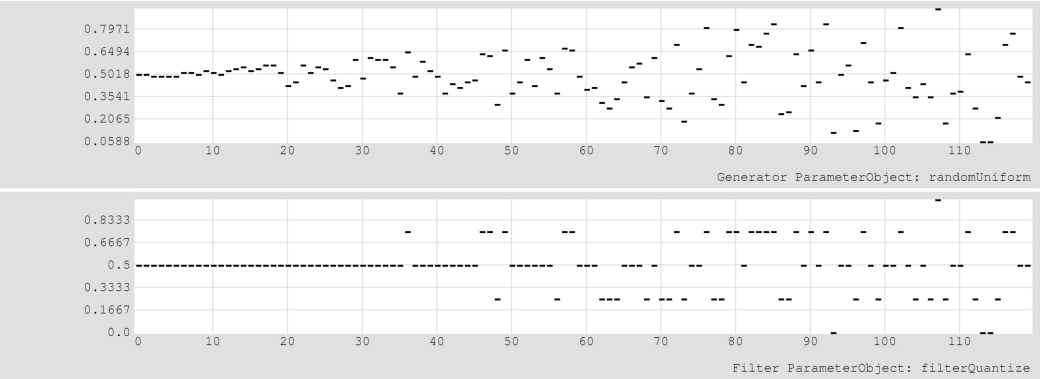
filterQuantize, parameterObject, parameterObject, stepCount, parameterObject

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

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

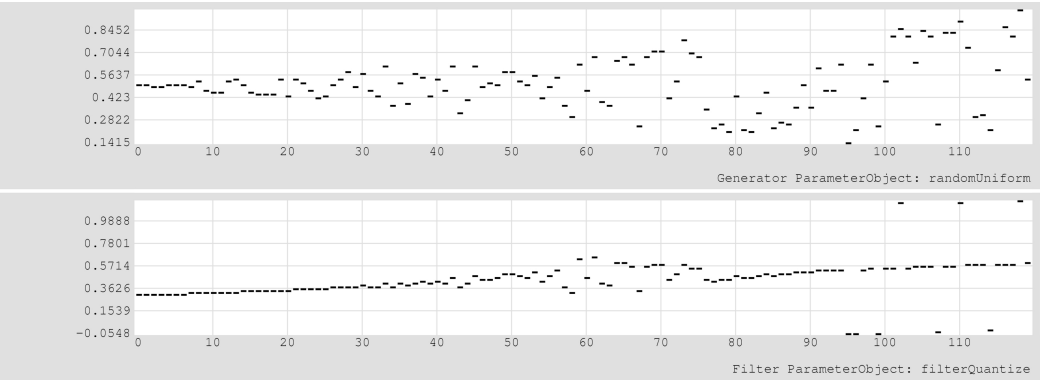
Sample Arguments: `fq, (c,0), (c,0.25), 1, (c,1)`

Example C-200. filterQuantize Demonstration 1




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

Example C-201. filterQuantize Demonstration 2




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

C.3.10. maskFilter (mf)

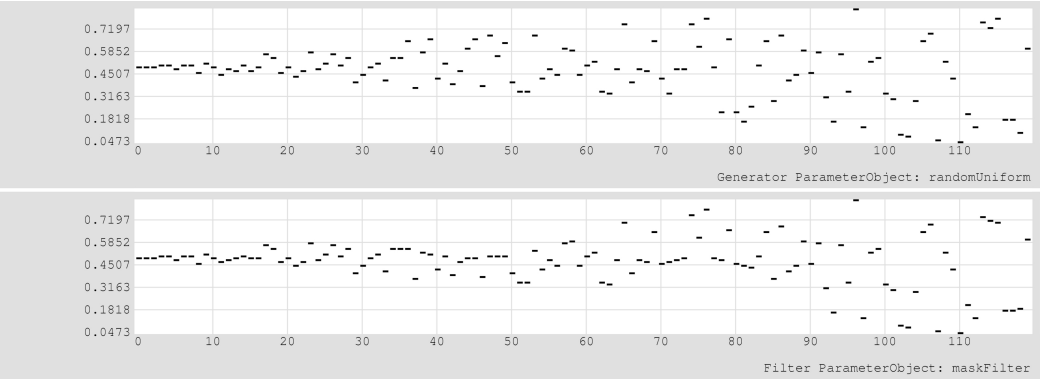
maskFilter, boundaryString, parameterObject, parameterObject

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

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

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

Example C-202. maskFilter Demonstration 1



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

C.3.11. maskScaleFilter (msf)

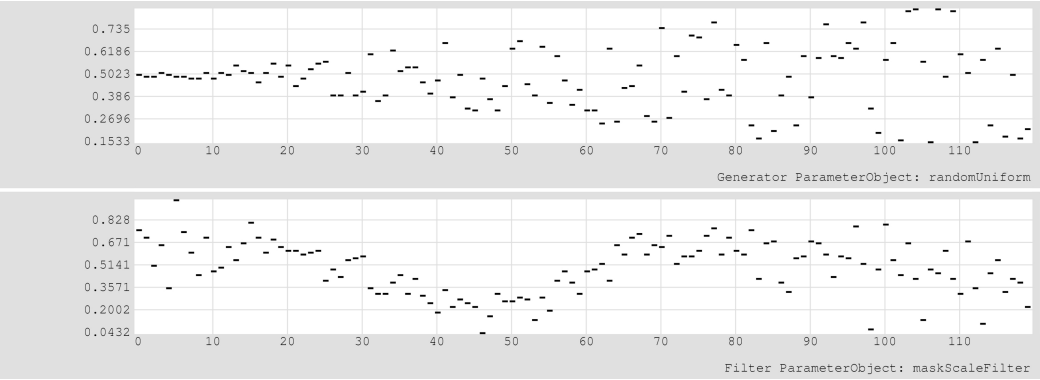
maskScaleFilter, min, max, selectionString

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

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

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

Example C-203. maskScaleFilter Demonstration 1




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

C.3.12. orderBackward (ob)

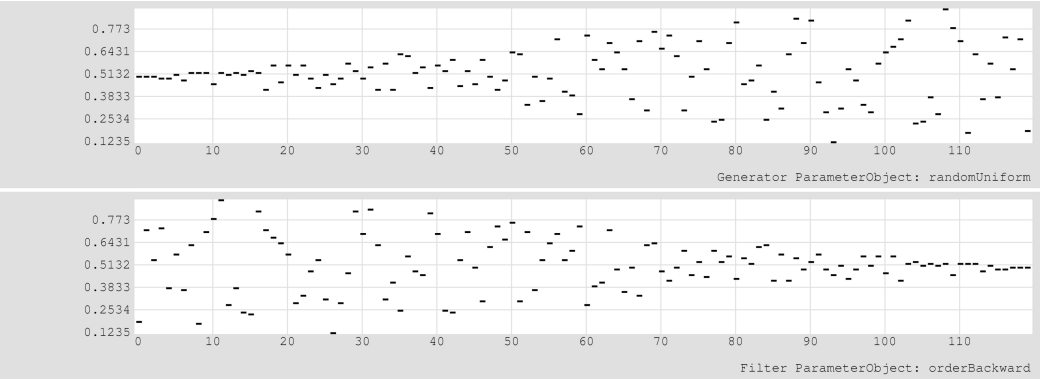
orderBackward

Description: All values input are returned in reversed order.

Arguments: (1) name

Sample Arguments: `ob`

Example C-204. orderBackward Demonstration 1




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

C.3.13. orderRotate (or)

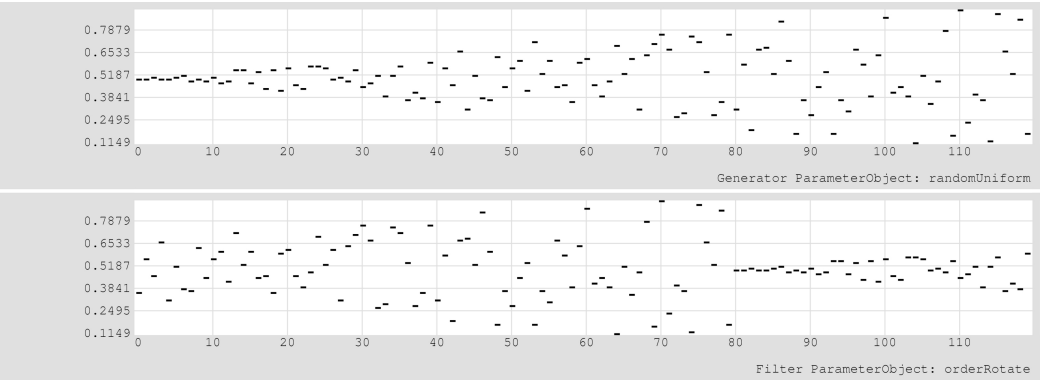
orderRotate, rotationSize

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

Arguments: (1) name, (2) rotationSize

Sample Arguments: `or, 40`

Example C-205. orderRotate Demonstration 1



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

C.3.14. pipeLine (pl)

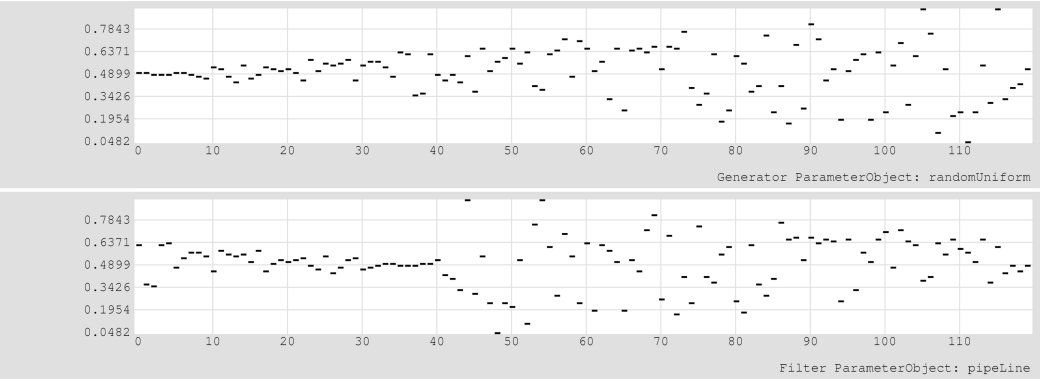
pipeLine, filterParameterObjectList

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

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

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

Example C-206. pipeLine Demonstration 1




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

C.3.15. replace (r)

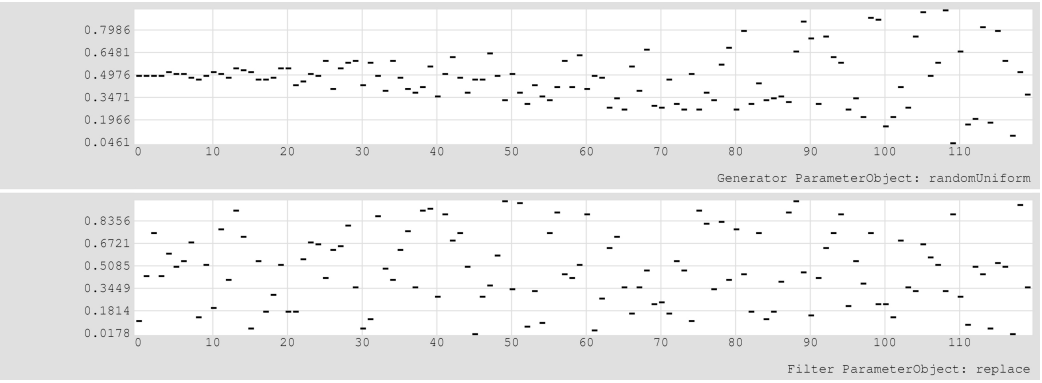
replace, parameterObject

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

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

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

Example C-207. replace Demonstration 1




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

C.4. TextureStatic ParameterObjects

C.4.1. eventDensityPartition (edp)

eventDensityPartition, level

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

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

Sample Arguments: `edp, duration`

C.4.2. interpolationMethodControl (imc)

interpolationMethodControl, method

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

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

Sample Arguments: `imc, linear`

C.4.3. levelEventCount (lec)

levelEventCount, level

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

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

Sample Arguments: `lec, segment`

C.4.4. levelEventPartition (lep)

levelEventPartition, level

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

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

Sample Arguments: `lep, path`

C.4.5. levelFrameDuration (lfd)

levelFrameDuration, level

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

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

Sample Arguments: `lfd, event`

C.4.6. levelFieldMonophonic (lfm)

levelFieldMonophonic, level

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

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

Sample Arguments: `lfm, event`

C.4.7. levelFieldPolyphonic (lfp)

levelFieldPolyphonic, level

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

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

Sample Arguments: `lfp, event`

C.4.8. levelOctaveMonophonic (lom)

levelOctaveMonophonic, level

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

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

Sample Arguments: `lom, event`

C.4.9. levelOctavePolyphonic (lop)

levelOctavePolyphonic, level

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

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

Sample Arguments: `lop, event`

C.4.10. loopWithinSet (lws)

loopWithinSet, onOff

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

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

Sample Arguments: `lws, on`

C.4.11. multisetSelectorControl (msc)

multisetSelectorControl, selectionString

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

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

Sample Arguments: `msc, randomPermutate`

C.4.12. maxTimeOffset (mto)

maxTimeOffset, time

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

Arguments: (1) name, (2) time

Sample Arguments: `mto, 0.03`

C.4.13. ornamentLibrarySelect (ols)

ornamentLibrarySelect, libraryName

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

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

Sample Arguments: `ols, diatonicGroupA`

C.4.14. ornamentMaxDensity (omd)

ornamentMaxDensity, percent

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

Arguments: (1) name, (2) percent

Sample Arguments: `omd, 1`

C.4.15. pathDurationFraction (pdf)

pathDurationFraction, onOff

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

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

Sample Arguments: `pdf, on`

C.4.16. parameterInterpolationControl (pic)

parameterInterpolationControl, onOff

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

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

Sample Arguments: `pic, on`

C.4.17. parallelMotionList (pml)

parallelMotionList, transpositionList, timeDelay

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

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

Sample Arguments: `pml, (), 0.0`

C.4.18. pitchSelectorControl (psc)

pitchSelectorControl, selectionString

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

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

Sample Arguments: `psc, randomPermutate`

C.4.19. snapEventTime (set)

snapEventTime, onOff

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

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

Sample Arguments: `set, on`

C.4.20. snapSustainTime (sst)

snapSustainTime, onOff

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

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

Sample Arguments: `sst, on`

C.4.21. totalEventCount (tec)

totalEventCount, count

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

Arguments: (1) name, (2) count

Sample Arguments: `tec, 20`

C.4.22. totalSegmentCount (tsc)

totalSegmentCount, count

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

Arguments: (1) name, (2) count

Sample Arguments: `tsc, 10`

C.5. CloneStatic ParameterObjects

C.5.1. retrogradeMethodToggle (rmt)

retrogradeMethodToggle, name

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

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

Sample Arguments: `rmt, off`

C.5.2. timeReferenceSource (trs)

timeReferenceSource, name

Description: Selects time reference source used in calculating ParameterObjects.

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

Sample Arguments: `trs, textureTime`

Appendix D. Temperament and TextureModule Reference

D.1. Temperaments

D.1.1. Temperament Interleave24Even

Even steps of a 24 tone equal tempered scale

D.1.2. Temperament Interleave24Odd

Odd steps of a 24 tone equal tempered scale

D.1.3. Temperament Just

Static Just tuning

D.1.4. Temperament MeanTone

Static Mean Tone tuning

D.1.5. Temperament NoiseHeavy

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

D.1.6. Temperament NoiseLight

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

D.1.7. Temperament NoiseMedium

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

D.1.8. Temperament Pythagorean

Static Pythagorean tuning

D.1.9. Temperament Split24Lower

Lower half of a 24 tone equal tempered scale

D.1.10. Temperament Split24Upper

Upper half of a 24 tone equal tempered scale

D.1.11. Temperament TwelveEqual

Twelve tone equal temperament

D.2. TextureModules

D.2.1. TextureModule DroneArticulate

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

D.2.2. TextureModule DroneSustain

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

D.2.3. TextureModule HarmonicAssembly

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

D.2.4. TextureModule HarmonicShuffle

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

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

D.2.5. TextureModule InterpolateFill

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

D.2.6. TextureModule InterpolateLine

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

D.2.7. TextureModule IntervalExpansion

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

D.2.8. TextureModule LineCluster

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

D.2.9. TextureModule LineGroove

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

D.2.10. TextureModule LiteralHorizontal

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

D.2.11. TextureModule LiteralVertical

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

D.2.12. TextureModule MonophonicOrnament

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

D.2.13. TextureModule TimeFill

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

D.2.14. TextureModule TimeSegment

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

Appendix E. OutputFormat and OutputEngine Reference

E.1. OutputFormats

E.1.1. acToolbox

acToolbox: AC Toolbox Environment file. (.act)

E.1.2. audioFile

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

E.1.3. csoundBatch

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

E.1.4. csoundData

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

E.1.5. csoundOrchestra

csoundOrchestra: Csound orchestra file. (.orc)

E.1.6. csoundScore

csoundScore: Csound score file. (.sco)

E.1.7. midiFile

midiFile: Standard MIDI file. (.mid)

E.1.8. pureDataArray

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

E.1.9. scScd

scScd: SuperCollider task data format. (.scd)

E.1.10. textSpace

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

E.1.11. textTab

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

E.1.12. xmlAthenaObject

xmlAthenaObject: athenaCL native XML format. (.xml)

E.2. OutputEngines

E.2.1. EngineAcToolbox

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

E.2.2. EngineAudioFile

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

E.2.3. EngineCsoundExternal

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

E.2.4. EngineCsoundNative

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

E.2.5. EngineCsoundSilence

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

E.2.6. EngineMidiFile

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

E.2.7. EnginePureDataArray

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

E.2.8. EngineSuperColliderTask

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

E.2.9. EngineText

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

Appendix F. Demonstration Command Scripts in Python

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

F.1. MIDI-based Output

F.1.1. script01a.py: Configuring Rhythms

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

F.1.2. script01b.py: Configuring Time Range

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

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

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

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

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

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

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

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

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

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

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

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

F.1.7. script02e.py: Creating Mensural Canons

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

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

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

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

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

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

F.1.10. script03a.py: Grouping Selection

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

F.2. Csound-based Output

F.2.1. script01a.py: Testing Csound

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

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

F.2.2. script01b.py: A Noise Instrument

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix G. Frequently Asked Questions

General Information

Q: Can users add Csound instruments to athenaCL?

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

Q: How can I contribute to this project?

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

Q: How much does athenaCL cost?

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

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

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

Q: What does athenaCL do?

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

Q: What is Python?

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

Q: What is an interactive command-line program?

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

Q: Where can I ask questions about athenaCL?

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

Q: Where is the source code?

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

Q: Who is athenaCL designed for?

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

Installing, Starting, and Uninstalling athenaCL

Q: How do I uninstall athenaCL?

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

Q: Is Csound required to use athenaCL?

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

Q: Is Python required to use athenaCL?

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

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

Q: What platforms does athenaCL run on?

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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