AthenaCL Score Generator Manual

referencing AthenaCL version 2.1

https://github.com/tjingboem/AthenaCL-for-Blue

program created by Christopher Ariza
manual updated by Menno Knevel
for use with Blue (https://blue.kunstmusik.com/)

Published August 2024

Preface

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.

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.

Version 2.1 introduces two seed options: one for the Texture and one for all ParameterObjects. This allows to ‘pin down’ the generated random data.

Integration of AthenaCL in Blue: the AthenaCL Score Generator

Programs that are written in the Python language are not among the fastest and it does take a few seconds for a Csound Score to be generated. It may help speeding up the work process to Freeze the ObjectBuilder containing the AthenaCL code, especially if more than one of these Objects are used in the composition. By Freezing the ObjectBuilder, the code is rendered into a sound file. A waveform is displayed and CPU cycles are saved. This sound file can be transformed back to the original AthenaCL Score Generator code at any time as long as no files in the directory where the saved files are deleted.

Most of the functionality of the AthenaCL code in Blue is redundant. The AthenaCL Score Generator is a slim version, using only elements that make sense in the context of working with Blue.

The creator of Blue, Steven Yi, created a Python program called athenaPipe.py; an interface between Blue and an installed AthenaCL. It reads the data from a temporary text file and brings it as a score data back into Blue. The ObjectBuilder (the ‘container’of the AthenaCL Score Generator) and its GUI are created by Menno Knevel.

Some words on how the AthenaCL Score Generator works:
first, the event type is set to CsoundExternal (**emo ce**), then the seeds options for the TextureModule and TextureParameters (ParameterObjects) are set (**tmsd** and **tpsd**). A TextureModule is chosen (**tmo**) and a Path for the notes or frequencies is set (**pin**).

By default, the order of the pfields is as follows:

p1	instrument number
p2	event start
p3	event duration
p4	note or frequencies (pitch)
p5….p12	appointed by the user

4 Multisets and its weightings are set (**pidf**) and a new Texture named ‘Objectname’ is created (**tin**). The Texture Temperament (the grid for the pitches) is set (**tto**). This allows for micro-tonality. All ParameterObjects are set (**tie**) for p5 to p12.

CAVEAT: Be sure not to make mistakes in typing the () and commas in the format as needed by AthenaCL. Any typo results in a blank Generated Score Screen in Blue.

In order for the Filters option of AthenaCL to be used, a Texture Clone has to be created (**tcn**). To prevent that both the original and the Clone produce scores and create doubles, the original is muted (**timute**) and only the Texture Clone data are generated.

All Filter ParameterObjects can be edited or just set to bypass (**tce**).

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.

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.

To create a PathInstance, enter PIn (for PathInstance new).
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.

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.

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.

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.

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.

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 accessedwith acronyms.

Generator ParameterObject:

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:

binaryAccent
convertSecond
convertSecondTriple
gaRhythm
iterateRhythmGroup
iterateRhythmHold
iterateRhythmWindow
loop
markovPulse
markovRhythmAnalysis
pulseSieve
pulseTriple
rhythmSieve

Filter ParameterObject:

bypass
filterAdd
filterDivide
filterDivideAnchor
filterFunnelBinary
filterMultiply
filterMultiplyAnchor
filterPower
filterQuantize
maskFilter
maskScaleFilter
orderBackward
orderRotate
pipeLine
replace

For details on the ParameterObjects, see the AthenaCL Manual.

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).

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.

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.

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.

Temperaments

Interleave24Even
Even steps of a 24 tone equal tempered scale
Interleave24Odd
Odd steps of a 24 tone equal tempered scale
Just
Static Just tuning

MeanTone
Static Mean Tone tuning
NoiseHeavy
Provide uniform random +/- 15 cent noise on each pitch
NoiseLight
Provide uniform random +/- 5 cent noise on each pitch
NoiseMedium
Provide uniform random +/- 10 cent noise on each pitch
NoiseUser
Provide uniform random +/- 50 cent noise on each pitch
Pythagorean
Static Pythagorean tuning
Split24Lower
Lower half of a 24 tone equal tempered scale
Split24Upper
Upper half of a 24 tone equal tempered scale
TwelveEqual
Twelve tone equal temperament

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.

TextureModules

A Texture is an instance of a TextureModule. Every time a Texture is created, athenaCL creates an independent instance of the active 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.

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.

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.

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.

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.

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.

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.

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

voicings.

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.

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.

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.

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.

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.

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.

Listing of commands used by the AthenaCL Score Generator

PathInstance:

Pin(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.

Pidf (duration)- 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.

TextureModule:

Tmo (select) - Choose the active TextureModule.

Tmsd (seed) - Sets the global random seed if used by the Texture Modules.

TextureParameter:

Tpsd (seed) - Sets the global random seed used by Texture Parameters.

TextureInstance:

Tin (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.

Tie (edit) - Edit a user-selected attribute of the active Texture.

Timute (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.

TextureClone:

Tcn (new) - Creates a new Clone associated with the active Texture.

Tce (edit) - Edit attributes of the active Clone.

TextureTemperament:

Tto (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.

EventMode:

Emo (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.

appendix

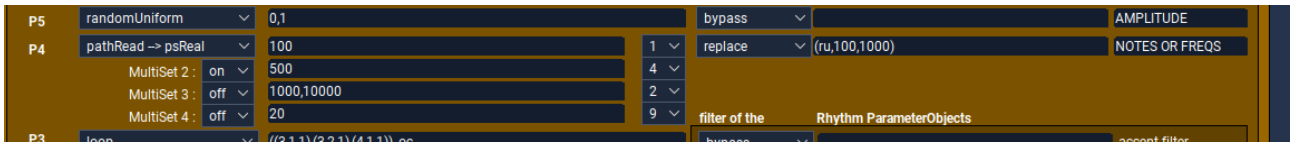
User case 1: Pitch treatment

The AthenaCL Score Generator can be found in BlueShare and imported in Blue.

The default settings for pitch consists of frequencies and 4 Multisets for pfield 4. Here, 2 Multisets are active:

The pathRead is set to Frequncies (psReal). Frequency of 100 Hz and 500 Hz are entered. The frequency of 500 is 4, the frequency for 100 is 100. This means that 500 is used 4 times more than 100 in the Texture. The Filter is bypassed.

However, it is possible to use ParameterObjects for frequency generation, for example in the example below, a random frequency between 100 and 1000 (ru,100,1000). In this case the ParameterObject replaces the pathRead→psReal values:



User case 2: Object Length

For the generated rhythm to correspond with the length of the Athena Score Generator, set the Object length accordingly. Of course, if desired, as you may use this option to speed up or slow down the rhythm.