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