Sonification in ChucK, MiniAudicle and STK
Perry R. Cook, ICAD Workshop, Atlanta, June 2012

ChucK Language Intro, Basics:

Hello Sine!

```
SinOsc s => dac;       // connect a sine oscillator to audio out
1.0 :: second => now;  // advance time by 1 second (thus, sound)
```

The ChucK Operator connection (patching), assignment, argument passing, time

```
SinOsc sl => dac.left;  // connect sine osc to dac left channel
SinOsc sr => dac.right; // connect another sine to dac right channel
200.0 => sl.freq;      // assign 200.0 to the frequency of left sine
sr.freq(202.0);        // assign 202.0 to frequency of right sine
5.0 :: second => now;  // hang out for 5 seconds
204.0 => sr.freq;      // set right sine freq to 204 Hz.
4.0 :: second => now;  // hang out another 4 seconds
```

The Mini Audicle: Smart Editor, Virtual Machine Monitor, Console, Shred Handling, …

```
SinOsc s => JCre v => dac; // sine thru reverb to dac
while (true) {             // infinite loop
    Std.rand2f(200.0,1000.0) => s.freq; // 200-1000 Hz. random freq.
    0.2 :: second => now;  // hang out for 1/5 second
}
```

Modifications: 0.01 => r.mix;  // change dry/reverb mixture
Or… 1.0 :: samp => now;  // update every sample

On-the-fly Coding/Editing/Debugging: Virtual Machine, Shreds, Concurrency

Precise Control of Time: samp, second, ms, day, week, fractions/multiples of any of these

```
Impulse imp => dac;       // impulse generator (gateway) to dac
while (1) {               // infinite loop
    Std.rand2f(0.01, 0.1) :: second => now; // random waiting time
    1.0 => imp.next;          // randomly timed pulses, Geiger to noise
}
```

Data Types: int, float, time, dur, void, complex, polar
show examples of math on time, strong typing, casting, etc.

Objects: An Object is an encapsulated collection of data and behavior (more on this later)
Class, inheritance, message passing (ChucK), polymorphism (behavior)

Object: base (mother) class of all other classes
Array: N-dimensional ordered set of data (of same type)
UGen: Unit Generator base class (SinOsc, JCre v, etc. Inherit from this)
Others Event, string
Arrays, Looping

```chuck
SinOsc s => dac; // our sine buddy
[60, 62, 64, 65, 67, 69, 71, 72] => int notes[]; // array of ints
for (0 => int i; i < notes.cap(); 1 += i) {
  Std.mtof(notes[i]) => s.freq; // set freq by midi notes
  0.5 :: second => now;
}
```

Modify above example, change notes, add duration array, other

Concurrency, Synchronization

```chuck
Impulse imp => TwoPole f => JCre v => dac; // tuned “pops”
0.04 => r.mix; // dry/reverb mix
0.99 => f.radius; // set up filter to ring
0.2 :: second => dur T;
while (1) {
  Std.rand2f(500.0, 2000.0) => f.freq; // random filter freq
  1.0 => imp.next; // make an impulse
  T => now;
}
```

Fire up a few of these, note that they all work, but synchronized?
Add this line to it, before the while loop:

```chuck
T - (now % T) => now; // Huh? Compute and delay time until next T
```

Play around with changing T, changing frequency limits, etc. Music!!

Functions

Built in: Std.fabs(x), Math.sqrt(x), UG.gain(x), Machine.add(process), me.args(), ...
User Defined:

```chuck
fun int couldbe(float prob) { // return true/false based on prob 0-1
  if (Std.rand2f(0.0, 1.0) < prob) return 1;
  else return 0;
}

fun void sweepUp(float initFreq) {
  SinOsc s => dac; // alloc UG (not a good idea)
  while (s.freq() < 10.0*initFreq) { // sweep to 10x initial
    s.freq() * 1.01 => s.freq; // expon. ramp frequency
    0.01 :: second => now;
  }
  s <= dac; // unchuck (for GC (later))
}
```

Shreds, Spork(ing): non-preemptive threads

```chuck
while (1) {
  if (0.2 => couldbe) spork ~ sweepUp(100.0); // 1/5 chance low
  else spork ~ sweepUp(Std.rand2f(500.0, 1000.0)); // otherwise, hi
  Std.rand2f(0.3, 2.0) :: second => now;
}
```
Chuck Language Math Library: Math.sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(x,y) sinh(x), cosh(x), tanh(x), hypot(x,y) pow(x,y), sqrt(x), exp(x), log(x), log2(x), log10(x) floor(x), ceil(x), round(x), trunc(x), fmod(x,y), remainder(x,y) min(x,y), max(x,y), nextpow2(x) isinf(x), isnan(x)

Impulse imp => dac; // provides a means of writing directly to dac
0.0 => float phase; // make a phase variable for sine argument
while (1) {
    Math.sin(phase) => imp.next; // write out next sample
    0.1 += phase; // increment phase
    if (phase > (2*pi)) phase-(2*pi) => phase; // modulo two pi
    1.0 :: samp => now; // write out each sample
}

ChucK Language Standard Library (Std)

int abs ( int value ); returns absolute value of integer
float fabs ( float value ); returns absolute value of floating point number
int rand (); generates random integer
int rand2 ( int min, int max ); generates random integer in range [min, max]
float randf (); generates random floating point number in the range [-1, 1]
float rand2f ( float min, float max ); random float in the range [min, max]
float sgn ( float value ); compute sign of input as -1.0 (neg), 0, or 1.0 (pos)
int atoi ( string value ); converts ascii (string) to integer (int)
string itoa ( int value ); integer to ascii (string)
float atof ( string value ); convert ascii (string) to floating point value (float)
float ftoa ( string value ); float to ascii (string)
float mtotf ( float value ); converts a MIDI note number to frequency (Hz)
    note the input value is of type 'float' (supports fractional note number)
float ftom ( float value ); convert frequency (Hz) to MIDI note number space
float powtodb ( float value ); convert signal power ratio to decibels (dB)
float rmstodb ( float value ); convert linear amplitude to decibels (dB)
float dbtopow ( float value ); convert decibels (dB) to signal power ratio
float dbtorms ( float value ); converts decibles (dB) to linear amplitude

plus maybe some other undocumented ones 😊

Some more Std. functions (System Power Tools)

int system ( string cmd ); pass a command to be executed in the shell
string getenv ( string key ); returns the value of an environment variable, such as of "PATH"
int setenv ( string key, string value ); sets environment variable named 'key' to 'value'
Machine Commands: Machine.add(“MyCode.ck”) => int myShred; Machine.remove(myShred);

int add ( string path );
  compile and spork a new shred from file at
  ‘path’ into the VM now, returns the shred ID

int spork ( string path );
  same as add

int remove ( int id );
  remove shred from VM by shred ID (returned by add/spork)

int replace ( int id, string path );
  replace shred with new shred from file

display current status of VM

int status ( );

void crash ( );
  literally causes the VM to crash. the very last resort;
  use with care. Thanks.

me Object:

me.id(); me.yield(); me.exit(); me.running(), me.clone(), me.done(), me.nargs(), me.arg();

ChucK Language Unit Generators:

global special unit generators:
  adc  dac  blackhole

standard ChucK unit generators:
  SinOsc  PulseOsc  SqrOsc  TriOsc  SawOsc  Phasor  Noise  Impulse  Step  Gain
  SndBuf  HalfRect  FullRect  ZeroX  Mix2  Pan2  GenX  CurveTable  WarpTable  LiSa

filters:
  Filter  OneZero  TwoZero  OnePole  TwoPole  PoleZero  BiQuad  LPF  HPF  BPF  BRF  ResonZ  Dyno

STK unit generators in ChucK:
  Envelope  ADSR  Delay  DelayL  Echo  JCRev  NRev  PRCRev  Chorus  Modulate  PitShift
  SubNoise  Blit  BlitSaw  BlitSquare  WvIn  WaveLoop  WvOut

STK instruments unit generators
  StkInstrument  BandedWG  BlowBotl  BlowHole  Bowed  Brass  Clarinet  Flute  Mandolin
  ModalBar  Moog  Saxofony  Shakers  Sitar  StilKarp  VoicForm
  FM  BeeThree  FMVoices  HevyMetl  PercFlut  Rhodey  TubeBell  Wurley

All UGs obey gain(float), op(int), last(), channels(), chan(int), most have other properties

// Simple FM Example
SinOsc modulator => ADSR menv => SinOsc carrier => ADSR cenv => dac;
2 => carrier.sync;  // setup carrier input for FM
cenv.set(0.01 :: second, 0.1 :: second, 0.5, 1.0 :: second);
menv.set(0.2 :: second, 0.2 :: second, 0.3, 1.0 :: second);
1000.0 => carrier.freq;  // roughly the spectral center
500.0 => modulator.freq;  // roughly the "pitch" (or inharmonic)
500.0 => modulator.gain;  // make this enough to do good modulation
1 => cenv.keyOn => menv.keyOn;  // spark this baby up!
0.2 :: second => now;  // let it get rolling
1 => cenv.keyOff => menv.keyOff;  // shut ‘er down
2.0 :: second => now;  // let it finish up
Objects/Classes

/* Define a new Class in one file */

public class FluteSweep {
    Flute f; // Flute physical model
    int sweeping;

    public void connect(UGen ugen) { f => ugen; } // connection

    public void blow(float freq) { // blow with float argument
        freq => f.freq;
        1 => f.noteOn => f.pressure;
        spork ~ sweepUp(); }

    public void blow(int note) { // overload/polymorph function
        blow(Std.mtof(note)); } // use existing mechanics

    public void shaddap() {
        1 => f.noteOff; // noteOff
        0 => sweeping; }

    private void sweepUp() {
        0.5 :: second => now; // let the note establish
        f.freq() => float temp; // low starting freq
        temp * 8.0 => float temp2; // ending frequency
        1.0 => float bl; // beginning blowing pressure
        1 => sweeping; // state variable for blowing
        while ((temp < temp2) & sweeping) {
            temp * 1.02 => temp => f.freq; // sweep freq up
            0.93 * bl => bl => f.noteOn; // ramp down blowing
            0.01 :: second => now;
        }
    }
}

/* Then you can use that class in any other subsequent file */

FluteSweep f; // make a new FluteSweep Object
f.connect(dac); // hook it up to audio out

[60, 62, 63, 65, 66, 67, 71, 72] @=> int notes[]; // CMaj scale

for (0 => int i; i < notes.cap(); i++) { // shorthand increment
    f.blow(notes[i]);
    0.9 :: second => now;
    f.shaddap();
    0.1 :: second => now;
}

Hook it to MIDI, Make an array of them, whatever you like!!
Events

Some asynchronous object => now;  (MIDI, OSC, MAUI Button, Slider, etc.)
  // we'll see that shortly

// signalEvent.ck : signaling events

Event e;            // declare an event

fun int hi( Event e ) {  // declare function that uses event
  <<< "OK, now what?" >>>;
  e => now;  // wait on event e
  <<< "success" >>>;  // only happens AFTER e is signaled
}

spork ~ hi( e );     // spork shred with e

1.0 :: second => now;  // advance time

e.signal();          // signal e (could also e.broadcast()

1.0 :: samp => now;   // hang around just one samp longer

See also the Conducting with Events programs

MIDI

0 => int device;  // device # to open (see: chuck --probe)
if( me.args() ) me.arg(0) => Stdatoi => device;  // get command line

MidiIn mdin;        // the midi event
MidiMsg msg;        // message for retrieving data

if( !mdin.open( device ) ) me.exit();  // open the device
<<< "MIDI device":, mdin.num(), "->", mdin.name() >>>;  // print device

SinOsc s => dac;    // our "synth"
0.0 => s.gain;

while( true ) {  // infinite time-loop
  mdin => now;   // wait on event 'mdin'
  while( mdin.recv(msg) ) {  // get message(s)
    <<< msg.data1, msg.data2, msg.data3 >>>;  // print message
    if (msg.data1 == 144) {  // if NoteOn msg
      Std.mtof(msg.data2) => s.freq;  // set freq
      msg.data3 / 127.0 => s.gain;  // Note On
    }
    else 0.0 => s.gain;  // Note Off (stupid version)
  }
}

Sonification in ChucK Workshop, ICAD 2012
OSC (Open Sound Control)

// Simple OSC Sender Example
"localhost" => string hostname; // who we're gonna talk to
6449 => int port; // port we're gonna talk on
OscSend xmit; // OSC Send Object
xmit.setHost( hostname, port ); // open and hook it up

while( true ) {
  xmit.startMsg( "/foo/notes", "i f" ); // start msg, int, float
  Std.rand2( 30, 80 ) => xmit.addInt; // fill in the int
  Std.rand2f( .1, .5 ) => xmit.addFloat; // fill in float
  0.2::second => now;
}

// Simple OSC Receiver Example
Rhodey ins => JCRRev rev => dac; // “synth”
.1 => rev.mix;

OscRecv recv; // make a receiver
6449 => recv.port; // on port number
recv.listen(); // start the receiver up
recv.event( "/foo/notes, i f" ) @=> OscEvent @ oe; // format message

int i;
float f;
while( true ) {
  oe => now; // wait for message to come in
  while( oe.nextMsg() ) { // peel off the arguments
    oe.getInt() => i => Std.mtof => ins.freq; // and use
    oe.getFloat() => f => ins.gain; // them for . . .
    f => ins.noteOn; // . . . music!!
    <<< "got (via OSC):", i, f >>>
  }
}

See iPhone Singer (TouchOSC App => ChucK Formant Voice) (note WiiOSC, other)
See Draw Homer (Processing Drawing => ChucK “Dope!” manipulation)
See Video Action (Processing Video => ChucK Synthesis) (note FaceOSC too)
Try to do a net-conducted ICADDTOrk “piece” (before or after break)

BREAK!!!!!!
HID, SMELT

Keyboard, Mouse, Trackpad, Joystick (various), Accelerometers, mic (adc)

// Use laptop shaking to control model of coin in mug
Shakers peso => JCrev rev => dac.right; // coin in coffee mug
0.03 => rev.mix;
19 => peso.preset; 10 => peso.objects; // setup parameters
1.0 => peso.decay; 1 => peso.noteOn; // and git this party started

Hid hi; // make a Hid object
HidMsg msg; // and holder for messages
if (!hi.openTiltSensor()) {
    <<"tilt sensor unavailable", "" >>>;
    me.exit();
}
<<"tilt sensor ready", "" >>>; // if success opening hid

float lacc[3]; // to hold our last accel values

while( true ) {
    hi.read( 9, 0, msg ); // read accel (device 9) number 0
    // << msg.x, msg.y, msg.z >>>; // (optional) print results
    (msg.x - lacc[0])*(msg.x - lacc[0]) => float shaking; // get total
    (msg.y - lacc[1])*(msg.y - lacc[1]) => shaking; // 3D squared
    (msg.z - lacc[2])*(msg.z - lacc[2]) => shaking; // velocity
    shaking / 4000.0 => peso.energy; // and use that to shake model
    msg.x => lacc[0]; msg.y => lacc[1]; msg.z => lacc[2]; // store last
    30 :: ms => now; // hang out until next read
}

Another example using “wind”

// Use wind on microphone to excite virtual bamboo wind chimes
Shakers bamboo => JCrev rev => dac.right; // bamboo wind chimes
5 => bamboo.preset; 1.0 => bamboo.decay; 4 => bamboo.objects;
adc => LPF lp => LPF lp2 => LPF lp3; // chain of low pass filters
50.0 => lp.freq => lp2.freq => lp3.freq; // set them up to pass
4.0 => lp.Q => lp2.Q => lp3.Q; // only low frequencies
lp3 => OnePole envFollow => blackhole; // envelope follower
0.999 => envFollow.pole; 200.0 => envFollow.gain; // wind detector
3 => envFollow.op; lp3 => envFollow; // square input
0.03 => rev.mix;

1 => bamboo.noteOn; // git this party started

while (1) {
    0.05 :: second => now; // update wind signal to
    envFollow.last() => bamboo.energy; // drive particle model
}

Also Demo Some Tether Controller Examples
UANAs, SMIRK

UpChucK!!! (vector buffer operations)

```plaintext
// FFT-based pitch shifting down by an octave.
adc => FFT fft ^= IFFT ifft => dac; // DSP Chain

1024 => fft.size => ifft.size; // Size
Windowing.hamming(512) => fft.window; // Window for smoothing
UAnaBlob blob; // Blob to hold data

while (1) {
    256 :: samp => now; // Advance time by hopsize
    fft.upchuck() @= blob; // Get data
    /**/ for (0 => int i; i < fft.size()/4; 1 += i) {
        /**/ blob.cvals()[i*2] => blob.cvals()[i]; // Copy spectrum
        /**/ ifft.upchuck(); // Inverse FFT and output
    }
}

Getting Features from Spectral Data

// Use FFT to track main peak in spectrum with a sine wave
adc => FFT fft => blackhole; // draw samples through FFT
SinOsc s => dac; // Our "synthesizer"

2048 => fft.size;
Windowing.hamming(1024) => fft.window;
UAnaBlob blob;

while (1) {
    512 :: samp => now; // Hop along by size/4
    fft.upchuck() @= blob; // Compute and
    blob.fvals() @= float mag_spec[]; // store spectrum
    0.0 => float peak;
    0.0 => float power;
    0 => int peakloc;
    for (0 => int i; i < fft.size()/2; 1 += i) {
        mag_spec[i]*mag_spec[i] += power; // Accumulate power
        if (mag_spec[i] > peak) {
            mag_spec[i] => peak;
            i => peakloc;
        }
    }
    44100.0 * peakloc / fft.size() => s.freq; // Set freq
    Math.sqrt(power) => s.gain; // Set gain
    // <<< s.freq() >>>;
}

Demo more FFT Examples, Features, 1NN Classifiers

Point to WEKINATOR
```
Multi-channel support

// Multi-channel output example
\texttt{\textbf{dac}.channels() =\text{ int numChans};} // number of available channels

\texttt{\textbf{SinOsc} s[\text{dac}.channels()];} // suitable sized array of sin oscs
\texttt{200.0 \Rightarrow float freq;} // base frequency variable

\begin{verbatim}
for (0 => \text{int i}; i < \text{numChans}; i++) \{ // iterate over all channels
freq => s[i].\text{freq}; // set sin osc frequency
s[i] => \text{\textbf{dac}.\text{chan}(i)}; // hook up to \	extbf{dac} channel
1.059 * freq => freq; // frequency up by half step
2.0 :: second => now; // hang out a bit
\}
\end{verbatim}

\begin{verbatim}
2.0 :: second => now; // hang out a bit
\end{verbatim}

File I/O

\texttt{\textbf{SndBuf, WvIn, WaveLoop, WvOut, FileIn, FileOut (NetIn, NetOut) (SerIn, SerOut soon)}}

\texttt{\textbf{\textbf{SndBuf} s => WvOut w \Rightarrow \textbf{\textbf{dac}};}} // make a \	extbf{sndbuf} and hook it to audio out
\texttt{"special:dope" => s.\text{read};} // this could be any valid .wav, other
\texttt{"test.wav" => w.\text{wavFilename};} // this opens a sound file for writing
\texttt{\textbf{\textbf{FileIO} log}; // make a data file object
log.open("LOG.txt", \textbf{\textbf{FileIO.WRITE}} ); // and open it for writing

\texttt{s.length()} => now; // let it play once

\texttt{-1.0 => s.\text{rate};} // set it to play backward
\texttt{(s.length()/samp) \Rightarrow \text{int} => s.\text{pos};} // set play position to end
\texttt{s.length()} => now; // let it play backward

\texttt{1 => s.\text{loop};} // set it to loop
\texttt{0 => \text{int} counter;

\texttt{now + 10.0 :: second => time then;}} // let this run for
\texttt{while (now < then) \{ // exactly 10 seconds
\texttt{Std.rand2f(0.0,s.length()/samp) \Rightarrow \text{int} => s.\text{pos};} // random position
\texttt{Std.rand2f(-2.0,2.0) => s.\text{rate};} // random rate
\texttt{Std.rand2f(0.1,1.0) :: second => now;}} // for random time
\texttt{1 += counter;
\}
\texttt{w.closeFile();} // clean up sound file

\texttt{log.write("We visited some part of Homer a total of ");} // write some
\texttt{log.write(counter);} // meaningful info into
\texttt{log.write(" times. \n");} // our log file
\texttt{log.close();} // and close it

Show this with some other .wav .aif files

Sonification in ChucK Workshop, ICAD 2012
Some More Sonification Examples:

- Desktop/Browser (cursor with regions)
- Google Stock Prices: Min, Max, Volume, Closing
- Light/Motion/Red/Blue in Lobby FM Bells?
- Joint-angle (finger) musification

Extending ChucK: ChuGenS, ChuGIns, and ChubGraphs

These are all Class definitions, but work as, or inherit from, UGen

Chugen \choo-jen\ (define your own UGs from within ChucK) example:

```chucK
class MyCosine extends Chugen {
    0 => int p;
    440 => float f;
    second/samp => float SRATE;

    fun float tick(float in) {
        return Math.cos(p++ * 2 * pi * f / SRATE);
    }
}
```

Chubgraph (define your UG, but must use only existing UGs) example:

```chucK
class Feedback extends Chubgraph {
    inlet => Gain dry => outlet;
    dry => Delay delay => outlet;
    delay => Gain feedback => delay;
    0.8 => feedback.gain;
    1 :: second => delay.delay;
}
```

```chucK
SinOsc s => Feedback f => dac;
```

```chucK
1.0 :: second => now;
300.0 => s.freq; 2.0 :: second => now;
1000.0 => s.freq; 1.0 :: second => now;
100.0 => s.freq; 3.0 :: second => now;
```

ChuGin \chug-in\ (define your own UGs in C/C++), outside our scope today

FaucK (Faust diagrams can generate ChucK ChuGins), outside our scope

Enough Already, Time to Hack ChucK!!!!!!! (in a second...
MAUI (MiniAudicle User Interface)
Mac Only (for now)

Sliders (Vertical and Horizontal)  Buttons (with images)  LEDs
Synthesis ToolKit in C++

STK is a set of classes in C++ which allow rapid experimentation with sound synthesis and processing. Available for free:

http://ccrma.stanford.edu/software/stk/

Ported and used a lot (Faust, mobile, ...)

“Unit Generators” the classical computer music/sound building blocks:

Oscillators, Filters, Delay Lines, etc.

STK Synthesis

PhiSM
Model
Samples
FM
Physical:
Plucked
Winds
Bowed

STK Unit Generators

Sources

Filters

Sinks

STK GUls in TCL / TK

Common simple controls for all algorithms

References and Resources

Book on interactive sound synthesis

ChuckK Book Coming (Soon-lish)

Book on all topics Sonification
ChucK is:  

- Ge Wang
- Perry Cook
- Phil Davidson
- Spencer Salazar
- Many others

SMELT is:  

- Rebecca Fiebrink
- Ge Wang

SMIRK is:  

- Rebecca Fiebrink
- Ge Wang

Wekinator is:  

- Rebecca Fiebrink
- Dan Trueman
- Perry Cook

chuck.cs.princeton.edu  
smelt.cs.princeton.edu  
smirk.cs.princeton.edu  
ccrma.stanford.edu/software/stk/
wekinator.cs.princeton.edu

(join the forums!)