Using OSC in ChucK: 2/27/08

To send OSC:

Decisions:

Host
Decide on a host to send the messages to. E.g., “splash.local” if sending to computer named “Splash,” or “localhost” to send to the same machine that is sending.

Port
Decide on a port to which the messages will be sent. This is an integer, like 1234.

Message “address”
For each type of message you’re sending, decide on a way to identify this type of message, formatted like a web URL.
E.g., “conductor/downbeat/beat1” or “Rebecca/message1”

Message contents
Decide on whether the message will contain data, which can be 0 or more ints, floats, strings, or any combination of them.

Code: For each sender:

Create an OscSend object:

OscSend xmit;

Set the host and port of this object:

xmit.setHost("localhost", 1234);

For every message, start the message by supplying the address and format of contents, where “f” stands for float, “i” stands for int, and “s” stands for string:

To send a message with no contents:

xmit.startMsg("conductor/downbeat");

To send a message with one integer:

xmit.startMsg("conductor/downbeat, i");

To send a message with a float, an int, and another float:

xmit.startMsg("conductor/downbeat, f, i, f");

For every piece of information in the contents of each message, add this information to the message:

E.g., to add an int: xmit.addInt(10);
   to add a float: xmit.addFloat(10.);
   to add a string: xmit.addString("abc");

Once all parts of the message have been added, the message will automatically be sent.
To receive OSC:

Decisions:
Port: decide what port to listen on. This must be the same as the port the sender is using.

Message address and format of contents: This must also be the same as what the sender is using; i.e., the same as in the sender’s startMsg function.

Code: for each receiver
Create an OscRecv object:
OscRecv orenc;

Tell the OscRecv object the port:
1234 => orenc.port;

Tell the OscRecv object to start listening for OSC messages on that port:
orenc.listen();

For each type of message, create an event that will be used to wait on that type of message, using the same argument as the sender’s startMsg function:
e.g.,
orenc.event("conductor/downbeat, i") @=> OscEvent
myDownbeat;

To wait on an OSC message that matches the message type used for a particular event e, do
e => now;
(just like waiting for regular Events in chuck)

To process the message:
Grab the message out of the queue (mandatory!)
e.nextMsg();

For every piece of information in the message, get the data out. You must call these functions in order, according to your formatting string used above.
e.getInt() => int i;
e.getFloat() => float f;
e.getString() => string s;

If you expect you may receive more than one message for an event at once, you should process every message waiting in the cue:
while (e.nextMsg() != 0) {
   //process message here (no need to call nextMsg again
}