Contents

Instr File Development Reference

Overview

Instr files are a simplified Csound orchestra. For those familiar with Csound, you will feel right at home.

All of the file handling, and external control for the Nebulae is done externally in Python, and then sent to an instance of Csound running the selected instrument file.

To have as much flexibility as possible, the main version of the Nebulae is a .instr file itself.

The instr file has two sections.

There's a small config section, where the user can set options for performance, and control handling. This section is denoted with a "nebconfigbegin" and "nebconfigend". It is explained in more detail below.

Following that is the instrument. You can create multiple Csound instruments, but by default instr 1 is needed, and will run for an infinite amount of time. (You can start/stop other instruments from instr 1)

Examples

Test IO

A very short example is a simple audio pass through. We've also illustrated the nebconfig, though there are default settings for these items and this could be skipped.

You'll also see the gksource value used to mute the audio. This will cause the source button to mute the audio.

```
nebconfigbegin
sr,48000
ksmps,64
-B,512
-b,128
nebconfigend

instr 1
ainl, ainr inch 1, 2
if gksource == 1 then
    ainl = 0
    ainr = 0
endif
outs ainl, ainr
endin
```

Handling Stereo Files

Here is a snippet that will load the right channel of all loaded audio files into their own tables. This snippet assumes a limit of 100 unique stereo files.

```
; generates right channel ftables for stereo files
gifilesel_offset = 399
gifile_right_offset = 599
itempidx = 0
loop:
    if gichn[itempidx] == 2 then
        giwoffset = itempidx + gifile_right_offset
     giwoffset ftgen (itempidx+gifile_right_offset), 0, 0, 1, gSname[itempidx], 0, 0, 2
    endif
    itempidx += 1
    if (itempidx < 100) igoto loop
```

Global List

All hardware controls, and loaded audio data are simply accessed as global variables (gkblend for example). The full list is detailed below.

Global Data from External Software

All of the following globals are set from external software.

Controls are named after their hardware control name.

- gilen[] - Array containing all file lengths
- gichn[] - Array containing the number of channels per file
- gSname[] - Array containing all of the names of the files.
- gisr[] - Array containing sample rate of the files.
- gipeak[] - Array containing peak amplitude of the files.
- ginumfiles - Number of files loaded into Csound.
- gkpitch - 0-1 value from pitch control. (Unipolar CV and encoder) (clicks to 0.6)
- gkspeed - 0-1 value from speed control. (Bipolar CV and encoder (clicks between 0.625, 0.375)
- gkloopstart - 0-1 value from start control
- gkloopsize - 0-1 value from size control
- gkdensity - 0-1 value from density control
- gkoverlap - 0-1 value from overlap control
- gkblend - 0-1 value from blend control
- gkwindow - 0-1 value from window control
- gkfilesel - index of table containing audio file data (unless set differently by alternate instrument).
- gkfreeze - freeze button/gate state.
- gkreset - reset button/gate state
- gksource - source button/gate state
- gkrecord - record button/gate state
- gkfilestate - state of the file button, independent of latching state and gate input.
- gkeol - communication channel to control the gate output state of the nebulae. (cleared in python on next control cycle)
- gksizestatus - value available for communication back to python. Used in main instr to disable EOL when rate is greater than the system can produce.
- gkrecordstatus - value available for communication back to python. Used in main instr to enable/disable recording UI from Csound.
- gksourcegate - state of the source gate input, independent of source button.
- gksourcebuttonstate - state of the source button, independent of source gate.

All 5 buttons, and 8 primary controls have an “_alt” variant that is updated from a secondary control menu. These are also available for use. The CV/Gate inputs will always affect the primary control, only the pots and encoders will affect the secondary controls.

The 5 Buttons and Gates can be configured with the config section

Config

You can include certain information that python will want for your instrument.

To configure something all you need is the following:

```
nebconfigbegin
ksmps, 128
freeze, triggered, rising
density_alt, 0.5
nebconfigend
```

for now here's what you can configure: - ksmps - sr - -B (Buffer Size) - -b (Period Size) - button mode (triggered, latching, momentary, incremental) - button edge sensitivity (rising, falling, state) - button maximum (for incremental you'll need a max value) - secondary control defaults (secondary buttons defaults cannot be reconfigured at this time)

Instr Tester

There is a Mac/PC/Linux friendly piece of software that can be used to test .instr files away from the Nebulae. It has a simple GUI, using sliders instead of knobs, but is functional, and useful for testing out instr files before running them on the hardware.

Pure Data Development Reference

The Nebulae has an installation of Pure Data Vanilla on it.

This means you can run existing or custom Pure Data patches on the Nebulae.

To run a PD instrument on the Nebulae:

At this time, you must load PD patches from any of the alternate instruments on the Nebulae.

- Use the instr selector menu to load one of the factory .instr files (any, except for the default will do).
- Access the instr selector menu by holding the Speed encoder
- Rotate the Speed encoder to the right, so that only the positive speed LED is illuminated green.
- You'll now see a visual representation of your PD patches loaded in alphabetical order displayed on the button LEDs below.
- Select the PD patch you'd like to load by pressing the corresponding LED button.
- Press the speed encoder to load the patch.

You can press the pitch encoder at anytime to exit the instr selector without loading a new instrument.

Controls

Here's a list of all usable controls in Pure Data and their expected ranges:

All of the secondary controls labelled, "alt" are accessed while holding down the source button.

- speed: 0-1 value of encoder + CV amount. Clicks between 0.625 and 0.375
- pitch: 0-1 value of encoder + 1V/Oct amount. Clicks to 0.6
- start: 0-1
- size: 0-1
- density: 0-1
- overlap: 0-1
- window: 0-1
- reset: 10ms 1 value sent when pressed or triggered.
- freeze: latching 0 or 1 control. Button is rising edge sensitive
- record: latching 0 or 1 control. Button is rising edge sensitive
- file: latching 0 or 1 control. Button is falling edge sensitive
- source: latching 0 or 1 control. Button is falling edge sensitive
- filestate: state of the file button independant of gate. 0 or 1
- sourcegate: state of the source gate input independant of button. 0 or 1
- pitch_alt: 0-1
- start_alt: 0-1
- size_alt: 0-1
- density_alt: 0-1
- overlap_alt: 0-1

- window_alt: 0-1
- reset_alt: 10ms 1 value sent when button is pressed in secondary control menu
- freeze_alt: latching 0 or 1 control. Button is rising edge sensitive
- record_alt: latching 0 or 1 control. Button is falling edge sensitive
- file_alt: latching 0 or 1 control. Button is falling edge sensitive

Creating a patch that uses the controls

Gaining access to the controls is as easy as adding a netreceive object, pointed at Port 3000

For example, if we wanted to use the density and overlap in our new patch, we would setup the following small chain.

[netreceive 3000] -> [route density overlap]

Connecting an [s “controlname”] to each outlet of route, will add a named channel in Pure Data that can be used any where in your patch with [r “controlname”]

Using Audio Input and Output

For Audio I/O, the dac~ and adc~ objects work just fine, and simplify the process of testing the patch on a computer.

Loading Files

Loading files in Pure Data is different than with the .instr file.

All audio files from the USB drive are copied to the /home/alarm/audio directory at bootup.

Reading the audio file from Pure Data can be done using the readsf~ object.

Writing Audio files and Capturing Buffers

writesf~ can be used to write a soundfile

tabwrite~ can be used to write to an array that’s loaded in RAM.

Reading of recorded arrays can be done with tabread~ and tabosc~ objects

Testing on your Personal Computer

Aside from the physical controls being received via TCP, the entirety of a patch can be tested away from the Nebulae.

The simplest approach to testing the range of your controls is to use sliders or number objects that sum with the named control before being scaled and sent to the destination. This allows the user to test adjusting the control without having to maintain a separate local version of the patch.

Planned Improvements

There is currently no way to control the pulse output on the Nebulae from Puredata.

With .instr files, the user can use a nebconfig section in the beginning of their code to tweak the default values, and set up the behavior for each button. This is currently not available within Pure Data

We'll be adding a "Pure Data" mode to the Instr-Tester, allowing you to run your Pure Data patch on a desktop simulation of the Nebulae hardware.