SYNOPSIS

airsh :: AIR shell, a simple air-coremix.pd control shell to demonstrate OSC parameters to control the AIR core mixer | B-AIR project Creative Europe 2020-2023

PUREDATA gpp-air-coremixer ELEMENT

OSC-controlled puredata core mixer for AIR platform:

[gpp-air-coremixer.pd <num.input.channels> <OSC.inaddress> <OSC.outaddress>]

Air core mixer automatically connects to the puredata input [adc~] and output [dac~]. Mixer output is stereo, while number of the input channels can be user-defined (as a fixed param 1 of the pd element). The params 2 and 3 are OSC input/output ports.

AIRMIX :: AIR CORE MIXER OSC COMMAND PROTOCOL

CHANNEL ADDRESSING

The structure of the OSC message is:

/airmix/<channel>/<controller> argument(s)

channel = master (in short: m) or <channel> / <set of channels> - see COMPLEX CHANNEL ADDRESSING

MONITORING

<ch> mon / <ch> nom

local per-channel monitoring

mix / xim

show whole mixer-monitor in a pd window

con / noc

inform client (console) to show or hide console gui window

CONTROLLERS

master|m fader|f|level <airliner_vector>

send an airliner_vector to a master fader (both channels)

master|m lfader|rfader|left|right <airliner_vector>

send an airliner_ector to a master fader (separate channels)

<ch> fader|f <airliner_vector>

send an airliner_ector to a separate channel's faders

<ch> pan|p <airliner_vector>

send an airliner_vector to a separate channel pan

* NOTE master has no pan but both faders /stereo/ instead

AIRLINER_VECTOR

is managed by gpp-airliner~.pm. Airliner can be bound to any mixer param, currently it controls faders and panning

it will accept 1 - 4 input parameters:

<value_to_reach [rms 0-1]> <time_of_operation [ms]>? <curve [string - see below]>? <initial delay [ms]>?

where only the first parameter, <value_to_reach [rms 0-1]>, is compulsory. The other three, if omitted, will default to:

<time_of_operation [ms]> defaults to 'jump' micro-fader time (50 ms)

<curve [see]> defaults to 'lin' or 'linear'

<initial delay [ms]> defaults to 0

Currently available curves:

jump micro-fader (50ms) linear curve

lin linear

sin sinusoidal (soft)

hsin half-sinusoidal (equal power)

log logarithmic, hard neck

pow power (exponential), slow sudden attack

<airliner_vector> examples

0.9 => jump to 0.9

0.422 7500 => slide to 0.422 in 7.5 seconds

1 4230 sin => slide to 1 in 4.23 seconds following sinusoidal curve

0.5 7600 pow 2000 => slide to .5 in 7.6 seconds following power curve, but start after 2 seconds from now

AUDIO PRODUCTION / PROCESSING TOOLS

PLAY

<ch> play <file>

Play a file (wav,aiff)

<ch> play <file> <hh:mm:ss.uuu>

Play a file from the selected timing on

<ch> play <file> <hh:mm:ss.uuu> <hh:mm:ss.uuu>

Play a file from the selected timing to the selected end

<ch> play <file> <hh:mm:ss.uuu> <hh:mm:ss.uuu> +<hh:mm:ss.uuu>

Play a file from the selected timing to the selected end with the selected +delay. Note that the player / looper delay parameter should

be marked with +

<ch> stop

Stop current playing session at the addressed track

LOOP

<ch> loop <file>

Loop whole file

<ch> loop <file> <hh:mm:ss.uuu> <hh:mm:ss.uuu>

Loop file clip from selected timing to selected timing

<ch> loop <file> <hh:mm:ss.uuu> <hh:mm:ss.uuu> +<hh:mm:ss.uuu>

Loop file clip from selected timing to selected timing with intermediate +delay

<ch> stop

Stop current looping session at the addressed track

RECORD

<ch> record <file>

Record to file

<ch> record <file> <hh:mm:ss.uuu>

Record to file after initial +delay

<ch> record <file> <hh:mm:ss.uuu> <hh:mm:ss.uuu>

Record to file after initial lead-in till selected end. The second figure being absolute time, not a time interval. Timing display will change color to red when the recording actually starts.

<ch> rstop

Stop current recording session at the addressed track

COMPLEX CHANNEL ADDRESSING

<ch> can be defined as:

channel number : send to a selected channel 1-n

all : send to all channels

even : send to even-numbered channels

odd : send to odd-numbered channels

selection span : send to a selection span which consist from '-' delimited ranges and ',' delimited lists, for instance 1,3,5-7

GLOBAL AND PER-TRACK ALIASES

ch solo : set the channel 'solo'

ch mute | m : mute the channel (monitoring: /airmon/etc/mute 1)

ch unmute | um: mute the channel (monitoring: /airmon/etc/mute 0)

master ml | mr : mute left master or right master channel

master uml | umr : mute left master or right master channel

ch prelisten | pl: add the channel to the pre-listening bus (separate prelisten monitoring audio port) (monitoring: /airmon/etc/prelisten 1)

ch premute | pm: remove the channel from the pre-listening bus (separate prelisten monitoring audio port) (monitoring: /airmon/etc/prelisten 0)

ch presolo | ps: set the channel as pre-listening solo (monitoring: /airmon/etc/presolo)

panic : all track faders set to 0

dir | cd : set global wavedata directory

ls : get global wavedata directory

trimfader : set trimfader time or trimfader type [*not yet implemented]

centerpan : center all pans

fullout : set all faders to full

stereopan : set even-numbered faders to 0 and odd-numbered faders to 1 (receiving in full stereo mode) [*not yet implemented]

globepan : evenly distribute panning across panorama 0-1 accross all channels

setff : set server's fanning factor (distance between serial OSC monitoring request, somewhere between 2 and 6, default 3) :: the serial monitoring requests (fader, pan) get distributed after the following formula:

setmode : set server's monitoring mode (auto, mirror, coarse). See 'Two main server-side monitoring modes' below.

fader : (<ch> * <FF> * 2) msec DELAY

pan : (<ch> * <FF>) msec DELAY

INTERNAL CLIENT- COMMANDS

These are client- (console) initiated

/airmix/cinit/ bang => client signalizing server their own loadbang and querying for initial parameters such as number of channels, airdir etc. This request is handled separately on air-coremixer input.

available as a separate jack output bus. operated with:

ch prelisten | pl: add the channel to the pre-listening bus (separate prelisten monitoring audio port) (monitoring: /airmon/etc/prelisten 1)

ch premute | pm: remove the channel from the pre-listening bus (separate prelisten monitoring audio port) (monitoring: /airmon/etc/prelisten 0)

ch presolo | ps: set the channel as pre-listening solo (monitoring: /airmon/etc/presolo)

prelevel | prl : set the level of prelistening bus. Bound to 'jump' (50ms) ramp, no other functionality.

AIRMON :: AIR CORE MONITORING OSC COMMAND PROTOCOL

MONITORING TACTICS

Serves dispatching information about the mixer state to all subordinated (remote) web client subjects. While the data, dispatched on momentary basis, does not represent a problem regarding the OSC traffic demands, all continuous data types, such as fader pan etc, represent a challenge. Air engine will internally handle such types by the usage of signal-level streaming, therefore monitoring can be done using 3 different tactics:

A request : whole request (for instance: fader .5 3000 sin 6000) is transfered (copied from the airmix OSC input) to the web clients and clients themselves will take care about rendering itself, from begining to end. Not only gpp-airliner directive, but also timer requests can be given this way.

B snapshot : current state of controller transferred on periodical basis (metronome pulse). Intermediate request data is sent using relatively low frequency in order to avoid OSC wire overheads

C contour : this tactics is actually a modification of snapshot tactics, but will only send snapshot to the client(s) when RF (request_fulfilled) signal on the respected controller is broadcast (this is called VR (value_reached), also provided by gpp-airliner~). So instead of metronome pulse, the initiative for broadcasting snapshot is based on reaching controllers' border states. All intermediate request data is ignored. This tactics alone can be used for low bandwidth cases, but also to complement tactics A and B. With the A tactics, it can help assure that the final state is obeyed regardless of potential server/client desync. With the B tactics it can assure that the border state information is delivered exactly in time, even if the server's snapshot heartbeat is low.

OSC CONTROLLER MAP

MONITORING TACTICS:

A. REQUEST

/airmon/request/<ch>/<controller> <request params>

any kind of user-request as such, copied directly from the OSC input for each respective controller, such as:

/airmon/request/<ch>/fader <airliner~ params>

/airmon/request/<ch>/pan <airliner~ params>

_

A2. VIRTUAL REQUEST

A request assembled by the server engine (not passed by the user) in order to encapsule bandwidth-demanding movement to be rendered by the client engine, such as:

/airmon/request/<ch>/timer/<type> [start|stop] [initial_timing_ms]?

display timer, where <type> can be "play" or "record"

_

B. SNAPSHOT

/airmon/snapshot/<ch>/<controller>

controller state at the very moment

C. CONTOUR

/airmon/contour/<ch>/...

separate reporting of basic border states :: the point of this tactics is to report some basic params that the client (web or pd) can use to render their own movement. There are several 'contour handlers', such as:

... rf/<controller> => controller rf when reached (request fulfilled => reached destination boolean (1=yes 0=not yet), fader or pan

... vr/<controller> => controller vr when rf reached (value reached => the actual destination value triggered when reached, fader or pan, and also play, record (border states)

... ac/<controller> => controller ac (active) : play, record and such things => player is active boolean (1=yes, 0=not)

*note difference between rf and ac switches: rf policy is negative (= positive when not active) while ac policy is positive (= negative when not active).

ALL THE REST

All data that cannot be ranged into the 3 main monitoring tactics, will be passed via etc

/airmon/etc/<anyparam> => other data or switches, non policy-classified

/airmon/etc/<ch>/<anyparam> => other data or switches, non policy-classified : per-track

for instance:

/airmon/etc/<ch>/filename/ => current file name distributed exactly before each play- or record- request

/airmon/etc/<ch>/mute => mute boolean for the selected channel

/airmon/etc/master/mute => master mute boolean (both channels)

/airmon/etc/master/rmute => master mute boolean (right channel only)

/airmon/etc/master/lmute => master mute boolean (left channel only)

DIRECT SYS PARAMS

System params (internal communication between air-coremix and console, without user interaction) are sent via /sys/ and /syscore/ controllers. /sys/ will deliver the messaga to the console frontend (shell that envelopes console-core) and /syscore/ will to the console core itself. These params is no thing the end user should be bothered with.

/airmon/sys/CHNUM/ => number of channels, required by gpp-air-coremixer creation argument - the console automatically resizes if the argument changes.

/airmon/syscore/AIRDIR/ => airdir path storage (path itself stored as text)

/airmon/syscore/SHOW/ [boolean] => show / hide console

TWO MAIN SERVER-SIDE MONITORING MODES BASED ON DIFFERENT MONITORING TACTICS

MIRROR and COARSE :: implementing contour and snapshot tactics:

MIRROR : use both tactics => high bandwidth and low client capacity; will report border states and intermediate snapshots according to network capacity (tunable). Contour skeleton is used to prevent possible timing-based anomalies, especially with extremely lo frequency monitoring heartbeat.

COARSE : use only contour tactic => low bandwidth and low client capacity; will only report border (crucial) states.

AUTO :: use request and snapshot tactics => client themselves will render events; this is only safe when client is capable of running puredata console. Suitable for very low web bandwidth. Contour skeleton is used to prevent possible timing-based anomalies.

Server can be set to either mode in terms of communicating with its console(s):

/airmix/setmode mirror or airsh setmode mirror :: set mirroring mode (contour + snapshot)

/airmix/setmode coarse or airsh setmode coarse :: set mirroring mode with only contour tactics enabled

/airmix/setmode auto or airsh setmode auto :: set autonomous mode (all rendering based on request tactics data vectors)

/airmix/lsmode :: list currently selected monitoring mode

AIRSH EXAMPLES

airsh m fader 1 2000 sin => fade complex curve

airsh 1 fader .3 4000 log 2000 => fade after delay

airsh play file.wav 0:12.345 0:15.012 +1:0.300 => play a clip after delay

airsh loop file.wav 0:12.345 0:15.012 +1:0.300 => loop a clip after delay

airsh rec 0:10 0:15 => record from (abs) time 10 to 15 seconds (5 sec long block)

MON OSC PROTOCOL EXAMPLES

```
/airmon/request/master/lfader/ 1 3000 sin

/airmon/request/master/rfader/ 1 3000 sin

/airmon/request/master/fader/ 1 3000 sin

/airmon/snapshot/1/fader .346567

/airmon/contour/master/rf/lfader 1

/airmon/contour/master/rf/lfader 1

/airmon/contour/3/ac/player 1

/airmon/contour/1/timer/player start 3543 down

/airmon/contour/1/timer/player stop

/airmon/contour/3/timer/recorder start 0 up
```

AUTHOR

Gregor Pirs (c)2022 <gregor.pirs@guest.arnes.si>