SoundServer

The sound server reads a YAML file specifying sounds to be played. Sounds can then be triggered and manipulated remotely over OSC (e.g. fading in and out tracks).

OSC commands

The server responds to OSC messages. The following messages are supported:

- **/sound_server/spawn** *(sound_name (string), spawn_name(string))* Create a new transient sound. The sound with name sound_name is created. It will be assigned the name spawn_name, so that you can subsequently adjust its gain, position etc. spawn_names should be unique. **pools**: you may specify a *pool* name instead of a sound name; this will choose a random sound from the pool and spawn it.

Only transient sounds should be spawned; (singleton) background layers are already spawned at server launch and can be started and stopped subsequently.

In the following *sound_name* can be either the name of a sound or of a channel group.

- **/sound_server/start** *(sound_name (string))* Starts a sound. **All sounds are initialised in the stopped state -- they must be started before anything will be heard.** Any initial parameters of a sound (e.g. gain, position) should be set *before* calling */sound_server/start*.
- **/sound_server/stop** (sound_name (string)) Stops/pauses a sound. Can be resumed with */sound_server/start*.
- **/sound_server/gain** *(sound_name (string), dB (float), [time (float)])* Sets the gain of the specified sound in decibels (e.g. -6.0 means half the volume). *time* is optional -- if not given, the gain is set immediately; otherwise the sound fades from its current gain to the set gain in *time* seconds.
- **/sound_server/position** *(sound_name (string), x (float), y (float), z (float), [time(float)])* Sets the 3D position of the sound. Optional argument *time* behaves as */sound_server/gain*.
- **/sound_server/filter** *(sound_name (string), Hz (float), [time(float)])* Sets the lowpass filtering of a sound. Note that channel groups always have a filter active, but bare sounds only have one if it was specified in the YAML configuration (if you try and set a sound without a filter, nothing will happen). Optional argument *time* behaves as */sound_server/gain*.
- **/sound_server/mute** *(sound_name (string))* Mute the given sound
- **/sound_server/unmute** *(sound_name (string))* Unmute the given sound
- **/sound_server/reverb** *(reverb_name (string))* Set the current reverb scene to the specified name.
- **/sound_server/eq** *(eq_name (string))* Set the current EQ scene to the

specified name.
- **/sound_server/burst_enable** *(burst_name (string))* Enable the given random burst sound.
- **/sound_server/burst_disable** *(burst_name (string))* Disable the given random burst sound.
- **/sound_server/listener/position** *(x (float), y (float), z (float)* Set the position of the 3D listener
- **/sound_server/listener/fwd** *(x (float), y (float), z (float)* Set the forward vector of the 3D listener
- **/sound_server/listener/up** *(x (float), y (float), z (float)* Set the up vector of the 3D listener
- **/sound_server/automation/attach** *(sound_name (string), automation (string)* Attach and activate the given automation to the given sound.
- **/sound_server/automation/detach** *(sound_name (string), automation (string)* Stop and dettach the given automation from the given sound.
- **/sound_server/shutdown** Exit the server loop and fade out.

YAML file format

The YAML file consists of a number of sections:

- *config* Global configuration
- *channel_groups* Mixer channel groups
- *sounds* Individual sounds
- *bursts* Stochastic event emitters
- *automations* Automatic adjustments to gain/position/filtering (run on the server)
- *pools* Collections of sounds to randomly select from
- *listener* Configuration of the 3D listener
- *eqs* Master equaliser configuration
- *reverbs* Master reverbs configuration

Config

config The configuration block sets global parameters for the server. Valid parameters are:

- *channels* total number of mixer channels to allocate (e.g. 96)
- *base_path* base path to prepend to all sound filenames (e.g. sounds/)
- *update_rate* rate of the soundserver update loop, in Hz (e.g. 100.0)
- *dsp_jitter* timing jitter added to sounds to avoid machine gun effects. Should be roughly equal to 1/update_rate (e.g. 0.01)
- *ip_address* IP address to listen on for the OSC server (e.g. 127.0.0.1 for localhost)
- *port* Port used for OSC server (e.g. 8000)
- *audio_device* part of the name of an audio device to use. The first audio device with a name string which contains audio_device will be used. If omitted, the first audio device will be used.
- *buffer_size* length of one audio buffer in samples. Longer may be more stable, but increases latency

- *n_buffers* number of buffers of length *buffer_size*. Total latency = (*n_buffers* x *buffer_size*) / sample_rate
- *speaker_mode* the speaker mode to use. Can be one of: *stereo, mono, quad, surround, 5.1, 7.1, srs5.1, dolby5.1*
- *speaker_location* the speaker location block. Each speaker can be one of: *front_left, front_center, front_right, low_frequency, back_left, back_right, side_left, side_right* stereo uses *front_center* and *front_right*. mono uses *front_left* only. Each speaker can be enabled and the x,y spatial position in the range [-1, 1]

- *spatial_scale* scaling of the 3D units used. The default space is -10000 -> 10000 (e.g. 0.01 would make a world space from [-100, 100])

  Example:

```
  speaker_location:
      front_left:
          x: -1
          y: 1
          enabled: True
      front_right:
          x: 1
          y:1
          # enabled is default True
```

Example:

```
config:
    channels: 96
    base_path: sounds
    dsp_jitter: 0.02
    update_rate: 100.0
    ip_address: 127.0.0.1
    port: 8000
    audio_device: M-Audio
```

Channel Groups

Each of the channel groups is a mixer channel. Multiple sounds can be mixed on a channel group. Each channel group has a gain, and can be filtered. Transient sounds *must* be assigned to a channel group; the number of channels reserved for that groups transient sounds needs to be specified in the channel group specifcation. Channel groups can be attached to other channel groups, to form a hierarchy of mixers. *channel_group* valid parameters

- *name* Name of the channel group
- *transient_channels* Number of transient sub channels to allocate to this group
- *gain* initial gain of the channel group in dB (default: 0.0)
- *mute* initial mute state (default: False)
- *filter* initial lowpass filter state, in Hz (default: 30000 (disabled))
- *subgroups* a list of channel group names to attach as subgroups of this channel

Example:

```
channel_groups:
    -   name: background_layers
        subgroups: [drone_layer]

    -   name: drone_layer
        gain: -20.0
        filter: 1000

    -   name: bubble_sounds
        transient_channels: 16
        gain: 0.0
```

Sounds

Each sound has at least a name and filename. A sound can be transient (played on demand) or not (played as background layer, for the whole duration of the server). The 3D position of the sound can be specified, along with the gain, a low-pass filter, and whether or not the sound loops. min_distance specifies the "scale" of the sound in world terms. A sound at exactly min_distance away has no gain attenuation at that level. The channel group this sound belongs to can be specified, which is mandatory for transient sounds

Valid parameters for a sound:

- *name* name of the sound (should be unique)
- *file* filename of the wavefile (combined with base_path to find the actual file) (e.g. bloops.wav)
- *min_distance* minimum distance at which sound is at full volume (e.g. 1000.0)
- *position* 3D position of sound (default. [0,0,0])
- *channel_group* the channel_group mixer to play on (e.g. background_layer)
- *gain* the initial gain, in db (default: 0.0)
- *filter* filter frequency, in Hz. Note that if this sound has a filter specified in the YAML, */sound_server/filter* will be able to adjust its frequency; otherwise no filter will be attached to this sound and */sound_server/filter* messages directed at this sound will be ignored. Use 30000 to create a filter initially off. (default: None)
- *loop* whether or not to loop (default: False)
- *transient* whether or not this is a transient sound (triggered multiple times) or a background sound layer. Transient sounds **must** have a channel_group specified, and that channel_group must have transient channels allocated.

Example:

```
sounds:
    -   name: touch_beep
        min_distance: 60
        transient: True
        position: [0,0,0]
```

```
        file: pop-sound.wav
        channel_group: touch_sounds


    -   name: clicky
        min_distance: 10000
        gain: -18
        loop: True
        position: [0,0,0]
        file: layer_clicky.wav
        channel_group: background_layers
```

Pools

Pools are groups of transient sounds that can be randomly selected on a spawn request. This is useful for triggering sounds without being excessively repetitive. */sound_server/spawn/<pool_name>* will choose one of the sounds from *<pool_name>* and spawn it. Each pool has a name, and a list of sounds to trigger. All sounds in a pool must be transient. Valid parameters:

- *name* Name of this pool
- *sounds* List of sound names in this pool.

Example:

```
pools:
    -   name: release
        sounds:
            -   release_bad
            -   release_jazz
            -   release_good
```

Listener

Listener sets the initial position and orientation of the 3D listener (the audio 'camera'). Orientation is given as a an up, forward vector pair. Valid parameters are:

- *position* 3D position, as a list
- *forward* 3D forward vector, as a list. Should be normalised.
- *up* 3D forward vector, as a list. Should be normalised.

Example:

```
listener:
    position:   [0,0,0]
    forward:    [0,0,1]
    up:         [0,1,0]
```

eqs

The eqs section lists all of the equaliser scenes available. Each equaliser is a

list of bands, specifying the frequencies and gain to set. Valid parameters are:

- *name* Name of the eq scene. If the name is *default*, then this eq will be activated on startup.
- *bands* List of frequency bands. Each band is a list [frequency, octaves, gain], where *frequency* is in Hz, *octaves* is the width of the band in octaves, and *gain* is the gain in dB. *gain* should be in the range [-30, 6.0].

Example:

```
eqs:
    name: flat
    bands:
        - [100.0,1.0, 0.0]
        - [200.0,1.0, 0.0]
        - [400.0,1.0, 0.0]
        - [800.0,1.0, 0.0]
        - [1600.0,1.0, 0.0]
        - [3200.0,1.0, 0.0]
        - [6400.0,1.0, 0.0]
        - [12800.0,1.0, 0.0]

    name: low_only
        bands:
        - [50.0,1.0, 0.0]
        - [100.0,1.0, 0.0]
        - [200.0,1.0, 0.0]
        - [400.0,1.0, 0.0]
        - [800.0,1.0, 0.0]
        - [1600.0,1.0, -15.0]
        - [3200.0,1.0, -20.0]
        - [6400.0,1.0, -25.0]
        - [12800.0,1.0, -30.0]
```

Automations

Automations specify automatic adjustments to position, gain or filtering of sounds or channel groups. Once activated, these continuously modulate the assigned variables.

For example, an automation can be used to whirl a sound round in circles, or randomly fade in or fade out a sound. Each automation has a name, a type (random, sine or spline), an attribute (gain, filter or position) and a specification for the specific type used.

Any automation can also have a time block, which specifies a sub-automation which will control the time rate of the original automation. e.g. a spline automation can be used to speed up and slow down a sine automation. The sub-automation must be one dimensional.

Valid parameters are:

- *name* name of the automation

- *type* type of the automation. Can be *sine*, *spline* or *random*
- *attr* attribute to modify. Can be *gain*, *filter_val*, or *position*
- *time* A sub-automation that controls the rate of this automation.

For *sine* automations, there should be a *sine* block with the following parameters:

- *min* minimum value of the sine wave
- *max* maximum value of the sine wave
- *frequency* frequency of the modulation, in Hz
- *phase* (optional), phase offset, in radians (default: 0.0)

For *random* automations, there should be a *random* block with the following parameters:

- *range* range of the value
- *rate* rate of change, in units/seconds

For *spline* automations, there should be a *spline* block with the following parameters:

- *rate* rate of playback, in seconds per point
- *points* a list of points on the spline
- *loop* (optional) whether or not to loop the spline, or just run it once (default: False)
- *tension* (optional) tension of the spline (TCB spline) (default: -0.5)
- *bias* (optional) bias of the spline (default: 0.0)
- *continuity* (optional) continuity of the spline (default: 0.0)

Example:

```
automations:
    - name: fade
      type: random
      attr: gain
      random:
        range: [-18, 0]
        rate: 20.0

    - name: flutter
      type: sine
      attr: gain
      sine:
        frequency: 5.0
        min: -20
        max: 0
     # note the use of the time block to vary the rate
      time:
        type: spline
        spline:
            points: [0.25, 0.5, 1.0, 2.0, 3.0, 4.0]
            rate: 0.5
            loop: True

    - name: spline
```

```
      type: spline
      attr: gain
      spline:
        points: [0.0, -20.0, -40.0, -60.0, 0, 6.0]
        rate: 2.0
        loop: False
```

Bursts

Bursts are sounds which can be automatically triggered by the server, on a random schedule. This is useful for things like bubbles, firework sounds, bird noises, etc. Each burst specifies a random rate at which to operate, a gain range, and a 3D position box for the sounds to go in.

Each burst has two modes: A and B, each of which can have different random rates, gains and position. This is useful to acheive bursty sounds (e.g. a rush of bubbles followed by a few occasional bubbles then a rush, etc.)

The switching between mode A and mode B is governed by a simple Markov chain, which has a probability of switching into A from B and from B to A. If burstiness is not required, the parameters for mode A and B can be the same.

Valid parameters:

- *name* Name of the burst.
- *pool* Sound pool to trigger from. Only pools can be triggered by bursts -- you can add a single sound to a pool if required.
- *switching* The mode A->B and B->A probabilities, as a list. The probabilities are in in probablity of switching per second. e.g. [0.1, 0.9] makes a mode which transitions to B every 10 seconds or so, and back to B within less than a second.
- *space* The 3D space in which to trigger a sound, as a bounding box [[x1, y1, z1], [x2, y2, z2]]. e.g. [[0,0,0], [100, 100, 100]]
- *states* Two state elements
  - *rate* Rate of generation, in events/second. e.g. 2.0
  - *gain* gain range of events in dB, as a pair [min, max]. e.g. [-20, 0]

Example:

```
bursts:
-   name: bubbles
    pool: bubbles
    switching: [0.1, 0.3]
    space: [[0,0,0], [0,0,0]]
    states:
        -   rate: 2.0
            gain: [-10, 0]
        -   rate: 2.0
            gain: [-10, 0]
```

Reverbs

The reverb settings. The reverb with name 'default' will be loaded and

enabled at start up, if it exists. The reverb settings are parameters for the FMODEx REVERBSFX DSP object -- refer to the FMODEx documentation for details of their effect.

Example (listing all valid parameters):

```
  - name: default
    dry: 0
    room: -20
    room_hf: 0.0
    room_rolloff: 10.0
    room_decaytime: 10
    room_decay_hfratio: 0.5
    reflections_level: -100
    reflections_delay: 0.02
    reverb_level: 0.0
    reverb_delay: 0.04
    diffusion: 100.0
    density: 100.0
    hf_reference: 5000.0
    room_lf: 0.0
    lf_reference: 250.0
```