Audio System (aud)

Audaspace (pronounced “outer space”) is a high level audio library.

Basic Sound Playback

This script shows how to use the classes: `Device`, `Sound` and `Handle`.

```
import aud

device = aud.Device()
# load sound file (it can be a video file with audio)
sound = aud.Sound('music.ogg')

# play the audio, this return a handle to control play/pause
handle = device.play(sound)
# if the audio is not too big and will be used often you can buffer it
sound_buffered = aud.Sound.cache(sound)
handle_buffered = device.play(sound_buffered)

# stop the sounds (otherwise they play until their ends)
handle.stop()
handle_buffered.stop()
```

aud.**AP_LOCATION**

Constant value 3

aud.**AP_ORIENTATION**

Constant value 4

aud.**AP_PANNING**

Constant value 1

aud.**AP_PITCH**

Constant value 2

aud.**AP_VOLUME**

Constant value 0

aud.**CHANNELS_INVALID**

Constant value 0

aud.**CHANNELS_MONO**

Constant value 1

aud.**CHANNELS_STEREO**

Constant value 2

aud.**CHANNELS_STEREO_LFE**

Constant value 3

aud.**CHANNELS_SURROUND4**

Constant value 4

aud.**CHANNELS_SURROUND5**

Constant value 5

aud.**CHANNELS_SURROUND51**

Constant value 6

aud.**CHANNELS_SURROUND61**

Constant value 7

aud.**CHANNELS_SURROUND71**

Constant value 8

aud.**CODEC_AAC**

Constant value 1

aud.**CODEC_AC3**

Constant value 2

aud.**CODEC_FLAC**

Constant value 3

aud.**CODEC_INVALID**

Constant value 0

aud.**CODEC_MP2**

Constant value 4

aud.**CODEC_MP3**

Constant value 5

aud.**CODEC_OPUS**

Constant value 8

aud.**CODEC_PCM**

Constant value 6

aud.**CODEC_VORBIS**

Constant value 7

aud.**CONTAINER_AAC**

Constant value 8

aud.**CONTAINER_AC3**

Constant value 1

aud.**CONTAINER_FLAC**

Constant value 2

aud.**CONTAINER_INVALID**

Constant value 0

aud.**CONTAINER_MATROSKA**

Constant value 3

aud.**CONTAINER_MP2**

Constant value 4

aud.**CONTAINER_MP3**

Constant value 5

aud.**CONTAINER_OGG**

Constant value 6

aud.**CONTAINER_WAV**

Constant value 7

aud.**DISTANCE_MODEL_EXPONENT**

Constant value 5

aud.**DISTANCE_MODEL_EXPONENT_CLAMPED**

Constant value 6

aud.**DISTANCE_MODEL_INVALID**

Constant value 0

aud.**DISTANCE_MODEL_INVERSE**

Constant value 1

aud.**DISTANCE_MODEL_INVERSE_CLAMPED**

Constant value 2

aud.**DISTANCE_MODEL_LINEAR**

Constant value 3

aud.**DISTANCE_MODEL_LINEAR_CLAMPED**

Constant value 4

aud.**FORMAT_FLOAT32**

Constant value 36

aud.**FORMAT_FLOAT64**

Constant value 40

aud.**FORMAT_INVALID**

Constant value 0

aud.**FORMAT_S16**

Constant value 18

aud.**FORMAT_S24**

Constant value 19

aud.**FORMAT_S32**

Constant value 20

aud.**FORMAT_U8**

Constant value 1

aud.**RATE_11025**

Constant value 11025

aud.**RATE_16000**

Constant value 16000

aud.**RATE_192000**

Constant value 192000

aud.**RATE_22050**

Constant value 22050

aud.**RATE_32000**

Constant value 32000

aud.**RATE_44100**

Constant value 44100

aud.**RATE_48000**

Constant value 48000

aud.**RATE_8000**

Constant value 8000

aud.**RATE_88200**

Constant value 88200

aud.**RATE_96000**

Constant value 96000

aud.**RATE_INVALID**

Constant value 0

aud.**STATUS_INVALID**

Constant value 0

aud.**STATUS_PAUSED**

Constant value 2

aud.**STATUS_PLAYING**

Constant value 1

aud.**STATUS_STOPPED**

Constant value 3

class aud.**Device**

Device objects represent an audio output backend like OpenAL or SDL, but might also represent a file output or RAM buffer output.

lock()

Locks the device so that it’s guaranteed, that no samples are read from the streams until `unlock()` is called. This is useful if you want to d
start/stop/pause/resume some sounds at the same time.

Note

The device has to be unlocked as often as locked to be able to continue playback.

Warning

Make sure the time between locking and unlocking is as short as possible to avoid clicks.

play(sound, keep=False)

Plays a sound.

PARAMETERS:

- **sound** (`Sound`) – The sound to play.
- **keep** (*bool*) – See `Handle.keep` .

RETURNS:

The playback handle with which playback can be controlled with.

RETURN TYPE:

`Handle`

stopAll()

Stops all playing and paused sounds.

unlock()

Unlocks the device after a lock call, see `lock()` for details.

channels

The channel count of the device.

distance_model

The distance model of the device.

See also

OpenAL Documentation

doppler_factor

The doppler factor of the device. This factor is a scaling factor for the velocity vectors in doppler calculation. So a value bigger than 1 will exaggerate the effect as it raises the velocity.

format

The native sample format of the device.

listener_location

The listeners's location in 3D space, a 3D tuple of floats.

listener_orientation

The listener's orientation in 3D space as quaternion, a 4 float tuple.

listener_velocity

The listener's velocity in 3D space, a 3D tuple of floats.

rate

The sampling rate of the device in Hz.

speed_of_sound

The speed of sound of the device. The speed of sound in air is typically 343.3 m/s.

volume

The overall volume of the device.

class aud.**DynamicMusic**

The DynamicMusic object allows to play music depending on a current scene, scene changes are managed by the class, with the possibility of custo transitions. The default transition is a crossfade effect, and the default scene is silent and has id 0

addScene(scene)

Adds a new scene.

PARAMETERS:

scene (`Sound`) – The scene sound.

RETURNS:

The new scene id.

RETURN TYPE:

int

addTransition(ini, end, transition)

Adds a new scene.

PARAMETERS:

- **ini** (*int*) – the initial scene foor the transition.
- **end** (*int*) – The final scene for the transition.
- **transition** (`Sound`) – The transition sound.

RETURNS:

false if the ini or end scenes don't exist, true othrwise.

RETURN TYPE:

bool

pause()

Pauses playback of the scene.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

resume()

Resumes playback of the scene.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

stop()

Stops playback of the scene.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

fadeTime

The length in seconds of the crossfade transition

position

The playback position of the scene in seconds.

scene

The current scene

status

Whether the scene is playing, paused or stopped (=invalid).

volume

The volume of the scene.

class aud.**HRTF**

An HRTF object represents a set of head related transfer functions as impulse responses. It's used for binaural sound

loadLeftHrtfSet(extension, directory)

Loads all HRTFs from a directory.

PARAMETERS:

- **extension** (*string*) – The file extension of the hrtfs.
- **directory** – The path to where the HRTF files are located.

RETURNS:

The loaded `HRTF` object.

RETURN TYPE:

`HRTF`

loadRightHrtfSet(extension, directory)

Loads all HRTFs from a directory.

PARAMETERS:

- **extension** (*string*) – The file extension of the hrtfs.
- **directory** – The path to where the HRTF files are located.

RETURNS:

The loaded `HRTF` object.

RETURN TYPE:

`HRTF`

addImpulseResponseFromSound(sound, azimuth, elevation)

Adds a new hrtf to the HRTF object

PARAMETERS:

- **sound** (`Sound`) – The sound that contains the hrtf.
- **azimuth** (*float*) – The azimuth angle of the hrtf.
- **elevation** (*float*) – The elevation angle of the hrtf.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

class aud.**Handle**

Handle objects are playback handles that can be used to control playback of a sound. If a sound is played back multiple times then there are as ma handles.

pause()

Pauses playback.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

resume()

Resumes playback.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

stop()

Stops playback.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

Note

This makes the handle invalid.

attenuation

This factor is used for distance based attenuation of the source.

See also

`Device.distance_model`

cone_angle_inner

The opening angle of the inner cone of the source. If the cone values of a source are set there are two (audible) cones with the apex at the `location` of the source and with infinite height, heading in the direction of the source's `orientation`. In the inner cone the volume i normal. Outside the outer cone the volume will be `cone_volume_outer` and in the area between the volume will be interpolated linear

cone_angle_outer

The opening angle of the outer cone of the source.

See also

`cone_angle_inner`

cone_volume_outer

The volume outside the outer cone of the source.

See also

`cone_angle_inner`

distance_maximum

The maximum distance of the source. If the listener is further away the source volume will be 0.

> See also
>
> `Device.distance_model`

distance_reference

The reference distance of the source. At this distance the volume will be exactly `volume` .

> See also
>
> `Device.distance_model`

keep

Whether the sound should be kept paused in the device when its end is reached. This can be used to seek the sound to some position and star
playback again.

> Warning
>
> If this is set to true and you forget stopping this equals a memory leak as the handle exists until the device is destroyed.

location

The source's location in 3D space, a 3D tuple of floats.

loop_count

The (remaining) loop count of the sound. A negative value indicates infinity.

orientation

The source's orientation in 3D space as quaternion, a 4 float tuple.

pitch

The pitch of the sound.

position

The playback position of the sound in seconds.

relative

Whether the source's location, velocity and orientation is relative or absolute to the listener.

status

Whether the sound is playing, paused or stopped (=invalid).

velocity

The source's velocity in 3D space, a 3D tuple of floats.

volume

The volume of the sound.

volume_maximum

The maximum volume of the source.

> See also
>
> `Device.distance_model`

volume_minimum

The minimum volume of the source.

> See also
>
> `Device.distance_model`

class aud.**ImpulseResponse**

An ImpulseResponse object represents a filter with which to convolve a sound.

class aud.**PlaybackManager**

A PlabackManager object allows to easily control groups os sounds organized in categories.

addCategory(volume)

Adds a category with a custom volume.

PARAMETERS:

volume (*float*) – The volume for ther new category.

RETURNS:

The key of the new category.

RETURN TYPE:

int

clean()

Cleans all the invalid and finished sound from the playback manager.

getVolume(catKey)

Retrieves the volume of a category.

PARAMETERS:

catKey (*int*) – the key of the category.

RETURNS:

The volume of the cateogry.

RETURN TYPE:

float

pause(catKey)

Pauses playback of the category.

PARAMETERS:

catKey (*int*) – the key of the category.

RETURNS:

Whether the action succeeded.

RETURN TYPE:

bool

play(sound, catKey)

Plays a sound through the playback manager and assigns it to a category.

PARAMETERS:

- **sound** (`Sound`) – The sound to play.
- **catKey** (*int*) – the key of the category in which the sound will be added, if it doesn't exist, a new one will be created.

RETURNS:

The playback handle with which playback can be controlled with.

The playback handle with which playback can be controlled with.

RETURN TYPE:
`Handle`

resume(catKey)

Resumes playback of the catgory.

PARAMETERS:
catKey (*int*) – the key of the category.

RETURNS:
Whether the action succeeded.

RETURN TYPE:
bool

setVolume(volume, catKey)

Changes the volume of a category.

PARAMETERS:

- **volume** (*float*) – the new volume value.
- **catKey** (*int*) – the key of the category.

RETURNS:
Whether the action succeeded.

RETURN TYPE:
int

stop(catKey)

Stops playback of the category.

PARAMETERS:
catKey (*int*) – the key of the category.

RETURNS:
Whether the action succeeded.

RETURN TYPE:
bool

class aud.**Sequence**

This sound represents sequenced entries to play a sound sequence.

add()

Adds a new entry to the sequence.

PARAMETERS:

- **sound** (`Sound`) – The sound this entry should play.
- **begin** (*double*) – The start time.
- **end** (*double*) – The end time or a negative value if determined by the sound.
- **skip** (*double*) – How much seconds should be skipped at the beginning.

RETURNS:
The entry added.

RETURN TYPE:
`SequenceEntry`

remove()

Removes an entry from the sequence.

PARAMETERS:

entry (`SequenceEntry`) – The entry to remove.

setAnimationData()

Writes animation data to a sequence.

PARAMETERS:

- **type** (*int*) – The type of animation data.
- **frame** (*int*) – The frame this data is for.
- **data** (*sequence of float*) – The data to write.
- **animated** (*bool*) – Whether the attribute is animated.

channels

The channel count of the sequence.

distance_model

The distance model of the sequence.

See also

OpenAL Documentation

doppler_factor

The doppler factor of the sequence. This factor is a scaling factor for the velocity vectors in doppler calculation. So a value bigger than 1 will exaggerate the effect as it raises the velocity.

fps

The listeners's location in 3D space, a 3D tuple of floats.

muted

Whether the whole sequence is muted.

rate

The sampling rate of the sequence in Hz.

speed_of_sound

The speed of sound of the sequence. The speed of sound in air is typically 343.3 m/s.

class aud.**SequenceEntry**

SequenceEntry objects represent an entry of a sequenced sound.

move()

Moves the entry.

PARAMETERS:

- **begin** (*double*) – The new start time.
- **end** (*double*) – The new end time or a negative value if unknown.
- **skip** (*double*) – How many seconds to skip at the beginning.

setAnimationData()

Writes animation data to a sequenced entry.

PARAMETERS:

- **type** (*int*) – The type of animation data.
- **frame** (*int*) – The frame this data is for.
- **data** (*sequence of float*) – The data to write.
- **animated** (*bool*) – Whether the attribute is animated.

attenuation

This factor is used for distance based attenuation of the source.

See also

`Device.distance_model`

cone_angle_inner

The opening angle of the inner cone of the source. If the cone values of a source are set there are two (audible) cones with the apex at the `location` of the source and with infinite height, heading in the direction of the source's `orientation`. In the inner cone the volume i normal. Outside the outer cone the volume will be `cone_volume_outer` and in the area between the volume will be interpolated linear

cone_angle_outer

The opening angle of the outer cone of the source.

See also

`cone_angle_inner`

cone_volume_outer

The volume outside the outer cone of the source.

See also

`cone_angle_inner`

distance_maximum

The maximum distance of the source. If the listener is further away the source volume will be 0.

See also

`Device.distance_model`

distance_reference

The reference distance of the source. At this distance the volume will be exactly `volume`.

See also

`Device.distance_model`

muted

Whether the entry is muted.

relative

Whether the source's location, velocity and orientation is relative or absolute to the listener.

sound

The sound the entry is representing and will be played in the sequence.

volume_maximum

The maximum volume of the source.

See also

`Device.distance_model`

volume_minimum

The minimum volume of the source.

See also

`Device.distance_model`

class aud.**Sound**

Sound objects are immutable and represent a sound that can be played simultaneously multiple times. They are called factories because they create reader objects internally that are used for playback.

classmethod buffer(data, rate)

Creates a sound from a data buffer.

PARAMETERS:

- **data** (`numpy.ndarray`) – The data as two dimensional numpy array.
- **rate** (*double*) – The sample rate.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

classmethod file(filename)

Creates a sound object of a sound file.

PARAMETERS:

filename (*string*) – Path of the file.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Warning

If the file doesn’t exist or can’t be read you will not get an exception immediately, but when you try to start playback of that sound.

classmethod list()

Creates an empty sound list that can contain several sounds.

PARAMETERS:

random (*int*) – whether the playback will be random or not.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

classmethod sawtooth(frequency, rate=48000)

Creates a sawtooth sound which plays a sawtooth wave.

PARAMETERS:

- **frequency** (*float*) – The frequency of the sawtooth wave in Hz.
- **rate** (*int*) – The sampling rate in Hz. It's recommended to set this value to the playback device's samling rate to avoid resamping.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

classmethod silence(rate=48000)

Creates a silence sound which plays simple silence.

PARAMETERS:

rate (*int*) – The sampling rate in Hz. It's recommended to set this value to the playback device's samling rate to avoid resamping.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

classmethod sine(frequency, rate=48000)

Creates a sine sound which plays a sine wave.

PARAMETERS:

- **frequency** (*float*) – The frequency of the sine wave in Hz.
- **rate** (*int*) – The sampling rate in Hz. It's recommended to set this value to the playback device's samling rate to avoid resamping.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

classmethod square(frequency, rate=48000)

Creates a square sound which plays a square wave.

PARAMETERS:

- **frequency** (*float*) – The frequency of the square wave in Hz.
- **rate** (*int*) – The sampling rate in Hz. It's recommended to set this value to the playback device's samling rate to avoid resamping.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

classmethod triangle(frequency, rate=48000)

Creates a triangle sound which plays a triangle wave.

PARAMETERS:

- **frequency** (*float*) – The frequency of the triangle wave in Hz.
- **rate** (*int*) – The sampling rate in Hz. It's recommended to set this value to the playback device's samling rate to avoid resamping.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

ADSR(attack, decay, sustain, release)

ADSR(attack, decay, sustain, release)

Attack-Decay-Sustain-Release envelopes the volume of a sound. Note: there is currently no way to trigger the release with this API.

PARAMETERS:

- **attack** (*float*) – The attack time in seconds.
- **decay** (*float*) – The decay time in seconds.
- **sustain** (*float*) – The sustain level.
- **release** (*float*) – The release level.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

accumulate(additive=False)

Accumulates a sound by summing over positive input differences thus generating a monotonic sigal. If additivity is set to true negative input differences get added too, but positive ones with a factor of two.

Note that with additivity the signal is not monotonic anymore.

PARAMETERS:

additive – Whether the accumulation should be additive or not.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

addSound(sound)

Adds a new sound to a sound list.

PARAMETERS:

sound (`Sound`) – The sound that will be added to the list.

Note

You can only add a sound to a sound list.

binaural()

Creates a binaural sound using another sound as source. The original sound must be mono

PARAMETERS:

- **hrtfs** – An HRTF set.
- **source** (`Source`) – An object representing the source position of the sound.
- **threadPool** (`ThreadPool`) – A thread pool used to parallelize convolution.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

cache()

Caches a sound into RAM.

This saves CPU usage needed for decoding and file access if the underlying sound reads from a file on the harddisk, but it consumes a lot of memory.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Note

Only known-length factories can be buffered.

Warning

Raw PCM data needs a lot of space, only buffer short factories.

convolver()

Creates a sound that will apply convolution to another sound.

PARAMETERS:

- **impulseResponse** (`ImpulseResponse`) – The filter with which convolve the sound.
- **threadPool** (`ThreadPool`) – A thread pool used to parallelize convolution.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

data()

Retrieves the data of the sound as numpy array.

RETURNS:

A two dimensional numpy float array.

RETURN TYPE:

`numpy.ndarray`

Note

Best efficiency with cached sounds.

delay(time)

Delays by playing adding silence in front of the other sound's data.

PARAMETERS:

time (*float*) – How many seconds of silence should be added before the sound.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

envelope(attack, release, threshold, arthreshold)

Delays by playing adding silence in front of the other sound's data.

PARAMETERS:

- **attack** (*float*) – The attack factor.
- **release** (*float*) – The release factor.
- **threshold** (*float*) – The general threshold value.
- **arthreshold** (*float*) – The attack/release threshold value.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

fadein(start, length)

Fades a sound in by raising the volume linearly in the given time interval.

PARAMETERS:

- **start** (*float*) – Time in seconds when the fading should start.
- **length** (*float*) – Time in seconds how long the fading should last.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Note

Before the fade starts it plays silence.

fadeout(start, length)

Fades a sound in by lowering the volume linearly in the given time interval.

PARAMETERS:

- **start** (*float*) – Time in seconds when the fading should start.
- **length** (*float*) – Time in seconds how long the fading should last.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Note

After the fade this sound plays silence, so that the length of the sound is not altered.

filter(b, a=(1,))

Filters a sound with the supplied IIR filter coefficients. Without the second parameter you'll get a FIR filter.

If the first value of the a sequence is 0, it will be set to 1 automatically. If the first value of the a sequence is neither 0 nor 1, all filter coefficients will be scaled by this value so that it is 1 in the end, you don't have to scale yourself.

PARAMETERS:

- **b** (*sequence of float*) – The nominator filter coefficients.
- **a** (*sequence of float*) – The denominator filter coefficients.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

highpass(frequency, Q=0.5)

Creates a second order highpass filter based on the transfer function \(H(s) = s^2 / (s^2 + s/Q + 1)\)

PARAMETERS:

- **frequency** (*float*) – The cut off trequency of the highpass.
- **Q** (*float*) – Q factor of the lowpass.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

join(sound)

Plays two factories in sequence.

PARAMETERS:

sound (`Sound`) – The sound to play second.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

> Note
>
> The two factories have to have the same specifications (channels and samplerate).

limit(start, end)

Limits a sound within a specific start and end time.

PARAMETERS:

- **start** (*float*) – Start time in seconds.
- **end** (*float*) – End time in seconds.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

loop(count)

Loops a sound.

PARAMETERS:

count (*integer*) – How often the sound should be looped. Negative values mean endlessly.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

> Note
>
> This is a filter function, you might consider using `Handle.loop_count` instead.

lowpass(frequency, Q=0.5)

Creates a second order lowpass filter based on the transfer function \(H(s) = 1 / (s^2 + s/Q + 1)\)

PARAMETERS:

- **frequency** (*float*) – The cut off trequency of the lowpass.
- **Q** (*float*) – Q factor of the lowpass.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

mix(sound)

Mixes two factories.

PARAMETERS:

sound (`Sound`) – The sound to mix over the other.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

> Note
>
> The two factories have to have the same specifications (channels and samplerate).

modulate(sound)

Modulates two factories.

PARAMETERS:

sound (`Sound`) – The sound to modulate over the other.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

> Note
>
> The two factories have to have the same specifications (channels and samplerate).

mutable()

Creates a sound that will be restarted when sought backwards. If the original sound is a sound list, the playing sound can change.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

pingpong()

Plays a sound forward and then backward. This is like joining a sound with its reverse.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

pitch(factor)

Changes the pitch of a sound with a specific factor.

PARAMETERS:

factor (*float*) – The factor to change the pitch with.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Note

This is done by changing the sample rate of the underlying sound, which has to be an integer, so the factor value rounded and the factor may not be 100 % accurate.

Note

This is a filter function, you might consider using `Handle.pitch` instead.

rechannel(channels)

Rechannels the sound.

PARAMETERS:

channels (*int*) – The new channel configuration.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

resample(rate, quality)

Resamples the sound.

PARAMETERS:

- **rate** (*double*) – The new sample rate.
- **quality** (*int*) – Resampler performance vs quality choice (0=fastest, 3=slowest).

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

reverse()

Plays a sound reversed.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Note

The sound has to have a finite length and has to be seekable. It's recommended to use this only with factories with fast and accurate seeking, which is not true for encoded audio files, such ones should be buffered using `cache()` before being played reversed.

Warning

If seeking is not accurate in the underlying sound you'll likely hear skips/jumps/cracks.

sum()

Sums the samples of a sound.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

threshold(threshold=0)

Makes a threshold wave out of an audio wave by setting all samples with a amplitude >= threshold to 1, all <= -threshold to -1 and all betwe
to 0.

PARAMETERS:

threshold (*float*) – Threshold value over which an amplitude counts non-zero.

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

volume(volume)

Changes the volume of a sound.

PARAMETERS:

volume (*float*) – The new volume..

RETURNS:

The created `Sound` object.

RETURN TYPE:

`Sound`

Note

Should be in the range [0, 1] to avoid clipping.

Note

This is a filter function, you might consider using `Handle.volume` instead.

write(filename, rate, channels, format, container, codec, bitrate, buffersize)

Writes the sound to a file.

PARAMETERS:

- **filename** (*string*) – The path to write to.
- **rate** (*int*) – The sample rate to write with.
- **channels** (*int*) – The number of channels to write with.
- **format** (*int*) – The sample format to write with.
- **container** (*int*) – The container format for the file.
- **codec** (*int*) – The codec to use in the file.
- **bitrate** (*int*) – The bitrate to write with.
- **buffersize** (*int*) – The size of the writing buffer.

length

The sample specification of the sound as a tuple with rate and channel count.

specs

The sample specification of the sound as a tuple with rate and channel count.

class aud.**Source**

The source object represents the source position of a binaural sound.

azimuth

The azimuth angle.

distance

The distance value. 0 is min, 1 is max.

elevation

The elevation angle.

class aud.**ThreadPool**

A ThreadPool is used to parallelize convolution efficiently.

class aud.**error**