

---

# Sound<sub>lib</sub>

***Release 0.8***

**Jan 20, 2023**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Introduction . . . . .	3
1.2	Installation . . . . .	4
1.3	Quickstart . . . . .	4
<b>2</b>	<b>The API Documentation / Guide</b>	<b>7</b>
2.1	Developer Interface . . . . .	7
<b>3</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Python Module Index</b>	<b>27</b>
	<b>Index</b>	<b>29</b>



**Sound\_lib** is a higher-level, python wrapper around [the Bass audio library](#).



This portion of the documentation will serve to help one get acquainted with the library and its features.

## 1.1 Introduction

Sound\_lib is a library which simplifies working with audio.

It is primarily a wrapper over the excellent Bass sound library, and includes a majority of Bass's add-ons.

### 1.1.1 Features

- Cross-platform, tested on Microsoft Windows, MacOS, and Linux
- Support for playback of an incredible variety of audio formats
- Support for recording from a microphone or other input device
- Support for reencoding audio to a variety of formats
- Support for several audio effects
- Support for streaming files from disk, memory, or the internet
- Small yet efficient
- Well-documented

### 1.1.2 Getting Help

If you're hung up on a certain function, class, or method it might be helpful to take a look at the comprehensive bass documentation directly (distributed with sound\_lib but also obtainable online).

Bass also has a highly active [forum](#). When interacting there, it's important to remember that most of the good folks you'll encounter have never used and have no idea what `sound_lib` is or does. Thus, it's in your best interest to either describe the problem in layman's terms or point to actual bass functions.

If you're willing and able, do have a look at the portion of `sound_lib` that happens to be giving you trouble. It's quite possible that you might be facing a simple misunderstanding with the way things work. The force is with those who read the code after all.

If all else fails, you may have found a bug in `sound_lib`. Please do let us know at our [github issue tracker](#). Thanks in advance for contributions!

## 1.2 Installation

Install from PyPI:

```
pip install sound_lib
```

Install from source:

```
git clone git://github.com/accessible_apps/sound_lib.git
cd sound_lib
pip install .
```

## 1.3 Quickstart

Here we give a brief overview of the many features `sound_lib` has to offer. It should be at least enough to get up and running.

### 1.3.1 Playing

To load audio from a filename on the system:

```
>>> from sound_lib import stream
>>> from sound_lib import output
>>> o = output.Output()
>>> s = stream.FileStream(file = "music.ogg")
```

Or to load from a URL:

```
>>> from sound_lib import stream
>>> from sound_lib import output
>>> o = output.Output()
>>> s = stream.URLStream(url = "some_site.com:8000/stream.mp3")
```

All streams inherit from a channel. In these examples, `s` is a stream meaning you can call any `sound_lib.channel.Channel` method.

```
>>> s.play()
>>> s.stop()
>>> s.pan = 1.0 # Set balance all the way to the right
>>> s.play_blocking()
```



In these examples, `o` is the `sound_lib.output.Output` object. One must be created when initializing a channel for playback. Failing to do so will raise an exception. Output objects are like channels, all be it with a few major differences. For example, they always effect playback attributes at a global level, independent of settings given to individual channels. For instance:

```
>>> o.volume = 0.2
```

Will set the volume of all output channels to 0.2.

note: It's recommended to manage attributes of channels independently. Feel free to initialize output and never touch it again.

note: Attributes on effected channels aren't updated accordingly. This has the potential to be confusing.

```
>>> s.volume = 1.0
>>> o.volume=0.5
>>> s.volume
1.0
>>> o.volume
0.5
```

If working with raw audio data, you can create a `sound_lib.stream.PushStream`, which will play everything it receives in realtime. A more practical example can be found in the pulling it all together section below. Here we're assuming you have audio to feed the stream.

```
>>> from sound_lib import output
>>> o=output.Output()
>>> p=stream.PushStream()
>>> p.play() #makes it so data is automatically played when added to the stream.
>>> p.push(data) #should start playing data.
```

## 1.3.2 Recording

To record to a wave file.

```
>>> from sound_lib import input
>>> from sound_lib import recording
>>> i = input.Input()
>>> rec = recording.WaveRecording(filename = "test.wav")
>>> rec.play() #starts recording
>>> rec.stop()
```

A `sound_lib.input.Input` object follows the same general principal as `sound_lib.output.Output`. That is, everything is done at a global level. See above for notes.

You can easily subclass `sound_lib.recording.Recording` if wishing to do something else with input. See the pulling it all together section.

## 1.3.3 Pulling it altogether

In order to combine recording and playing, the following snippet will act as an audio test. It will take input from the default input device and send it to the default output device. This is mostly useful when verifying functionality of an audio setup in a voice chat application.

```
import ctypes
from sound_lib import input
from sound_lib import output
from sound_lib import recording
from sound_lib import stream

def record_callback(handle, buffer, length, user):
    """A callback that will receive and process recorded sample data"""
    buffer=ctypes.string_at(buffer, length)
    #Buffer could just as easily be sent to a file or over the network after_
    ↪additional processing.
    push.push(buffer)
    return True #To keep recording. False stops.

o=output.Output()
i=input.Input()
push=stream.PushStream(chans=2)
push.play()
rec=recording.Recording(proc=record_callback)
rec.play_blocking()
```

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

## 2.1 Developer Interface

### 2.1.1 *sound\_lib.channel*

**class** `sound_lib.channel.Channel` (*handle*)

A “channel” can be a sample playback channel (HCHANNEL), a sample stream (HSTREAM), a MOD music (HMUSIC), or a recording (HRECORD). Chances are, if you’re playing audio, you’re using a channel on one level or another. Each “Channel” function can be used with one or more of these channel types.

**play** (*restart=False*)

Starts (or resumes) playback of a sample, stream, MOD music, or recording.

**Parameters** **restart** (*bool*) – Specifies whether playback position should be thrown to the beginning of the stream. Defaults to False

**Returns** True on success, False otherwise.

**Return type** bool

**play\_blocking** (*restart=False*)

Starts (or resumes) playback, waiting to return until reaching the end of the stream

**Parameters** **restart** (*bool*) – Specifies whether playback position should be thrown to the beginning of the stream. Defaults to False.

**pause** ()

Pauses a sample, stream, MOD music, or recording.

**Returns** True on success, False otherwise.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If this channel isn't currently playing, already paused, or is a decoding channel and thus not playable.

**is\_active()**

Checks if a sample, stream, or MOD music is active (playing) or stalled. Can also check if a recording is in progress.

**is\_playing**

Checks whether the stream is currently playing or recording.

**Returns** True if playing, False otherwise.

**Return type** bool

**is\_paused**

Checks whether the stream is currently paused.

**Returns** True if paused, False otherwise.

**Return type** bool

**is\_stopped**

Checks whether the stream is currently stopped.

**Returns** True if stopped, False otherwise.

**Return type** bool

**is\_stalled**

Checks whether playback of a stream has been stalled. This is due to a lack of sample data. Playback will automatically resume once there is sufficient data to do so.

**Returns** True if stalled, False otherwise.

**Return type** bool

**get\_position(mode=0)**

Retrieves the playback position of a sample, stream, or MOD music. Can also be used with a recording channel.

**Parameters** `mode (str)` – How to retrieve the position. Defaults to “byte”.

**Returns** The current position.

**Return type** int

**Raises** `sound_lib.main.BassError` – If the requested position is not available.

**set\_position(pos, mode=0)**

Sets the playback position of a sample, MOD music, or stream.

**Parameters**

- **pos (int)** – The position, in units determined by the mode.
- **mode (str)** – How to set the position. Defaults to “byte”.

**Returns** True if the position was set, False otherwise.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If the stream is not a `sound_lib.stream.FileStream` or the requested position/mode is not available.

**position**

Retrieves the playback position of a sample, stream, or MOD music. Can also be used with a recording channel.

**Parameters** `mode` (*str*) – How to retrieve the position. Defaults to “byte”.

**Returns** The current position.

**Return type** `int`

**Raises** `sound_lib.main.BassError` – If the requested position is not available.

**stop** ()

Stops a sample, stream, MOD music, or recording.

**update** (*length=0*)

Updates the playback buffer of a stream or MOD music.

**Parameters** `length` (*int*) – The amount of data to render, in milliseconds... 0 = default (2 x update period). This is capped at the space available in the buffer.

**Returns** True on success, False otherwise.

**Return type** `bool`

**Raises** `sound_lib.main.BassError` – If this channel has ended or doesn’t have an output -buffer.

**get\_length** (*mode=0*)

Retrieves the playback length of this channel.

**Parameters** `mode` – How to retrieve the length. Can take either a flag attribute (string) or bass constant (int). Defaults to “byte”.

**Returns** The channel length on success, -1 on failure.

**Return type** `int`

**Raises** `sound_lib.main.BassError` – If the requested mode is not available.

**get\_device** ()

Retrieves the device in use by this channel.

**Returns** The device number, -1 on failure.

**Return type** `int`

**set\_device** (*device*)

Changes the device in use by this channel. Must be a stream, MOD music or sample.

**Parameters** `device` – The device to use... 0 = no sound, 1 = first real output device, BASS\_NODEVICE = no device.

**Returns** True on success, False otherwise.

**Return type** `bool`

**Raises** `sound_lib.main.BassError` – If device is invalid, device hasn’t been initialized, this channel is already using the requested device, the sample format is not supported by the device/drivers or there is insufficient memory.

**device**

Retrieves the device in use by this channel.

**Returns** The device number, -1 on failure.

**Return type** `int`

**set\_fx** (*type, priority=0*)

Sets an effect on a stream, MOD music, or recording channel.

**Parameters**

- **type** – The type of effect
- **priority** – The priority of the new FX, which determines its position in the DSP chain. DSP/FX with higher priority are applied before those with lower. This parameter has no effect with DX8 effects when the “with FX flag” DX8 effect implementation is used. Defaults to 0.

**Returns** A handle to the new effect on success, False otherwise.

**Raises** `sound_lib.main.BassError` – If type is invalid, the specified DX8 effect is unavailable or this channel’s format is not supported by the effect.

**bytes\_to\_seconds** (*position=None*)

Translates a byte position into time (seconds), based on the format in use by this channel.

**Parameters** **position** – The position to translate. Defaults to None

**Returns** The translated length on success, a negative bass error code on failure.

**Return type** int

**length\_in\_seconds** ()

Retrieves the length of the stream, in seconds, regardless of position.

**Returns** The length on success, a negative bass error code on failure.

**Return type** int

**seconds\_to\_bytes** (*position*)

Translates a time (seconds) position into bytes, based on the format in use by this channel.

**Parameters** **position** – The position to translate.

**Returns** The translated length on success, a negative bass error code on failure.

**Return type** int

**get\_attribute** (*attribute*)

Retrieves the value of this channel’s attribute.

**Parameters** **attribute** – The attribute to get the value of. Can either be an of type str or int.

**Returns** The value on success, None on failure.

**Raises** `sound_lib.main.BassError` – If the attribute is either unavailable or invalid. Some attributes have additional possible instances where an exception might be raised.

**set\_attribute** (*attribute, value*)

Sets the value of an attribute on this channel.

**Parameters**

- **attribute** – The attribute to set the value of. Can either be of type str or int.
- **value** –

**Returns** True on success, False on failure.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If either attribute or value is invalid.

**slide\_attribute** (*attribute, value, time*)

Slides this channel’s attribute from its current value to a new value.

**Parameters**

- **attribute** – The attribute to slide the value of.
- **value** – The new attribute value. Consult specific documentation depending on the one in question.
- **time** – The length of time (in milliseconds) that it should take for the attribute to reach the value.

**Returns** True on success, False on failure.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If attribute is invalid, or the attributes value is set to go from positive to negative or vice versa when the BASS\_SLIDE\_LOG flag is used.

**is\_sliding** (*attribute=0*)

Checks if an attribute (or any attribute) of this channel is sliding. Must be a sample, stream, or MOD music.

**Parameters** **attribute** – The attribute to check for sliding, or 0 for any. Defaults to 0.

**Returns** True if sliding, False otherwise.

**Return type** bool

**get\_info** ()

Retrieves information on this channel.

**Returns** A BASS\_CHANNELINFO structure.

**get\_level** ()

Retrieves the level (peak amplitude) of a stream, MOD music or recording channel.

**Returns**

**-1 on error. If successful, the level of the left channel is returned in the low word (low 16 bits), and the level of the right channel is returned in the high word (high 16 bits).**  
If the channel is mono, then the low word is duplicated in the high word. The level ranges linearly from 0 (silent) to 32768 (max). 0 will be returned when a channel is stalled.

**Return type** int

**Raises** `sound_lib.main.BassError` – If this channel is not playing, or this is a decoding channel which has reached the end

**lock** ()

Locks a stream, MOD music or recording channel to the current thread.

**Returns** True on success, False on failure.

**Return type** bool

**unlock** ()

Unlocks a stream, MOD music or recording channel from the current thread.

**Returns** True on success, False on failure.

**Return type** bool

**get\_3d\_attributes** ()

Retrieves the 3D attributes of a sample, stream, or MOD music channel with 3D functionality.

**Returns** A dict containing the stream's 3d attributes

**Return type** dict

**Raises** `sound_lib.main.BassError` – If this channel does not have 3d functionality.

**set\_3d\_attributes** (*mode=-1, min=0.0, max=0.0, iangle=-1, oangle=-1, outvol=-1*)

Sets the 3D attributes of a sample, stream, or MOD music channel with 3D functionality.

**Parameters**

- **mode** – The 3D processing mode. Defaults to -1.
- **min** (*float*) – The minimum distance. The channel's volume is at maximum when the listener is within this distance. ... 0 or less = leave current. Defaults to 0.0.
- **max** (*float*) – The maximum distance. The channel's volume stops decreasing when the listener is beyond this distance. ... 0 or less = leave current. Defaults to 0.0.
- **iangle** (*int*) – The angle of the inside projection cone in degrees. ... 0 (no cone) to 360 (sphere), -1 = leave current. Defaults to -1.
- **oangle** (*int*) – The angle of the inside projection cone in degrees. ... 0 (no cone) to 360 (sphere), -1 = leave current. Defaults to -1.
- **outvol** (*float*) – The delta-volume outside the outer projection cone. ... 0 (silent) to 1 (same as inside the cone), less than 0 = leave current. Defaults to -1.0.

**Returns** True on success, False otherwise.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If this channel does not have 3d functionality, or one or more attribute values are invalid.

**get\_3d\_position** ()

Retrieves the 3D position of a sample, stream, or MOD music channel with 3D functionality.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**set\_3d\_position** (*position=None, orientation=None, velocity=None*)

Sets the 3D position of a sample, stream, or MOD music channel with 3D functionality.

**Parameters**

- **position** – Defaults to None.
- **orientation** – Defaults to None.
- **velocity** – Defaults to None

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**set\_link** (*handle*)

Links two MOD music or stream channels together.

**Parameters** **handle** – The bass handle to link with this one. Can take both a `sound_lib.channel` or bass handle. Must be HMUSIC or HSTREAM.

**Returns** True on success, False on failure.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If handle points to an invalid channel, either one is a decoding channel, or this channel is already linked to handle.

**remove\_link** (*handle*)

Removes a link between two MOD music or stream channels.



**Parameters** **handle** – The bass handle to unlink with this one. Can take both a `sound_lib.channel` or bass handle. Must be a HMUSIC or HSTREAM. Must currently be linked.

**Returns** True on success, False on failure.

**Return type** bool

**Raises** `sound_lib.main.BassError` – If chan is either not a valid channel, or is not already linked to handle.

**get\_frequency()**

Retrieves sample frequency (sample rate).

**Returns** True on success, False on failure.

**Return type** bool

**set\_frequency(frequency)**

Sets the frequency (sample rate) of this channel.

**Parameters** **frequency** (*float*) – The sample rate... 0 = original rate (when the channel was created).

**Returns** True on success, False on failure.

**Return type** bool

**frequency**

Retrieves sample frequency (sample rate).

**Returns** True on success, False on failure.

**Return type** bool

**get\_pan()**

Gets the panning/balance position of this channel.

**set\_pan(pan)**

Sets the panning/balance position of this channel.

**Parameters** **pan** (*float*) – The pan position... -1 (full left) to +1 (full right), 0 = centre.

**Returns** True on success, False on Failure.

**Return type** bool

**pan**

Gets the panning/balance position of this channel.

**get\_volume()**

Gets the volume level of a channel.

**Returns** The volume level... 0 = silent, 1.0 = normal, above 1.0 = amplification.

**Return type** float

**set\_volume(volume)**

sets the volume level of a channel.

**Parameters** **volume** (*float*) – The volume level... 0 = silent, 1.0 = normal, above 1.0 = amplification.

**Returns** True on success, False on failure.

**volume**

Gets the volume level of a channel.

**Returns** The volume level. . . 0 = silent, 1.0 = normal, above 1.0 = amplification.

**Return type** float

**get\_data** (*length=16384*)

Retrieves the immediate sample data (or an FFT representation of it) of this channel. Must be a sample channel, stream, MOD music, or recording channel.

**Parameters** **length** – Number of bytes wanted (up to 268435455 or 0xFFFFFFFF). Defaults to 16384.

**Returns** The requested bytes.

**Raises** `sound_lib.main.BassError` – If this channel has reached the end, or the BASS\_DATA\_AVAILABLE flag was used and this is a decoding channel.

**get\_looping** ()

Returns whether this channel is currently setup to loop.

**set\_looping** (*looping*)

Determines whether this channel is setup to loop.

**Parameters** **looping** – (bool): Specifies whether this channel should loop.

**looping**

Returns whether this channel is currently setup to loop.

**free** ()

Frees a channel.

**Returns** True on success, False on failure.

**Return type** bool

**get\_x** ()

Retrieves this channel's position on the X-axis, if 3d functionality is available.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**set\_x** (*val*)

Sets positioning of this channel on the X-axis, if 3d functionality is available.

**Parameters** **val** – The coordinate position.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**x**

Retrieves this channel's position on the X-axis, if 3d functionality is available.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**get\_y** ()

Retrieves this channel's position on the Y-axis, if 3d functionality is available.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**set\_y** (*val*)

Sets positioning of this channel on the Y-axis, if 3d functionality is available.

**Parameters** **val** – The coordinate position.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**y**

Retrieves this channel's position on the Y-axis, if 3d functionality is available.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**get\_z()**

Retrieves this channel's position on the Z-axis, if 3d functionality is available.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**set\_z(val)**

Sets positioning of this channel on the Z-axis, if 3d functionality is available.

**Parameters** `val` – The coordinate position.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**z**

Retrieves this channel's position on the Z-axis, if 3d functionality is available.

**Raises** `sound_lib.main.BassError` – If this channel was not initialized with support for 3d functionality.

**get\_attributes()**

Retrieves all values of all attributes from this object and displays them in a dictionary whose keys are determined by this object's `attribute_mapping`

## 2.1.2 `sound_lib.stream`

**class** `sound_lib.stream.BaseStream(handle)`

**free()**

Frees a sample stream's resources, including any sync/DSP/FX it has.

**get\_file\_position(mode)**

Retrieves the file position/status of a stream.

**Parameters** `mode` –

**Returns** The requested file position on success, -1 otherwise.

**Return type** `int`

**Raises** `sound_lib.main.BassError` – If the handle is invalid, the stream is not a `FileStream`, or the requested position is not available.

**setup\_flag\_mapping()**

**class** `sound_lib.stream.Stream(freq=44100, chans=2, flags=0, proc=None, user=None, three_d=False, autofree=False, decode=False)`

A sample stream. Higher-level streams are used in 90% of cases.

**class** `sound_lib.stream.FileStream(mem=False, file=None, offset=0, length=0, flags=0, three_d=False, mono=False, autofree=False, decode=False, unicode=True)`

A sample stream that loads from a supported filetype. Can load from both disk and memory.

```
class sound_lib.stream.URLStream (url="", offset=0, flags=0, downloadproc=None, user=None,
                                   three_d=False, autofree=False, decode=False, uni-
                                   code=True)
```

Creates a sample stream from a file found on the internet. Downloaded data can optionally be received through a callback function for further manipulation.

```
class sound_lib.stream.PushStream (freq=44100, chans=2, flags=0, user=None, three_d=False,
                                   autofree=False, decode=False)
```

A stream that receives and plays raw audio data in realtime.

```
push (data)
```

Adds sample data to the stream.

**Parameters** *data* (*bytes*) – Data to be sent.

**Returns** The amount of queued data on success, -1 otherwise.

**Return type** *int*

**Raises** *sound\_lib.main.BassError* – If the stream has ended or there is insufficient memory.

### 2.1.3 *sound\_lib.recording*

```
class sound_lib.recording.Recording (frequency=44100, channels=2, flags=32768,
                                     proc=None, user=None)
```

Base class for implementing audio recording functionality. Inherits from *sound\_lib.channel.Channel*. Everything works based on those functions. For example, calling play starts, stop stops, etc etc.

```
free ()
```

Frees all resources associated with the recording. Automatically called when a channel is destroyed. Define this in your subclass, as by default it does nothing.

```
class sound_lib.recording.WaveRecording (filename="", proc=None, *args, **kwargs)
```

Allows for making wave audio recordings to the filesystem.

```
play (*args, **kwargs)
```

Starts (or resumes) playback of a sample, stream, MOD music, or recording.

**Parameters** *restart* (*bool*) – Specifies whether playback position should be thrown to the beginning of the stream. Defaults to False

**Returns** True on success, False otherwise.

**Return type** *bool*

```
stop (*args, **kwargs)
```

Stops a sample, stream, MOD music, or recording.

### 2.1.4 *sound\_lib.output*

```
class sound_lib.output.Output (device=-1, frequency=44100, flags=0, window=0, clsid=None)
```

```
init_device (device=None, frequency=None, flags=None, window=None, clsid=None)
```

**Parameters**

- **device** – (Default value = None)
- **frequency** – (Default value = None)

- **flags** – (Default value = None)
- **window** – (Default value = None)
- **clsid** – (Default value = None)

Returns:

**start** ()

**pause** ()

**stop** ()

**get\_device** ()

**set\_device** (*device*)

**Parameters device** –

Returns:

**device**

**get\_volume** ()

**set\_volume** (*volume*)

**Parameters volume** –

Returns:

**volume**

**static free** ()

**get\_proxy** ()

**set\_proxy** (*proxy*)

**Parameters proxy** –

Returns:

**use\_default\_device** (*use=True*)

**Parameters use** – (Default value = True)

Returns:

**static get\_device\_names** ()

**Convenience method that returns a list of device names that are considered valid by bass.**

Args:

Returns:

**find\_device\_by\_name** (*name*)

**Parameters name** –

Returns:

**find\_default\_device** ()

**find\_user\_provided\_device** (*device\_name*)

**Parameters device\_name** –

Returns:

```
class sound_lib.output.ThreeDOutput (flags=4, *args, **kwargs)
```

```
get_3d_factors ()
```

```
set_3d_factors (distance_factor=-1, rolloff=-1, doppler_factor=-1)
```

**Parameters**

- **distance\_factor** – (Default value = -1)
- **rolloff** – (Default value = -1)
- **doppler\_factor** – (Default value = -1)

Returns:

**distance\_factor**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**rolloff**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**doppler\_factor**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

```
set_eax_parameters (environment=None, volume=None, decay=None, damp=None)
```

**Parameters**

- **environment** – (Default value = None)
- **volume** – (Default value = None)
- **decay** – (Default value = None)
- **damp** – (Default value = None)

Returns:

```
get_3d_algorithm ()
```

```
set_3d_algorithm (algo)
```

**Parameters algo –**

Returns:

## 2.1.5 *sound\_lib.input*

```
class sound_lib.input.Input (device=-1)
```

Provides initialization and management for recording on a global level. Initialization is required if wanting to grab audio from an input device.

```
free ()
```

Frees all resources used by the recording device.

```
get_device ()
```

Retrieves the device used for recording.

**Returns** The device index on success, -1 on failure.

**Return type** int

---

**set\_device** (*device*)  
 Sets the device to use for recording.

**Parameters** **device** (*int*) – Device to use... 0 = first.

**device**  
 Retrieves the device used for recording.

**Returns** The device index on success, -1 on failure.

**Return type** int

**static get\_device\_names** ()

**Convenience method that returns a list of device names that are considered** valid by bass.

**Returns** A list containing names of input devices on the system.

**Return type** list

**find\_device\_by\_name** (*name*)  
 Attempts to locate an input device given it's name.

**Parameters** **name** (*str*) – Name of the device to search for.

**Returns** Index of the device if found.

**Return type** int

**Raises** `ValueError` – If the provided name was not found among input devices.

**find\_default\_device** ()  
 Returns the index of the default device.

**find\_user\_provided\_device** (*device\_name*)  
 Locate an input device given it's name, falling back to the default if not found.

**Parameters** **device\_name** (*str*) – Name of the device to search for.

**Returns** Index of the requested device on success, default device on failure.

**Return type** int

**class** `sound_lib.input.WASAPIInput` (*device=-2, frequency=0, channels=0, flags=0, buffer=0.0, period=0.0, callback=None*)  
 Provides the ability to use WASAPI (windows audio session API) input. Supported on windows versions above vista.

**free** ()  
 Frees all resources used by the recording device.

**set\_device** (*device*)  
 Sets the device to use for recording.

**Parameters** **device** (*int*) – Device to use... 0 = first.

**get\_device** ()  
 Retrieves the device used for recording.

**Returns** The device index on success, -1 on failure.

**Return type** int

**device**  
 Retrieves the device used for recording.

**Returns** The device index on success, -1 on failure.

**Return type** int

**start** ()

Starts the device.

**Returns** True on success, False otherwise.

**Return type** bool

**stop** (*reset=False*)

Stops the device.

**Parameters** **reset** – (Default value = False): Flush the device buffer?

**Returns** True on success, False otherwise.

**Return type** bool

## 2.1.6 *sound\_lib.main*

**exception** *sound\_lib.main.BassError* (*code, description*)

Error that is raised when there is a problem with a Bass call.

*sound\_lib.main.bass\_call* (*function, \*args*)

Makes a call to bass and raises an exception if it fails. This will most likely prove unnecessary for external usage, however we keep it just in case.

**Parameters**

- **function** (*str*) – Name of the internal bass function to be called.
- **\*args** – Arguments that will be passed to function.

**Returns** Raw output from the specified bass function. Refer to the official docs.

*sound\_lib.main.bass\_call\_0* (*function, \*args*)

Makes a call to bass and raises an exception if it fails. Does not consider 0 an error. This will most likely prove unnecessary for external usage, however we keep it just in case.

**Parameters**

- **function** (*str*) – Name of the internal bass function to be called.
- **\*args** – Arguments that will be passed to function.

**Returns** Raw output from the specified bass function. Refer to the official docs.

*sound\_lib.main.update\_3d\_system* (*func*)

Decorator to automatically update the 3d system after a function call.

**class** *sound\_lib.main.FlagObject*

An object which translates bass flags into human-readable/usable items

**flags\_for** (*\*\*flags*)

Retrieves flags for given attributes.

**Parameters** **\*\*flags** – A mapping of human-readable attribute names to numeric values.

**Returns** A bitmask containing the specified flags, or 0 if nothing was provided.

**Return type** int

**setup\_flag\_mapping** ()

Sets up a class-level mapping of common flags, in the format human-readable name:value. Typically expanded upon in a subclass.



### 2.1.7 *sound\_lib.music*

```
class sound_lib.music.Music (mem=False, file=None, offset=0, length=0, flags=0, freq=0)
```

### 2.1.8 *sound\_lib.listener*

```
class sound_lib.listener.Listener
```

```
get_3d_position ()
```

```
set_3d_position (position=None, velocity=None, front=None, top=None)
```

Sets the position, velocity, and orientation of the listener (ie. the player).

#### Parameters

- **position** – (Default value = None)
- **velocity** – (Default value = None)
- **front** – (Default value = None)
- **top** – (Default value = None)

Returns:

```
get_position ()
```

```
set_position (position)
```

#### Parameters **position** –

Returns:

**position**

**x**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**y**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**z**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

```
get_velocity ()
```

```
set_velocity (velocity)
```

#### Parameters **velocity** –

Returns:

**velocity**

**x\_velocity**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**y\_velocity**

partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**z\_velocity**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**get\_front()**

**set\_front(front)**

**Parameters front –**

Returns:

**front**

**front\_x**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**front\_y**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**front\_z**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**get\_top()**

**set\_top(top)**

**Parameters top –**

Returns:

**top**

**top\_x**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**top\_y**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

**top\_z**  
partial(func, \*args, \*\*keywords) - new function with partial application of the given arguments and keywords.

### 2.1.9 *sound\_lib.encoder*

**class** sound\_lib.encoder.**Encoder**(source, command\_line, pcm=False, no\_header=False, rf64=False, big\_endian=False, fp\_8bit=False, fp\_16bit=False, fp\_24bit=False, fp\_32bit=False, queue=False, limit=False, no\_limit=False, pause=True, autofree=False, callback=None, user=None)

**setup\_flag\_mapping()**

**set\_title(title=None, url=None)**

**Parameters**

- **title** – (Default value = None)

- **url** – (Default value = None)

Returns:

**paused**

**is\_stopped()**

**stop()**

```
class sound_lib.encoder.BroadcastEncoder(source_encoder, server, password, content,  
                                         name=None, url=None, genre=None, de-  
                                         scription=None, headers=None, bitrate=0,  
                                         public=False)
```

**get\_stats** (type, password=None)

**Parameters**

- **type** –
- **password** – (Default value = None)

Returns:

```
class sound_lib.encoder.Server (encoder, port, buffer=64000, burst=64000, user=None)
```

Local Audio Server

**client\_callback** (handle, connect, client, headers, user)

**Parameters**

- **handle** –
- **client** –
- **connect** –
- **user** –

**Returns** An integer 0 or 1 if the connection should be accepted

### 2.1.10 *sound\_lib.effects*



## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### S

- `sound_lib.channel`, [7](#)
- `sound_lib.effects`, [23](#)
- `sound_lib.encoder`, [22](#)
- `sound_lib.input`, [18](#)
- `sound_lib.listener`, [21](#)
- `sound_lib.main`, [20](#)
- `sound_lib.music`, [21](#)
- `sound_lib.output`, [16](#)
- `sound_lib.recording`, [16](#)
- `sound_lib.stream`, [15](#)





## B

BaseStream (class in *sound\_lib.stream*), 15  
bass\_call() (in module *sound\_lib.main*), 20  
bass\_call\_0() (in module *sound\_lib.main*), 20  
BassError, 20  
BroadcastEncoder (class in *sound\_lib.encoder*), 23  
bytes\_to\_seconds() (*sound\_lib.channel.Channel* method), 10

## C

Channel (class in *sound\_lib.channel*), 7  
client\_callback() (*sound\_lib.encoder.Server* method), 23

## D

device (*sound\_lib.channel.Channel* attribute), 9  
device (*sound\_lib.input.Input* attribute), 19  
device (*sound\_lib.input.WASAPIInput* attribute), 19  
device (*sound\_lib.output.Output* attribute), 17  
distance\_factor (*sound\_lib.output.ThreeDOutput* attribute), 18  
doppler\_factor (*sound\_lib.output.ThreeDOutput* attribute), 18

## E

Encoder (class in *sound\_lib.encoder*), 22

## F

FileStream (class in *sound\_lib.stream*), 15  
find\_default\_device() (*sound\_lib.input.Input* method), 19  
find\_default\_device() (*sound\_lib.output.Output* method), 17  
find\_device\_by\_name() (*sound\_lib.input.Input* method), 19  
find\_device\_by\_name() (*sound\_lib.output.Output* method), 17  
find\_user\_provided\_device() (*sound\_lib.input.Input* method), 19

find\_user\_provided\_device() (*sound\_lib.output.Output* method), 17  
FlagObject (class in *sound\_lib.main*), 20  
flags\_for() (*sound\_lib.main.FlagObject* method), 20  
free() (*sound\_lib.channel.Channel* method), 14  
free() (*sound\_lib.input.Input* method), 18  
free() (*sound\_lib.input.WASAPIInput* method), 19  
free() (*sound\_lib.output.Output* static method), 17  
free() (*sound\_lib.recording.Recording* method), 16  
free() (*sound\_lib.stream.BaseStream* method), 15  
frequency (*sound\_lib.channel.Channel* attribute), 13  
front (*sound\_lib.listener.Listener* attribute), 22  
front\_x (*sound\_lib.listener.Listener* attribute), 22  
front\_y (*sound\_lib.listener.Listener* attribute), 22  
front\_z (*sound\_lib.listener.Listener* attribute), 22

## G

get\_3d\_algorithm() (*sound\_lib.output.ThreeDOutput* method), 18  
get\_3d\_attributes() (*sound\_lib.channel.Channel* method), 11  
get\_3d\_factors() (*sound\_lib.output.ThreeDOutput* method), 18  
get\_3d\_position() (*sound\_lib.channel.Channel* method), 12  
get\_3d\_position() (*sound\_lib.listener.Listener* method), 21  
get\_attribute() (*sound\_lib.channel.Channel* method), 10  
get\_attributes() (*sound\_lib.channel.Channel* method), 15  
get\_data() (*sound\_lib.channel.Channel* method), 14  
get\_device() (*sound\_lib.channel.Channel* method), 9  
get\_device() (*sound\_lib.input.Input* method), 18  
get\_device() (*sound\_lib.input.WASAPIInput* method), 19  
get\_device() (*sound\_lib.output.Output* method), 17

- `get_device_names()` (*sound\_lib.input.Input static method*), 19
- `get_device_names()` (*sound\_lib.output.Output static method*), 17
- `get_file_position()`  
(*sound\_lib.stream.BaseStream method*), 15
- `get_frequency()` (*sound\_lib.channel.Channel method*), 13
- `get_front()` (*sound\_lib.listener.Listener method*), 22
- `get_info()` (*sound\_lib.channel.Channel method*), 11
- `get_length()` (*sound\_lib.channel.Channel method*), 9
- `get_level()` (*sound\_lib.channel.Channel method*), 11
- `get_looping()` (*sound\_lib.channel.Channel method*), 14
- `get_pan()` (*sound\_lib.channel.Channel method*), 13
- `get_position()` (*sound\_lib.channel.Channel method*), 8
- `get_position()` (*sound\_lib.listener.Listener method*), 21
- `get_proxy()` (*sound\_lib.output.Output method*), 17
- `get_stats()` (*sound\_lib.encoder.BroadcastEncoder method*), 23
- `get_top()` (*sound\_lib.listener.Listener method*), 22
- `get_velocity()` (*sound\_lib.listener.Listener method*), 21
- `get_volume()` (*sound\_lib.channel.Channel method*), 13
- `get_volume()` (*sound\_lib.output.Output method*), 17
- `get_x()` (*sound\_lib.channel.Channel method*), 14
- `get_y()` (*sound\_lib.channel.Channel method*), 14
- `get_z()` (*sound\_lib.channel.Channel method*), 15
- I**
- `init_device()` (*sound\_lib.output.Output method*), 16
- `Input` (*class in sound\_lib.input*), 18
- `is_active()` (*sound\_lib.channel.Channel method*), 8
- `is_paused` (*sound\_lib.channel.Channel attribute*), 8
- `is_playing` (*sound\_lib.channel.Channel attribute*), 8
- `is_sliding()` (*sound\_lib.channel.Channel method*), 11
- `is_stalled` (*sound\_lib.channel.Channel attribute*), 8
- `is_stopped` (*sound\_lib.channel.Channel attribute*), 8
- `is_stopped()` (*sound\_lib.encoder.Encoder method*), 23
- L**
- `length_in_seconds()` (*sound\_lib.channel.Channel method*), 10
- `Listener` (*class in sound\_lib.listener*), 21
- `lock()` (*sound\_lib.channel.Channel method*), 11
- `looping` (*sound\_lib.channel.Channel attribute*), 14
- M**
- `Music` (*class in sound\_lib.music*), 21
- O**
- `Output` (*class in sound\_lib.output*), 16
- P**
- `pan` (*sound\_lib.channel.Channel attribute*), 13
- `pause()` (*sound\_lib.channel.Channel method*), 7
- `pause()` (*sound\_lib.output.Output method*), 17
- `paused` (*sound\_lib.encoder.Encoder attribute*), 23
- `play()` (*sound\_lib.channel.Channel method*), 7
- `play()` (*sound\_lib.recording.WaveRecording method*), 16
- `play_blocking()` (*sound\_lib.channel.Channel method*), 7
- `position` (*sound\_lib.channel.Channel attribute*), 8
- `position` (*sound\_lib.listener.Listener attribute*), 21
- `push()` (*sound\_lib.stream.PushStream method*), 16
- `PushStream` (*class in sound\_lib.stream*), 16
- R**
- `Recording` (*class in sound\_lib.recording*), 16
- `remove_link()` (*sound\_lib.channel.Channel method*), 12
- `rolloff` (*sound\_lib.output.ThreeDOutput attribute*), 18
- S**
- `seconds_to_bytes()` (*sound\_lib.channel.Channel method*), 10
- `Server` (*class in sound\_lib.encoder*), 23
- `set_3d_algorithm()`  
(*sound\_lib.output.ThreeDOutput method*), 18
- `set_3d_attributes()` (*sound\_lib.channel.Channel method*), 11
- `set_3d_factors()` (*sound\_lib.output.ThreeDOutput method*), 18
- `set_3d_position()` (*sound\_lib.channel.Channel method*), 12
- `set_3d_position()` (*sound\_lib.listener.Listener method*), 21
- `set_attribute()` (*sound\_lib.channel.Channel method*), 10
- `set_device()` (*sound\_lib.channel.Channel method*), 9
- `set_device()` (*sound\_lib.input.Input method*), 19
- `set_device()` (*sound\_lib.input.WASAPIInput method*), 19
- `set_device()` (*sound\_lib.output.Output method*), 17

set\_eax\_parameters() (sound\_lib.output.ThreeDOutput method), 18  
 set\_frequency() (sound\_lib.channel.Channel method), 13  
 set\_front() (sound\_lib.listener.Listener method), 22  
 set\_fx() (sound\_lib.channel.Channel method), 9  
 set\_link() (sound\_lib.channel.Channel method), 12  
 set\_looping() (sound\_lib.channel.Channel method), 14  
 set\_pan() (sound\_lib.channel.Channel method), 13  
 set\_position() (sound\_lib.channel.Channel method), 8  
 set\_position() (sound\_lib.listener.Listener method), 21  
 set\_proxy() (sound\_lib.output.Output method), 17  
 set\_title() (sound\_lib.encoder.Encoder method), 22  
 set\_top() (sound\_lib.listener.Listener method), 22  
 set\_velocity() (sound\_lib.listener.Listener method), 21  
 set\_volume() (sound\_lib.channel.Channel method), 13  
 set\_volume() (sound\_lib.output.Output method), 17  
 set\_x() (sound\_lib.channel.Channel method), 14  
 set\_y() (sound\_lib.channel.Channel method), 14  
 set\_z() (sound\_lib.channel.Channel method), 15  
 setup\_flag\_mapping() (sound\_lib.encoder.Encoder method), 22  
 setup\_flag\_mapping() (sound\_lib.main.FlagObject method), 20  
 setup\_flag\_mapping() (sound\_lib.stream.BaseStream method), 15  
 slide\_attribute() (sound\_lib.channel.Channel method), 10  
 sound\_lib.channel (module), 7  
 sound\_lib.effects (module), 23  
 sound\_lib.encoder (module), 22  
 sound\_lib.input (module), 18  
 sound\_lib.listener (module), 21  
 sound\_lib.main (module), 20  
 sound\_lib.music (module), 21  
 sound\_lib.output (module), 16  
 sound\_lib.recording (module), 16  
 sound\_lib.stream (module), 15  
 start() (sound\_lib.input.WASAPIInput method), 20  
 start() (sound\_lib.output.Output method), 17  
 stop() (sound\_lib.channel.Channel method), 9  
 stop() (sound\_lib.encoder.Encoder method), 23  
 stop() (sound\_lib.input.WASAPIInput method), 20  
 stop() (sound\_lib.output.Output method), 17  
 stop() (sound\_lib.recording.WaveRecording method), 16  
 Stream (class in sound\_lib.stream), 15

## T

ThreeDOutput (class in sound\_lib.output), 17  
 top (sound\_lib.listener.Listener attribute), 22  
 top\_x (sound\_lib.listener.Listener attribute), 22  
 top\_y (sound\_lib.listener.Listener attribute), 22  
 top\_z (sound\_lib.listener.Listener attribute), 22

## U

unlock() (sound\_lib.channel.Channel method), 11  
 update() (sound\_lib.channel.Channel method), 9  
 update\_3d\_system() (in module sound\_lib.main), 20  
 URLStream (class in sound\_lib.stream), 15  
 use\_default\_device() (sound\_lib.output.Output method), 17

## V

velocity (sound\_lib.listener.Listener attribute), 21  
 volume (sound\_lib.channel.Channel attribute), 13  
 volume (sound\_lib.output.Output attribute), 17

## W

WASAPIInput (class in sound\_lib.input), 19  
 WaveRecording (class in sound\_lib.recording), 16

## X

x (sound\_lib.channel.Channel attribute), 14  
 x (sound\_lib.listener.Listener attribute), 21  
 x\_velocity (sound\_lib.listener.Listener attribute), 21

## Y

y (sound\_lib.channel.Channel attribute), 15  
 y (sound\_lib.listener.Listener attribute), 21  
 y\_velocity (sound\_lib.listener.Listener attribute), 21

## Z

z (sound\_lib.channel.Channel attribute), 15  
 z (sound\_lib.listener.Listener attribute), 21  
 z\_velocity (sound\_lib.listener.Listener attribute), 22