pysox Documentation

Release 1.4.1

Rachel Bittner

Sep 30, 2020

Contents

pysox is a Python wrapper around the amazing SoX command line tool.

CHAPTER 1

Installation

On Linux

```
# optional - if you want support for mp3, flac and ogg files
$ apt-get install libsox-fmt-all
# install the sox command line tool
$ apt-get install sox
# install pysox
$ pip install sox
```

On Mac with Homebrew

```
# optional - if you want support for mp3, flac and ogg files
$ brew install sox --with-lame --with-flac --with-libvorbis
# install the sox command line tool
$ brew install sox
# install pysox
$ pip install sox
```

CHAPTER 2

Examples

2.1 Transformer Example

Transform audio files

```
import sox
# create transformer
tfm = sox.Transformer()
# trim the audio between 5 and 10.5 seconds.
tfm.trim(5, 10.5)
# apply compression
tfm.compand()
# apply a fade in and fade out
tfm.fade(fade_in_len=1.0, fade_out_len=0.5)
# create an output file.
tfm.build_file('path/to/input_audio.wav', 'path/to/output/audio.aiff')
# or equivalently using the legacy API
tfm.build('path/to/input_audio.wav', 'path/to/output/audio.aiff')
# get the output in-memory as a numpy array
# by default the sample rate will be the same as the input file
array_out = tfm.build_array(input_filepath='path/to/input_audio.wav')
# see the applied effects
tfm.effects_log
> ['trim', 'compand', 'fade']
```

Transform in-memory arrays

```
import numpy
import sox
# sample rate in Hz
sample_rate = 44100
# generate a 1-second sine tone at 440 Hz
y = np.sin(2 * np.pi * 440.0 * np.arange(sample_rate * 1.0) / sample_rate)
# create a transformer
```

(continues on next page)

(continued from previous page)

```
tfm = sox.Transformer()
# shift the pitch up by 2 semitones
tfm.pitch(2)
# transform an in-memory array and return an array
y_out = tfm.build_array(input_array=y, sample_rate_in=sample_rate)
# instead, save output to a file
tfm.build_file(
    input_array=y, sample_rate_in=sample_rate,
    output_filepath='path/to/output.wav'
)
# create an output file with a different sample rate
tfm.set_output_format(rate=8000)
tfm.build_file(
    input_array=y, sample_rate_in=sample_rate,
    output_filepath='path/to/output_8k.wav'
)
```

2.2 Combiner Example

```
import sox
# create combiner
cbn = sox.Combiner()
# pitch shift combined audio up 3 semitones
cbn.pitch(3.0)
# convert output to 8000 Hz stereo
cbn.convert(samplerate=8000, n_channels=2)
# create the output file
cbn.build(
    ['input1.wav', 'input2.wav', 'input3.wav'], output.wav, 'concatenate'
)

# the combiner does not currently support array input/output
```

2.3 File Info Example

```
import sox
# get the sample rate
sample_rate = sox.file_info.sample_rate('path/to/file.mp3')
# get the number of samples
n_samples = sox.file_info.num_samples('path/to/file.wav')
# determine if a file is silent
is_silent = sox.file_info.silent('path/to/file.aiff')

# file info doesn't currently support array input
```

CHAPTER 3

API Reference

3.1 Transformers

Python wrapper around the SoX library. This module requires that SoX is installed.

class sox.transform.**Transformer**

Audio file transformer. Class which allows multiple effects to be chained to create an output file, saved to output_filepath.

Methods

set_globals(self[, dither, guard, ...])	Sets SoX's global arguments.
build(self[, input_filepath, ...])	Given an input file or array, creates an output_file on disk by executing the current set of commands.
build_file(self[, input_filepath, ...])	An alias for build.
build_array(self[, input_filepath, ...])	Given an input file or array, returns the ouput as a numpy array by executing the current set of commands.

allpass (*self, frequency, width_q=2.0*)

Apply a two-pole all-pass filter. An all-pass filter changes the audio's frequency to phase relationship without changing its frequency to amplitude relationship. The filter is described in detail in at http://musicdsp.org/files/Audio-EQ-Cookbook.txt

Parameters

frequency [float] The filter's center frequency in Hz.

width_q [float, default=2.0] The filter's width as a Q-factor.

See also:

equalizer, ***highpass***, ***lowpass***, ***sinc***

bandpass (*self*, *frequency*, *width_q=2.0*, *constant_skirt=False*)
Apply a two-pole Butterworth band-pass filter with the given central frequency, and (3dB-point) bandwidth. The filter rolls off at 6dB per octave (20dB per decade) and is described in detail in http://musicdsp.org/files/Audio-EQ-Cookbook.txt

Parameters

frequency [float] The filter's center frequency in Hz.

width_q [float, default=2.0] The filter's width as a Q-factor.

constant_skirt [bool, default=False] If True, selects constant skirt gain (peak gain = width_q). If False, selects constant 0dB peak gain.

See also:

`bandreject`, `sinc`

bandreject (*self*, *frequency*, *width_q=2.0*)
Apply a two-pole Butterworth band-reject filter with the given central frequency, and (3dB-point) bandwidth. The filter rolls off at 6dB per octave (20dB per decade) and is described in detail in http://musicdsp.org/files/Audio-EQ-Cookbook.txt

Parameters

frequency [float] The filter's center frequency in Hz.

width_q [float, default=2.0] The filter's width as a Q-factor.

constant_skirt [bool, default=False] If True, selects constant skirt gain (peak gain = width_q). If False, selects constant 0dB peak gain.

See also:

`bandreject`, `sinc`

bass (*self*, *gain_db*, *frequency=100.0*, *slope=0.5*)
Boost or cut the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation.

The filters are described in detail in http://musicdsp.org/files/Audio-EQ-Cookbook.txt

Parameters

gain_db [float] The gain at 0 Hz. For a large cut use -20, for a large boost use 20.

frequency [float, default=100.0] The filter's cutoff frequency in Hz.

slope [float, default=0.5] The steepness of the filter's shelf transition. For a gentle slope use 0.3, and use 1.0 for a steep slope.

See also:

`treble`, `equalizer`

bend (*self*, *n_bends*, *start_times*, *end_times*, *cents*, *frame_rate=25*, *oversample_rate=16*)
Changes pitch by specified amounts at specified times. The pitch-bending algorithm utilises the Discrete Fourier Transform (DFT) at a particular frame rate and over-sampling rate.

Parameters

n_bends [int] The number of intervals to pitch shift

start_times [list of floats] A list of absolute start times (in seconds), in order

end_times [list of floats] A list of absolute end times (in seconds) in order. [start_time, end_time] intervals may not overlap!

cents [list of floats] A list of pitch shifts in cents. A positive value shifts the pitch up, a negative value shifts the pitch down.

frame_rate [int, default=25] The number of DFT frames to process per second, between 10 and 80

oversample_rate: int, default=16 The number of frames to over sample per second, between 4 and 32

See also:

`pitch`

biquad(*self*, *b*, *a*)

Apply a biquad IIR filter with the given coefficients.

Parameters

b [list of floats] Numerator coefficients. Must be length 3

a [list of floats] Denominator coefficients. Must be length 3

See also:

`fir`, `treble`, `bass`, `equalizer`

build(*self*, *input_filepath=None*, *output_filepath=None*, *input_array=None*, *sample_rate_in=None*, *extra_args=None*, *return_output=False*)

Given an input file or array, creates an output_file on disk by executing the current set of commands. This function returns True on success. If return_output is True, this function returns a triple of (status, out, err), giving the success state, along with stdout and stderr returned by sox.

Parameters

input_filepath [str or None] Either path to input audio file or None for array input.

output_filepath [str] Path to desired output file. If a file already exists at the given path, the file will be overwritten. If '-n', no file is created.

input_array [np.ndarray or None] An np.ndarray of an waveform with shape (n_samples, n_channels). sample_rate_in must also be provided. If None, input_filepath must be specified.

sample_rate_in [int] Sample rate of input_array. This argument is ignored if input_array is None.

extra_args [list or None, default=None] If a list is given, these additional arguments are passed to SoX at the end of the list of effects. Don't use this argument unless you know exactly what you're doing!

return_output [bool, default=False] If True, returns the status and information sent to stderr and stdout as a tuple (status, stdout, stderr). If output_filepath is None, return_output=True by default. If False, returns True on success.

Returns

status [bool] True on success.

out [str (optional)] This is not returned unless return_output is True. When returned, captures the stdout produced by sox.

err [str (optional)] This is not returned unless return_output is True. When returned, captures the stderr produced by sox.

Examples

```
>>> import numpy as np
>>> import sox
>>> tfm = sox.Transformer()
>>> sample_rate = 44100
>>> y = np.sin(2 * np.pi * 440.0 * np.arange(sample_rate * 1.0) / sample_rate)
```

file in, file out - basic usage

```
>>> status = tfm.build('path/to/input.wav', 'path/to/output.mp3')
```

file in, file out - equivalent usage

```
>>> status = tfm.build(
        input_filepath='path/to/input.wav',
        output_filepath='path/to/output.mp3'
    )
```

array in, file out

```
>>> status = tfm.build(
        input_array=y, sample_rate_in=sample_rate,
        output_filepath='path/to/output.mp3'
    )
```

build_array(*self*, *input_filepath=None*, *input_array=None*, *sample_rate_in=None*, *extra_args=None*)

Given an input file or array, returns the ouput as a numpy array by executing the current set of commands. By default the array will have the same sample rate as the input file unless otherwise specified using set_output_format. Functions such as rate, channels and convert will be ignored!

Parameters

input_filepath [str or None] Either path to input audio file or None.

input_array [np.ndarray or None] A np.ndarray of an waveform with shape (n_samples, n_channels). If this argument is passed, sample_rate_in must also be provided. If None, input_filepath must be specified.

sample_rate_in [int] Sample rate of input_array. This argument is ignored if input_array is None.

extra_args [list or None, default=None] If a list is given, these additional arguments are passed to SoX at the end of the list of effects. Don't use this argument unless you know exactly what you're doing!

Returns

output_array [np.ndarray] Output audio as a numpy array

Examples

```
>>> import numpy as np
>>> import sox
>>> tfm = sox.Transformer()
>>> sample_rate = 44100
>>> y = np.sin(2 * np.pi * 440.0 * np.arange(sample_rate * 1.0) / sample_rate)
```

file in, array out

```
>>> output_array = tfm.build(input_filepath='path/to/input.wav')
```

array in, array out

```
>>> output_array = tfm.build(input_array=y, sample_rate_in=sample_rate)
```

specifying the output sample rate

```
>>> tfm.set_output_format(rate=8000)
>>> output_array = tfm.build(input_array=y, sample_rate_in=sample_rate)
```

if an effect changes the number of channels, you must explicitly specify the number of output channels

```
>>> tfm.remix(remix_dictionary={1: [1], 2: [1], 3: [1]})
>>> tfm.set_output_format(channels=3)
>>> output_array = tfm.build(input_array=y, sample_rate_in=sample_rate)
```

build_file(*self*, *input_filepath=None*, *output_filepath=None*, *input_array=None*, *sample_rate_in=None*, *extra_args=None*, *return_output=False*)

An alias for build. Given an input file or array, creates an output_file on disk by executing the current set of commands. This function returns True on success. If return_output is True, this function returns a triple of (status, out, err), giving the success state, along with stdout and stderr returned by sox.

Parameters

input_filepath [str or None] Either path to input audio file or None for array input.

output_filepath [str] Path to desired output file. If a file already exists at the given path, the file will be overwritten. If '-n', no file is created.

input_array [np.ndarray or None] An np.ndarray of an waveform with shape (n_samples, n_channels). sample_rate_in must also be provided. If None, input_filepath must be specified.

sample_rate_in [int] Sample rate of input_array. This argument is ignored if input_array is None.

extra_args [list or None, default=None] If a list is given, these additional arguments are passed to SoX at the end of the list of effects. Don't use this argument unless you know exactly what you're doing!

return_output [bool, default=False] If True, returns the status and information sent to stderr and stdout as a tuple (status, stdout, stderr). If output_filepath is None, return_output=True by default. If False, returns True on success.

Returns

status [bool] True on success.

out [str (optional)] This is not returned unless return_output is True. When returned, captures the stdout produced by sox.

err [str (optional)] This is not returned unless return_output is True. When returned, captures the stderr produced by sox.

Examples

```
>>> import numpy as np
>>> import sox
>>> tfm = sox.Transformer()
>>> sample_rate = 44100
>>> y = np.sin(2 * np.pi * 440.0 * np.arange(sample_rate * 1.0) / sample_rate)
```

file in, file out - basic usage

```
>>> status = tfm.build('path/to/input.wav', 'path/to/output.mp3')
```

file in, file out - equivalent usage

```
>>> status = tfm.build(
        input_filepath='path/to/input.wav',
        output_filepath='path/to/output.mp3'
    )
```

array in, file out

```
>>> status = tfm.build(
        input_array=y, sample_rate_in=sample_rate,
        output_filepath='path/to/output.mp3'
    )
```

channels (*self*, *n_channels*)

Change the number of channels in the audio signal. If decreasing the number of channels it mixes channels together, if increasing the number of channels it duplicates.

Note: This overrides arguments used in the convert effect!

Parameters

n_channels [int] Desired number of channels.

See also:

convert

chorus (*self*, *gain_in=0.5*, *gain_out=0.9*, *n_voices=3*, *delays=None*, *decays=None*, *speeds=None*, *depths=None*, *shapes=None*)

Add a chorus effect to the audio. This can makeasingle vocal sound like a chorus, but can also be applied to instrumentation.

Chorus resembles an echo effect with a short delay, but whereas with echo the delay is constant, with chorus, it is varied using sinusoidal or triangular modulation. The modulation depth defines the range the modulated delay is played before or after the delay. Hence the delayed sound will sound slower or faster, that is the delayed sound tuned around the original one, like in a chorus where some vocals are slightly off key.

Parameters

gain_in [float, default=0.3] The time in seconds over which the instantaneous level of the input signal is averaged to determine increases in volume.

gain_out [float, default=0.8] The time in seconds over which the instantaneous level of the input signal is averaged to determine decreases in volume.

n_voices [int, default=3] The number of voices in the chorus effect.

delays [list of floats > 20 or None, default=None] If a list, the list of delays (in miliseconds) of length n_voices. If None, the individual delay parameters are chosen automatically to be between 40 and 60 miliseconds.

decays [list of floats or None, default=None] If a list, the list of decays (as a fraction of gain_in) of length n_voices. If None, the individual decay parameters are chosen automatically to be between 0.3 and 0.4.

speeds [list of floats or None, default=None] If a list, the list of modulation speeds (in Hz) of length n_voices If None, the individual speed parameters are chosen automatically to be between 0.25 and 0.4 Hz.

depths [list of floats or None, default=None] If a list, the list of depths (in miliseconds) of length n_voices. If None, the individual delay parameters are chosen automatically to be between 1 and 3 miliseconds.

shapes [list of 's' or 't' or None, deault=None] If a list, the list of modulation shapes - 's' for sinusoidal or 't' for triangular - of length n_voices. If None, the individual shapes are chosen automatically.

`clear_effects`(*self*)
Remove all effects processes.

`compand`(*self, attack_time=0.3, decay_time=0.8, soft_knee_db=6.0, tf_points=[(-70, -70), (-60, -20), (0, 0)]*)
Compand (compress or expand) the dynamic range of the audio.

Parameters

attack_time [float, default=0.3] The time in seconds over which the instantaneous level of the input signal is averaged to determine increases in volume.

decay_time [float, default=0.8] The time in seconds over which the instantaneous level of the input signal is averaged to determine decreases in volume.

soft_knee_db [float or None, default=6.0] The ammount (in dB) for which the points at where adjacent line segments on the transfer function meet will be rounded. If None, no soft_knee is applied.

tf_points [list of tuples] Transfer function points as a list of tuples corresponding to points in (dB, dB) defining the compander's transfer function.

See also:

`mcompand`, *`contrast`*

`contrast`(*self, amount=75*)
Comparable with compression, this effect modifies an audio signal to make it sound louder.

Parameters

amount [float] Amount of enhancement between 0 and 100.

See also:

`compand`, *`mcompand`*

convert (*self*, *samplerate=None*, *n_channels=None*, *bitdepth=None*)
Converts output audio to the specified format.

Parameters

samplerate [float, default=None] Desired samplerate. If None, defaults to the same as input.

n_channels [int, default=None] Desired number of channels. If None, defaults to the same as input.

bitdepth [int, default=None] Desired bitdepth. If None, defaults to the same as input.

See also:

rate

dcshift (*self*, *shift=0.0*)
Apply a DC shift to the audio.

Parameters

shift [float] Amount to shift audio between -2 and 2. (Audio is between -1 and 1)

See also:

highpass

deemph (*self*)
Apply Compact Disc (IEC 60908) de-emphasis (a treble attenuation shelving filter). Pre-emphasis was applied in the mastering of some CDs issued in the early 1980s. These included many classical music albums, as well as now sought-after issues of albums by The Beatles, Pink Floyd and others. Pre-emphasis should be removed at playback time by a de-emphasis filter in the playback device. However, not all modern CD players have this filter, and very few PC CD drives have it; playing pre-emphasised audio without the correct de-emphasis filter results in audio that sounds harsh and is far from what its creators intended.

The de-emphasis filter is implemented as a biquad and requires the input audio sample rate to be either 44.1kHz or 48kHz. Maximum deviation from the ideal response is only 0.06dB (up to 20kHz).

See also:

bass, *treble*

delay (*self*, *positions*)
Delay one or more audio channels such that they start at the given positions.

Parameters

positions: list of floats List of times (in seconds) to delay each audio channel. If fewer positions are given than the number of channels, the remaining channels will be unaffected.

downsample (*self*, *factor=2*)
Downsample the signal by an integer factor. Only the first out of each factor samples is retained, the others are discarded.

No decimation filter is applied. If the input is not a properly bandlimited baseband signal, aliasing will occur. This may be desirable e.g., for frequency translation.

For a general resampling effect with anti-aliasing, see rate.

Parameters

factor [int, default=2] Downsampling factor.

See also:

rate, *upsample*

earwax (*self*)

Makes audio easier to listen to on headphones. Adds 'cues' to 44.1kHz stereo audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Warning: Will only work properly on 44.1kHz stereo audio!

echo (*self, gain_in=0.8, gain_out=0.9, n_echos=1, delays=[60], decays=[0.4]*)

Add echoing to the audio.

Echoes are reflected sound and can occur naturally amongst mountains (and sometimes large buildings) when talking or shouting; digital echo effects emulate this behav- iour and are often used to help fill out the sound of a single instrument or vocal. The time differ- ence between the original signal and the reflection is the 'delay' (time), and the loudness of the reflected signal is the 'decay'. Multiple echoes can have different delays and decays.

Parameters

gain_in [float, default=0.8] Input volume, between 0 and 1

gain_out [float, default=0.9] Output volume, between 0 and 1

n_echos [int, default=1] Number of reflections

delays [list, default=[60]] List of delays in miliseconds

decays [list, default=[0.4]] List of decays, relative to gain in between 0 and 1

See also:

echos, *reverb*, *chorus*

echos (*self, gain_in=0.8, gain_out=0.9, n_echos=1, delays=[60], decays=[0.4]*)

Add a sequence of echoes to the audio.

Like the echo effect, echos stand for 'ECHO in Sequel', that is the first echos takes the input, the second the input and the first echos, the third the input and the first and the second echos, ... and so on. Care should be taken using many echos; a single echos has the same effect as a single echo.

Parameters

gain_in [float, default=0.8] Input volume, between 0 and 1

gain_out [float, default=0.9] Output volume, between 0 and 1

n_echos [int, default=1] Number of reflections

delays [list, default=[60]] List of delays in miliseconds

decays [list, default=[0.4]] List of decays, relative to gain in between 0 and 1

See also:

echo, *reverb*, *chorus*

equalizer (*self*, *frequency*, *width_q*, *gain_db*)
Apply a two-pole peaking equalisation (EQ) filter to boost or reduce around a given frequency. This effect can be applied multiple times to produce complex EQ curves.

Parameters

frequency [float] The filter's central frequency in Hz.

width_q [float] The filter's width as a Q-factor.

gain_db [float] The filter's gain in dB.

See also:

bass, *treble*

fade (*self*, *fade_in_len=0.0*, *fade_out_len=0.0*, *fade_shape='q'*)
Add a fade in and/or fade out to an audio file. Default fade shape is 1/4 sine wave.

Parameters

fade_in_len [float, default=0.0] Length of fade-in (seconds). If fade_in_len = 0, no fade in is applied.

fade_out_len [float, defaut=0.0] Length of fade-out (seconds). If fade_out_len = 0, no fade in is applied.

fade_shape [str, default='q']

Shape of fade. Must be one of

- 'q' for quarter sine (default),
- 'h' for half sine,
- 't' for linear,
- 'l' for logarithmic
- 'p' for inverted parabola.

See also:

splice

fir (*self*, *coefficients*)
Use SoX's FFT convolution engine with given FIR filter coefficients.

Parameters

coefficients [list] fir filter coefficients

flanger (*self*, *delay=0*, *depth=2*, *regen=0*, *width=71*, *speed=0.5*, *shape='sine'*, *phase=25*, *interp='linear'*)
Apply a flanging effect to the audio.

Parameters

delay [float, default=0] Base delay (in miliseconds) between 0 and 30.

depth [float, default=2] Added swept delay (in miliseconds) between 0 and 10.

regen [float, default=0] Percentage regeneration between -95 and 95.

width [float, default=71,] Percentage of delayed signal mixed with original between 0 and 100.

speed [float, default=0.5] Sweeps per second (in Hz) between 0.1 and 10.

shape ['sine' or 'triangle', default='sine'] Swept wave shape

phase [float, default=25] Swept wave percentage phase-shift for multi-channel flange between 0 and 100. 0 = 100 = same phase on each channel

interp ['linear' or 'quadratic', default='linear'] Digital delay-line interpolation type.

See also:

tremolo

gain (*self*, *gain_db=0.0*, *normalize=True*, *limiter=False*, *balance=None*)
Apply amplification or attenuation to the audio signal.

Parameters

gain_db [float, default=0.0] Gain adjustment in decibels (dB).

normalize [bool, default=True] If True, audio is normalized to gain_db relative to full scale. If False, simply adjusts the audio power level by gain_db.

limiter [bool, default=False] If True, a simple limiter is invoked to prevent clipping.

balance [str or None, default=None]

Balance gain across channels. Can be one of:

- None applies no balancing (default)
- **'e' applies gain to all channels other than that with the** highest peak level, such that all channels attain the same peak level
- **'B' applies gain to all channels other than that with the** highest RMS level, such that all channels attain the same RMS level
- **'b' applies gain with clipping protection to all channels other** than that with the highest RMS level, such that all channels attain the same RMS level

If normalize=True, 'B' and 'b' are equivalent.

See also:

loudness

highpass (*self*, *frequency*, *width_q=0.707*, *n_poles=2*)
Apply a high-pass filter with 3dB point frequency. The filter can be either single-pole or double-pole. The filters roll off at 6dB per pole per octave (20dB per pole per decade).

Parameters

frequency [float] The filter's cutoff frequency in Hz.

width_q [float, default=0.707] The filter's width as a Q-factor. Applies only when n_poles=2. The default gives a Butterworth response.

n_poles [int, default=2] The number of poles in the filter. Must be either 1 or 2

See also:

lowpass, *equalizer*, *sinc*, *allpass*

hilbert (*self*, *num_taps=None*)
Apply an odd-tap Hilbert transform filter, phase-shifting the signal by 90 degrees. This is used in many matrix coding schemes and for analytic signal generation. The process is often written as a multiplication by i (or j), the imaginary unit. An odd-tap Hilbert transform filter has a bandpass characteristic, attenuating the lowest and highest frequencies.

Parameters

num_taps [int or None, default=None] Number of filter taps - must be odd. If none, it is chosen to have a cutoff frequency of about 75 Hz.

loudness (*self*, *gain_db=-10.0*, *reference_level=65.0*)
Loudness control. Similar to the gain effect, but provides equalisation for the human auditory system.

The gain is adjusted by gain_db and the signal is equalised according to ISO 226 w.r.t. reference_level.

Parameters

gain_db [float, default=-10.0] Loudness adjustment amount (in dB)

reference_level [float, default=65.0] Reference level (in dB) according to which the signal is equalized. Must be between 50 and 75 (dB)

See also:

gain

lowpass (*self*, *frequency*, *width_q=0.707*, *n_poles=2*)
Apply a low-pass filter with 3dB point frequency. The filter can be either single-pole or double-pole. The filters roll off at 6dB per pole per octave (20dB per pole per decade).

Parameters

frequency [float] The filter's cutoff frequency in Hz.

width_q [float, default=0.707] The filter's width as a Q-factor. Applies only when n_poles=2. The default gives a Butterworth response.

n_poles [int, default=2] The number of poles in the filter. Must be either 1 or 2

See also:

highpass, *equalizer*, *sinc*, *allpass*

mcompand (*self*, *n_bands=2*, *crossover_frequencies=[1600]*, *attack_time=[0.005, 0.000625]*, *decay_time=[0.1, 0.0125]*, *soft_knee_db=[6.0, None]*, *tf_points=[[(-47, -40), (-34, -34), (-17, -33), (0, 0)], [(-47, -40), (-34, -34), (-15, -33), (0, 0)]]*, *gain=[None, None]*)
The multi-band compander is similar to the single-band compander but the audio is first divided into bands using Linkwitz-Riley cross-over filters and a separately specifiable compander run on each band.

When used with n_bands=1, this effect is identical to compand. When using n_bands > 1, the first set of arguments applies a single band compander, and each subsequent set of arugments is applied on each of the crossover frequencies.

Parameters

n_bands [int, default=2] The number of bands.

crossover_frequencies [list of float, default=[1600]] A list of crossover frequencies in Hz of length n_bands-1. The first band is always the full spectrum, followed by the bands specified by crossover_frequencies.

attack_time [list of float, default=[0.005, 0.000625]] A list of length n_bands, where each element is the time in seconds over which the instantaneous level of the input signal is averaged to determine increases in volume over the current band.

decay_time [list of float, default=[0.1, 0.0125]] A list of length n_bands, where each element is the time in seconds over which the instantaneous level of the input signal is averaged to determine decreases in volume over the current band.

soft_knee_db [list of float or None, default=[6.0, None]] A list of length n_bands, where each element is the ammount (in dB) for which the points at where adjacent line segments on the transfer function meet will be rounded over the current band. If None, no soft_knee is applied.

tf_points [list of list of tuples, default=[]

[(-47, -40), (-34, -34), (-17, -33), (0, 0)], [(-47, -40), (-34, -34), (-15, -33), (0, 0)]]

A list of length n_bands, where each element is the transfer function points as a list of tuples corresponding to points in (dB, dB) defining the compander's transfer function over the current band.

gain [list of floats or None] A list of gain values for each frequency band. If None, no gain is applied.

See also:

`compand`, `contrast`

noiseprof(*self*, *input_filepath*, *profile_path*)

Calculate a profile of the audio for use in noise reduction. Running this command does not effect the Transformer effects chain. When this function is called, the calculated noise profile file is saved to the *profile_path*.

Parameters

input_filepath [str] Path to audiofile from which to compute a noise profile.

profile_path [str] Path to save the noise profile file.

See also:

`noisered`

noisered(*self*, *profile_path*, *amount=0.5*)

Reduce noise in the audio signal by profiling and filtering. This effect is moderately effective at removing consistent background noise such as hiss or hum.

Parameters

profile_path [str] Path to a noise profile file. This file can be generated using the *noiseprof* effect.

amount [float, default=0.5] How much noise should be removed is specified by amount. Should be between 0 and 1. Higher numbers will remove more noise but present a greater likelihood of removing wanted components of the audio signal.

See also:

`noiseprof`

norm (*self*, *db_level=-3.0*)
Normalize an audio file to a particular db level. This behaves identically to the gain effect with normalize=True.

Parameters

db_level [float, default=-3.0] Output volume (db)

See also:

`gain`, `loudness`

oops (*self*)
Out Of Phase Stereo effect. Mixes stereo to twin-mono where each mono channel contains the difference between the left and right stereo channels. This is sometimes known as the 'karaoke' effect as it often has the effect of removing most or all of the vocals from a recording.

overdrive (*self*, *gain_db=20.0*, *colour=20.0*)
Apply non-linear distortion.

Parameters

gain_db [float, default=20] Controls the amount of distortion (dB).

colour [float, default=20] Controls the amount of even harmonic content in the output (dB).

pad (*self*, *start_duration=0.0*, *end_duration=0.0*)
Add silence to the beginning or end of a file. Calling this with the default arguments has no effect.

Parameters

start_duration [float] Number of seconds of silence to add to beginning.

end_duration [float] Number of seconds of silence to add to end.

See also:

`delay`

phaser (*self*, *gain_in=0.8*, *gain_out=0.74*, *delay=3*, *decay=0.4*, *speed=0.5*, *modulation_shape='sinusoidal'*)
Apply a phasing effect to the audio.

Parameters

gain_in [float, default=0.8] Input volume between 0 and 1

gain_out: float, default=0.74 Output volume between 0 and 1

delay [float, default=3] Delay in miliseconds between 0 and 5

decay [float, default=0.4] Decay relative to gain_in, between 0.1 and 0.5.

speed [float, default=0.5] Modulation speed in Hz, between 0.1 and 2

modulation_shape [str, defaul='sinusoidal'] Modulation shpae. One of 'sinusoidal' or 'triangular'

See also:

`flanger`, `tremolo`

pitch (*self*, *n_semitones*, *quick=False*)
Pitch shift the audio without changing the tempo.

This effect uses the WSOLA algorithm. The audio is chopped up into segments which are then shifted in the time domain and overlapped (cross-faded) at points where their waveforms are most similar as determined by measurement of least squares.

Parameters

n_semitones [float] The number of semitones to shift. Can be positive or negative.

quick [bool, default=False] If True, this effect will run faster but with lower sound quality.

See also:

bend, *speed*, *tempo*

power_spectrum (*self*, *input_filepath*)
Calculates the power spectrum (4096 point DFT). This method internally invokes the stat command with the -freq option.

Note: The file is downmixed to mono prior to computation.

Parameters

input_filepath [str] Path to input file to compute stats on.

Returns

power_spectrum [list] List of frequency (Hz), amplitude pairs.

See also:

stat, *stats*, *sox.file_info*

preview (*self*, *input_filepath*)
Play a preview of the output with the current set of effects

Parameters

input_filepath [str] Path to input audio file.

rate (*self*, *samplerate*, *quality='h'*)
Change the audio sampling rate (i.e. resample the audio) to any given *samplerate*. Better the resampling quality = slower runtime.

Parameters

samplerate [float] Desired sample rate.

quality [str]

Resampling quality. One of:

- q : Quick - very low quality,
- l : Low,
- m : Medium,
- h : High (default),
- v : Very high

See also:

upsample, *downsample*, *convert*

remix (*self*, *remix_dictionary=None*, *num_output_channels=None*)
Remix the channels of an audio file.

Note: volume options are not yet implemented

Parameters

remix_dictionary [dict or None] Dictionary mapping output channel to list of input channel(s). Empty lists indicate the corresponding output channel should be empty. If None, mixes all channels down to a single mono file.

num_output_channels [int or None] The number of channels in the output file. If None, the number of output channels is equal to the largest key in remix_dictionary. If remix_dictionary is None, this variable is ignored.

Examples

Remix a 4-channel input file. The output file will have input channel 2 in channel 1, a mixdown of input channels 1 an 3 in channel 2, an empty channel 3, and a copy of input channel 4 in channel 4.

```
>>> import sox
>>> tfm = sox.Transformer()
>>> remix_dictionary = {1: [2], 2: [1, 3], 4: [4]}
>>> tfm.remix(remix_dictionary)
```

repeat (*self*, *count=1*)
Repeat the entire audio count times.

Parameters

count [int, default=1] The number of times to repeat the audio.

reverb (*self*, *reverberance=50*, *high_freq_damping=50*, *room_scale=100*, *stereo_depth=100*, *pre_delay=0*, *wet_gain=0*, *wet_only=False*)
Add reverberation to the audio using the 'freeverb' algorithm. A reverberation effect is sometimes desirable for concert halls that are too small or contain so many people that the hall's natural reverberance is diminished. Applying a small amount of stereo reverb to a (dry) mono signal will usually make it sound more natural.

Parameters

reverberance [float, default=50] Percentage of reverberance

high_freq_damping [float, default=50] Percentage of high-frequency damping.

room_scale [float, default=100] Scale of the room as a percentage.

stereo_depth [float, default=100] Stereo depth as a percentage.

pre_delay [float, default=0] Pre-delay in milliseconds.

wet_gain [float, default=0] Amount of wet gain in dB

wet_only [bool, default=False] If True, only outputs the wet signal.

See also:

echo

reverse (*self*)
Reverse the audio completely

set_globals (*self*, *dither=False*, *guard=False*, *multithread=False*, *replay_gain=False*, *verbosity=2*)
Sets SoX's global arguments. Overwrites any previously set global arguments. If this function is not explicity called, globals are set to this function's defaults.

Parameters

dither [bool, default=False] If True, dithering is applied for low files with low bit rates.

guard [bool, default=False] If True, invokes the gain effect to guard against clipping.

multithread [bool, default=False] If True, each channel is processed in parallel.

replay_gain [bool, default=False] If True, applies replay-gain adjustment to input-files.

verbosity [int, default=2]

SoX's verbosity level. One of:

- 0 : No messages are shown at all
- **1** [Only error messages are shown. These are generated if SoX] cannot complete the requested commands.
- **2** [Warning messages are also shown. These are generated if] SoX can complete the requested commands, but not exactly according to the requested command parameters, or if clipping occurs.
- **3** [Descriptions of SoX's processing phases are also shown.] Useful for seeing exactly how SoX is processing your audio.
- 4, >4 : Messages to help with debugging SoX are also shown.

set_input_format (*self*, *file_type=None*, *rate=None*, *bits=None*, *channels=None*, *encoding=None*, *ignore_length=False*)
Sets input file format arguments. This is primarily useful when dealing with audio files without a file extension. Overwrites any previously set input file arguments.

If this function is not explicity called the input format is inferred from the file extension or the file's header.

Parameters

file_type [str or None, default=None] The file type of the input audio file. Should be the same as what the file extension would be, for ex. 'mp3' or 'wav'.

rate [float or None, default=None] The sample rate of the input audio file. If None the sample rate is inferred.

bits [int or None, default=None] The number of bits per sample. If None, the number of bits per sample is inferred.

channels [int or None, default=None] The number of channels in the audio file. If None the number of channels is inferred.

encoding [str or None, default=None] The audio encoding type. Sometimes needed with file-types that support more than one encoding type. One of:

- **signed-integer** [PCM data stored as signed ('two's] complement') integers. Commonly used with a 16 or 24bit encoding size. A value of 0 represents minimum signal power.

- **unsigned-integer** [PCM data stored as unsigned integers.] Commonly used with an 8-bit encoding size. A value of 0 represents maximum signal power.
- **floating-point** [PCM data stored as IEEE 753 single precision] (32-bit) or double precision (64-bit) floating-point ('real') numbers. A value of 0 represents minimum signal power.
- **a-law** [International telephony standard for logarithmic] encoding to 8 bits per sample. It has a precision equivalent to roughly 13-bit PCM and is sometimes encoded with reversed bit-ordering.
- **u-law** [North American telephony standard for logarithmic] encoding to 8 bits per sample. A.k.a. μ-law. It has a precision equivalent to roughly 14-bit PCM and is sometimes encoded with reversed bit-ordering.
- **oki-adpcm** [OKI (a.k.a. VOX, Dialogic, or Intel) 4-bit ADPCM;] it has a precision equivalent to roughly 12-bit PCM. ADPCM is a form of audio compression that has a good compromise between audio quality and encoding/decoding speed.
- **ima-adpcm** [IMA (a.k.a. DVI) 4-bit ADPCM; it has a precision] equivalent to roughly 13-bit PCM.
- **ms-adpcm** [Microsoft 4-bit ADPCM; it has a precision] equivalent to roughly 14-bit PCM.
- **gsm-full-rate** [GSM is currently used for the vast majority] of the world's digital wireless telephone calls. It utilises several audio formats with different bit-rates and associated speech quality. SoX has support for GSM's original 13kbps 'Full Rate' audio format. It is usually CPU-intensive to work with GSM audio.

ignore_length [bool, default=False] If True, overrides an (incorrect) audio length given in an audio file's header. If this option is given then SoX will keep reading audio until it reaches the end of the input file.

set_output_format(*self*, *file_type=None*, *rate=None*, *bits=None*, *channels=None*, *encoding=None*, *comments=None*, *append_comments=True*)

Sets output file format arguments. These arguments will overwrite any format related arguments supplied by other effects (e.g. rate).

If this function is not explicity called the output format is inferred from the file extension or the file's header.

Parameters

file_type [str or None, default=None] The file type of the output audio file. Should be the same as what the file extension would be, for ex. 'mp3' or 'wav'.

rate [float or None, default=None] The sample rate of the output audio file. If None the sample rate is inferred.

bits [int or None, default=None] The number of bits per sample. If None, the number of bits per sample is inferred.

channels [int or None, default=None] The number of channels in the audio file. If None the number of channels is inferred.

encoding [str or None, default=None] The audio encoding type. Sometimes needed with file-types that support more than one encoding type. One of:

- **signed-integer** [PCM data stored as signed ('two's] complement') integers. Commonly used with a 16 or 24bit encoding size. A value of 0 represents minimum signal power.
- **unsigned-integer** [PCM data stored as unsigned integers.] Commonly used with an 8-bit encoding size. A value of 0 represents maximum signal power.
- **floating-point** [PCM data stored as IEEE 753 single precision] (32-bit) or double precision (64-bit) floating-point ('real') numbers. A value of 0 represents minimum signal power.
- **a-law** [International telephony standard for logarithmic] encoding to 8 bits per sample. It has a precision equivalent to roughly 13-bit PCM and is sometimes encoded with reversed bit-ordering.
- **u-law** [North American telephony standard for logarithmic] encoding to 8 bits per sample. A.k.a. μ-law. It has a precision equivalent to roughly 14-bit PCM and is sometimes encoded with reversed bit-ordering.
- **oki-adpcm** [OKI (a.k.a. VOX, Dialogic, or Intel) 4-bit ADPCM;] it has a precision equivalent to roughly 12-bit PCM. ADPCM is a form of audio compression that has a good compromise between audio quality and encoding/decoding speed.
- **ima-adpcm** [IMA (a.k.a. DVI) 4-bit ADPCM; it has a precision] equivalent to roughly 13-bit PCM.
- **ms-adpcm** [Microsoft 4-bit ADPCM; it has a precision] equivalent to roughly 14-bit PCM.
- **gsm-full-rate** [GSM is currently used for the vast majority] of the world's digital wireless telephone calls. It utilises several audio formats with different bit-rates and associated speech quality. SoX has support for GSM's original 13kbps 'Full Rate' audio format. It is usually CPU-intensive to work with GSM audio.

comments [str or None, default=None] If not None, the string is added as a comment in the header of the output audio file. If None, no comments are added.

append_comments [bool, default=True] If True, comment strings are appended to SoX's default comments. If False, the supplied comment replaces the existing comment.

silence (*self*, *location=0*, *silence_threshold=0.1*, *min_silence_duration=0.1*, *buffer_around_silence=False*)

Removes silent regions from an audio file.

Parameters

location [int, default=0]

Where to remove silence. One of:

- 0 to remove silence throughout the file (default),
- 1 to remove silence from the beginning,
- -1 to remove silence from the end,

silence_threshold [float, default=0.1] Silence threshold as percentage of maximum sample amplitude. Must be between 0 and 100.

min_silence_duration [float, default=0.1] The minimum ammount of time in seconds required for a region to be considered non-silent.

buffer_around_silence [bool, default=False] If True, leaves a buffer of min_silence_duration around removed silent regions.

See also:

vad

sinc (*self*, *filter_type='high'*, *cutoff_freq=3000*, *stop_band_attenuation=120*, *transition_bw=None*, *phase_response=None*)

Apply a sinc kaiser-windowed low-pass, high-pass, band-pass, or band-reject filter to the signal.

Parameters

filter_type [str, default='high']

Type of filter. One of:

- 'high' for a high-pass filter
- 'low' for a low-pass filter
- 'pass' for a band-pass filter
- 'reject' for a band-reject filter

cutoff_freq [float or list, default=3000] A scalar or length 2 list indicating the filter's critical frequencies. The critical frequencies are given in Hz and must be positive. For a high-pass or low-pass filter, cutoff_freq must be a scalar. For a band-pass or band-reject filter, it must be a length 2 list.

stop_band_attenuation [float, default=120] The stop band attenuation in dB

transition_bw [float, list or None, default=None] The transition band-width in Hz. If None, sox's default of 5% of the total bandwith is used. If a float, the given transition bandwith is used for both the upper and lower bands (if applicable). If a list, the first argument is used for the lower band and the second for the upper band.

phase_response [float or None] The filter's phase response between 0 (minimum) and 100 (maximum). If None, sox's default phase repsonse is used.

See also:

band, *bandpass*, *bandreject*, *highpass*, *lowpass*

speed (*self*, *factor*)

Adjust the audio speed (pitch and tempo together).

Technically, the speed effect only changes the sample rate information, leaving the samples themselves untouched. The rate effect is invoked automatically to resample to the output sample rate, using its default quality/speed. For higher quality or higher speed resampling, in addition to the speed effect, specify the rate effect with the desired quality option.

Parameters

factor [float] The ratio of the new speed to the old speed. For ex. 1.1 speeds up the audio by 10%; 0.9 slows it down by 10%. Note - this argument is the inverse of what is passed to the sox stretch effect for consistency with speed.

See also:

rate, *tempo*, *pitch*

stat (*self*, *input_filepath*, *scale=None*, *rms=False*)

Display time and frequency domain statistical information about the audio. Audio is passed unmodified through the SoX processing chain.

Unlike other Transformer methods, this does not modify the transformer effects chain. Instead it computes statistics on the output file that would be created if the build command were invoked.

Note: The file is downmixed to mono prior to computation.

Parameters

input_filepath [str] Path to input file to compute stats on.

scale [float or None, default=None] If not None, scales the input by the given scale factor.

rms [bool, default=False] If True, scales all values by the average rms amplitude.

Returns

stat_dict [dict] Dictionary of statistics.

See also:

stats, *power_spectrum*, *sox.file_info*

stats (*self*, *input_filepath*)

Display time domain statistical information about the audio channels. Audio is passed unmodified through the SoX processing chain. Statistics are calculated and displayed for each audio channel

Unlike other Transformer methods, this does not modify the transformer effects chain. Instead it computes statistics on the output file that would be created if the build command were invoked.

Note: The file is downmixed to mono prior to computation.

Parameters

input_filepath [str] Path to input file to compute stats on.

Returns

stats_dict [dict] List of frequency (Hz), amplitude pairs.

See also:

stat, *sox.file_info*

stretch (*self*, *factor*, *window=20*)

Change the audio duration (but not its pitch). **Unless factor is close to 1, use the tempo effect instead.**

This effect is broadly equivalent to the tempo effect with search set to zero, so in general, its results are comparatively poor; it is retained as it can sometimes out-perform tempo for small factors.

Parameters

factor [float] The ratio of the new tempo to the old tempo. For ex. 1.1 speeds up the tempo by 10%; 0.9 slows it down by 10%. Note - this argument is the inverse of what is passed to the sox stretch effect for consistency with tempo.

window [float, default=20] Window size in miliseconds

See also:

tempo, *speed*, *pitch*

swap (*self*)
Swap stereo channels. If the input is not stereo, pairs of channels are swapped, and a possible odd last channel passed through.

E.g., for seven channels, the output order will be 2, 1, 4, 3, 6, 5, 7.

See also:

remix

tempo (*self, factor, audio_type=None, quick=False*)
Time stretch audio without changing pitch.

This effect uses the WSOLA algorithm. The audio is chopped up into segments which are then shifted in the time domain and overlapped (cross-faded) at points where their waveforms are most similar as determined by measurement of least squares.

Parameters

factor [float] The ratio of new tempo to the old tempo. For ex. 1.1 speeds up the tempo by 10%; 0.9 slows it down by 10%.

audio_type [str]

Type of audio, which optimizes algorithm parameters. One of:

- m : Music,
- s : Speech,
- l : Linear (useful when factor is close to 1),

quick [bool, default=False] If True, this effect will run faster but with lower sound quality.

See also:

stretch, *speed*, *pitch*

treble (*self, gain_db, frequency=3000.0, slope=0.5*)
Boost or cut the treble (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation.

The filters are described in detail in http://musicdsp.org/files/Audio-EQ-Cookbook.txt

Parameters

gain_db [float] The gain at the Nyquist frequency. For a large cut use -20, for a large boost use 20.

frequency [float, default=100.0] The filter's cutoff frequency in Hz.

slope [float, default=0.5] The steepness of the filter's shelf transition. For a gentle slope use 0.3, and use 1.0 for a steep slope.

See also:

bass, *equalizer*

tremolo (*self, speed=6.0, depth=40.0*)
Apply a tremolo (low frequency amplitude modulation) effect to the audio. The tremolo frequency in Hz is giv en by speed, and the depth as a percentage by depth (default 40).

Parameters

speed [float] Tremolo speed in Hz.

depth [float] Tremolo depth as a percentage of the total amplitude.

See also:

flanger

Examples

```
>>> tfm = sox.Transformer()
```

For a growl-type effect

```
>>> tfm.tremolo(speed=100.0)
```

trim(*self*, *start_time*, *end_time=None*)

Excerpt a clip from an audio file, given the start timestamp and end timestamp of the clip within the file, expressed in seconds. If the end timestamp is set to *None* or left unspecified, it defaults to the duration of the audio file.

Parameters

start_time [float] Start time of the clip (seconds)

end_time [float or None, default=None] End time of the clip (seconds)

upsample(*self*, *factor=2*)

Upsample the signal by an integer factor: zero-value samples are inserted between each pair of input samples. As a result, the original spectrum is replicated into the new frequency space (imaging) and attenuated. The upsample effect is typically used in combination with filtering effects.

Parameters

factor [int, default=2] Integer upsampling factor.

See also:

rate, *downsample*

vad(*self*, *location=1*, *normalize=True*, *activity_threshold=7.0*, *min_activity_duration=0.25*, *initial_search_buffer=1.0*, *max_gap=0.25*, *initial_pad=0.0*)

Voice Activity Detector. Attempts to trim silence and quiet background sounds from the ends of recordings of speech. The algorithm currently uses a simple cepstral power measurement to detect voice, so may be fooled by other things, especially music.

The effect can trim only from the front of the audio, so in order to trim from the back, the reverse effect must also be used.

Parameters

location [1 or -1, default=1] If 1, trims silence from the beginning If -1, trims silence from the end

normalize [bool, default=True] If true, normalizes audio before processing.

activity_threshold [float, default=7.0] The measurement level used to trigger activity detection. This may need to be cahnged depending on the noise level, signal level, and other characteristics of the input audio.

min_activity_duration [float, default=0.25] The time constant (in seconds) used to help ignore short bursts of sound.

initial_search_buffer [float, default=1.0] The amount of audio (in seconds) to search for quieter/shorter bursts of audio to include prior to the detected trigger point.

max_gap [float, default=0.25] The allowed gap (in seconds) between quiteter/shorter bursts of audio to include prior to the detected trigger point

initial_pad [float, default=0.0] The amount of audio (in seconds) to preserve before the trigger point and any found quieter/shorter bursts.

See also:

silence

Examples

```
>>> tfm = sox.Transformer()
```

Remove silence from the beginning of speech

```
>>> tfm.vad(initial_pad=0.3)
```

Remove silence from the end of speech

```
>>> tfm.vad(location=-1, initial_pad=0.2)
```

vol (*self*, *gain*, *gain_type='amplitude'*, *limiter_gain=None*)
Apply an amplification or an attenuation to the audio signal.

Parameters

gain [float] Interpreted according to the given *gain_type*. If 'gain_type' = 'amplitude', 'gain' is a positive amplitude ratio. If 'gain_type' = 'power', 'gain' is a power (voltage squared). If 'gain_type' = 'db', 'gain' is in decibels.

gain_type [string, default='amplitude']

Type of gain. One of:

- 'amplitude'
- 'power'
- 'db'

limiter_gain [float or None, default=None] If specified, a limiter is invoked on peaks greater than *limiter_gain' to prevent clipping. 'limiter_gain* should be a positive value much less than 1.

See also:

gain, *compand*

3.2 Combiners

Python wrapper around the SoX library. This module requires that SoX is installed.

class sox.combine.**Combiner**

Audio file combiner. Class which allows multiple files to be combined to create an output file, saved to output_filepath.

Inherits all methods from the Transformer class, thus any effects can be applied after combining.

Methods

allpass(self, frequency[, width_q])	Apply a two-pole all-pass filter.
bandpass(self, frequency[, width_q, ...])	Apply a two-pole Butterworth band-pass filter with the given central frequency, and (3dB-point) band-width.
bandreject(self, frequency[, width_q])	Apply a two-pole Butterworth band-reject filter with the given central frequency, and (3dB-point) band-width.
bass(self, gain_db[, frequency, slope])	Boost or cut the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls.
bend(self, n_bends, start_times, end_times, ...)	Changes pitch by specified amounts at specified times.
biquad(self, b, a)	Apply a biquad IIR filter with the given coefficients.
build(self, input_filepath_list, ... [, ...])	Builds the output_file by executing the current set of commands.
build_array(self[, input_filepath, ...])	Given an input file or array, returns the ouput as a numpy array by executing the current set of commands.
build_file(self[, input_filepath, ...])	An alias for build.
channels(self, n_channels)	Change the number of channels in the audio signal.
chorus(self[, gain_in, gain_out, n_voices, ...])	Add a chorus effect to the audio.
clear_effects(self)	Remove all effects processes.
compand(self[, attack_time, decay_time, ...])	Compand (compress or expand) the dynamic range of the audio.
contrast(self[, amount])	Comparable with compression, this effect modifies an audio signal to make it sound louder.
convert(self[, samplerate, n_channels, bitdepth])	Converts output audio to the specified format.
dcshift(self[, shift])	Apply a DC shift to the audio.
deemph(self)	Apply Compact Disc (IEC 60908) de-emphasis (a treble attenuation shelving filter).
delay(self, positions)	Delay one or more audio channels such that they start at the given positions.
downsample(self[, factor])	Downsample the signal by an integer factor.
earwax(self)	Makes audio easier to listen to on headphones.
echo(self[, gain_in, gain_out, n_echos, ...])	Add echoing to the audio.
echos(self[, gain_in, gain_out, n_echos, ...])	Add a sequence of echoes to the audio.
equalizer(self, frequency, width_q, gain_db)	Apply a two-pole peaking equalisation (EQ) filter to boost or reduce around a given frequency.
fade(self[, fade_in_len, fade_out_len, ...])	Add a fade in and/or fade out to an audio file.
fir(self, coefficients)	Use SoX's FFT convolution engine with given FIR filter coefficients.
flanger(self[, delay, depth, regen, width, ...])	Apply a flanging effect to the audio.

Continued on next page

Table 2 – continued from previous page

gain(self[, gain_db, normalize, limiter, …])	Apply amplification or attenuation to the audio signal.
highpass(self, frequency[, width_q, n_poles])	Apply a high-pass filter with 3dB point frequency.
hilbert(self[, num_taps])	Apply an odd-tap Hilbert transform filter, phase-shifting the signal by 90 degrees.
loudness(self[, gain_db, reference_level])	Loudness control.
lowpass(self, frequency[, width_q, n_poles])	Apply a low-pass filter with 3dB point frequency.
mcompand(self[, n_bands, …])	The multi-band compander is similar to the single-band compander but the audio is first divided into bands using Linkwitz-Riley cross-over filters and a separately specifiable compander run on each band.
noiseprof(self, input_filepath, profile_path)	Calculate a profile of the audio for use in noise reduction.
noisered(self, profile_path[, amount])	Reduce noise in the audio signal by profiling and filtering.
norm(self[, db_level])	Normalize an audio file to a particular db level.
oops(self)	Out Of Phase Stereo effect.
overdrive(self[, gain_db, colour])	Apply non-linear distortion.
pad(self[, start_duration, end_duration])	Add silence to the beginning or end of a file.
phaser(self[, gain_in, gain_out, delay, …])	Apply a phasing effect to the audio.
pitch(self, n_semitones[, quick])	Pitch shift the audio without changing the tempo.
power_spectrum(self, input_filepath)	Calculates the power spectrum (4096 point DFT).
preview(self, input_filepath_list, combine_type)	Play a preview of the output with the current set of effects
rate(self, samplerate[, quality])	Change the audio sampling rate (i.e.
remix(self[, remix_dictionary, …])	Remix the channels of an audio file.
repeat(self[, count])	Repeat the entire audio count times.
reverb(self[, reverberance, …])	Add reverberation to the audio using the 'freeverb' algorithm.
reverse(self)	Reverse the audio completely
set_globals(self[, dither, guard, …])	Sets SoX's global arguments.
set_input_format(self[, file_type, rate, …])	Sets input file format arguments.
set_output_format(self[, file_type, rate, …])	Sets output file format arguments.
silence(self[, location, silence_threshold, …])	Removes silent regions from an audio file.
sinc(self[, filter_type, cutoff_freq, …])	Apply a sinc kaiser-windowed low-pass, high-pass, band-pass, or band-reject filter to the signal.
speed(self, factor)	Adjust the audio speed (pitch and tempo together).
stat(self, input_filepath[, scale, rms])	Display time and frequency domain statistical information about the audio.
stats(self, input_filepath)	Display time domain statistical information about the audio channels.
stretch(self, factor[, window])	Change the audio duration (but not its pitch).
swap(self)	Swap stereo channels.
tempo(self, factor[, audio_type, quick])	Time stretch audio without changing pitch.
treble(self, gain_db[, frequency, slope])	Boost or cut the treble (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls.
tremolo(self[, speed, depth])	Apply a tremolo (low frequency amplitude modulation) effect to the audio.

Continued on next page

Table 2 – continued from previous page

`trim`(self, start_time[, end_time])	Excerpt a clip from an audio file, given the start timestamp and end timestamp of the clip within the file, expressed in seconds.
`upsample`(self[, factor])	Upsample the signal by an integer factor: zero-value samples are inserted between each pair of input samples.
`vad`(self[, location, normalize, . . .])	Voice Activity Detector.
`vol`(self, gain[, gain_type, limiter_gain])	Apply an amplification or an attenuation to the audio signal.

build(*self*, *input_filepath_list*, *output_filepath*, *combine_type*, *input_volumes=None*)
Builds the output_file by executing the current set of commands.

Parameters

input_filepath_list [list of str] List of paths to input audio files.

output_filepath [str] Path to desired output file. If a file already exists at the given path, the file will be overwritten.

combine_type [str]

Input file combining method. One of the following values:

- **concatenate** [combine input files by concatenating in the] order given.
- **merge** [combine input files by stacking each input file into] a new channel of the output file.
- **mix** [combine input files by summing samples in corresponding] channels.
- **mix-power** [combine input files with volume adjustments such] that the output volume is roughly equivlent to one of the input signals.
- **multiply** [combine input files by multiplying samples in] corresponding samples.

input_volumes [list of float, default=None] List of volumes to be applied upon combining input files. Volumes are applied to the input files in order. If None, input files will be combined at their original volumes.

Returns

status [bool] True on success.

preview(*self*, *input_filepath_list*, *combine_type*, *input_volumes=None*)
Play a preview of the output with the current set of effects

Parameters

input_filepath_list [list of str] List of paths to input audio files.

combine_type [str]

Input file combining method. One of the following values:

- **concatenate** [combine input files by concatenating in the] order given.
- **merge** [combine input files by stacking each input file into] a new channel of the output file.
- **mix** [combine input files by summing samples in corresponding] channels.

- **mix-power** [combine input files with volume adjustments such] that the output volume is roughly equivlent to one of the input signals.
- **multiply** [combine input files by multiplying samples in] corresponding samples.

input_volumes [list of float, default=None] List of volumes to be applied upon combining input files. Volumes are applied to the input files in order. If None, input files will be combined at their original volumes.

set_input_format (*self*, *file_type=None*, *rate=None*, *bits=None*, *channels=None*, *encoding=None*, *ignore_length=None*)

Sets input file format arguments. This is primarily useful when dealing with audio files without a file extension. Overwrites any previously set input file arguments.

If this function is not explicity called the input format is inferred from the file extension or the file's header.

Parameters

file_type [list of str or None, default=None] The file type of the input audio file. Should be the same as what the file extension would be, for ex. 'mp3' or 'wav'.

rate [list of float or None, default=None] The sample rate of the input audio file. If None the sample rate is inferred.

bits [list of int or None, default=None] The number of bits per sample. If None, the number of bits per sample is inferred.

channels [list of int or None, default=None] The number of channels in the audio file. If None the number of channels is inferred.

encoding [list of str or None, default=None] The audio encoding type. Sometimes needed with file-types that support more than one encoding type. One of:

- **signed-integer** [PCM data stored as signed ('two's] complement') integers. Commonly used with a 16 or 24bit encoding size. A value of 0 represents minimum signal power.
- **unsigned-integer** [PCM data stored as unsigned integers.] Commonly used with an 8-bit encoding size. A value of 0 represents maximum signal power.
- **floating-point** [PCM data stored as IEEE 753 single precision] (32-bit) or double precision (64-bit) floating-point ('real') numbers. A value of 0 represents minimum signal power.
- **a-law** [International telephony standard for logarithmic] encoding to 8 bits per sample. It has a precision equivalent to roughly 13-bit PCM and is sometimes encoded with reversed bit-ordering.
- **u-law** [North American telephony standard for logarithmic] encoding to 8 bits per sample. A.k.a. μ-law. It has a precision equivalent to roughly 14-bit PCM and is sometimes encoded with reversed bit-ordering.
- **oki-adpcm** [OKI (a.k.a. VOX, Dialogic, or Intel) 4-bit ADPCM;] it has a precision equivalent to roughly 12-bit PCM. ADPCM is a form of audio compression that has a good compromise between audio quality and encoding/decoding speed.
- **ima-adpcm** [IMA (a.k.a. DVI) 4-bit ADPCM; it has a precision] equivalent to roughly 13-bit PCM.

- **ms-adpcm** [Microsoft 4-bit ADPCM; it has a precision] equivalent to roughly 14-bit PCM.
- **gsm-full-rate** [GSM is currently used for the vast majority] of the world's digital wireless telephone calls. It utilises several audio formats with different bit-rates and associated speech quality. SoX has support for GSM's original 13kbps 'Full Rate' audio format. It is usually CPU-intensive to work with GSM audio.

ignore_length [list of bool or None, default=None] If True, overrides an (incorrect) audio length given in an audio file's header. If this option is given then SoX will keep reading audio until it reaches the end of the input file.

3.3 File info

Audio file info computed by soxi.

`sox.file_info.`**`bitdepth`**`(`*input_filepath*`)`
Number of bits per sample, or None if not applicable.

Parameters

input_filepath [str] Path to audio file.

Returns

bitdepth [int or None] Number of bits per sample. Returns None if not applicable.

`sox.file_info.`**`bitrate`**`(`*input_filepath*`)`
Bit rate averaged over the whole file. Expressed in bytes per second (bps), or None if not applicable.

Parameters

input_filepath [str] Path to audio file.

Returns

bitrate [float or None] Bit rate, expressed in bytes per second. Returns None if not applicable.

`sox.file_info.`**`channels`**`(`*input_filepath*`)`
Show number of channels.

Parameters

input_filepath [str] Path to audio file.

Returns

channels [int] number of channels

`sox.file_info.`**`comments`**`(`*input_filepath*`)`
Show file comments (annotations) if available.

Parameters

input_filepath [str] Path to audio file.

Returns

comments [str] File comments from header. If no comments are present, returns an empty string.

`sox.file_info.`**`duration`**`(`*input_filepath*`)`
Show duration in seconds, or None if not available.

Parameters

input_filepath [str] Path to audio file.

Returns

duration [float or None] Duration of audio file in seconds. If unavailable or empty, returns None.

`sox.file_info.`**`encoding`**`(`*input_filepath*`)`

Show the name of the audio encoding.

Parameters

input_filepath [str] Path to audio file.

Returns

encoding [str] audio encoding type

`sox.file_info.`**`file_extension`**`(`*filepath*`)`

Get the extension of a filepath.

Parameters

filepath [str] File path.

Returns

extension [str] The file's extension

`sox.file_info.`**`file_type`**`(`*input_filepath*`)`

Show detected file-type.

Parameters

input_filepath [str] Path to audio file.

Returns

file_type [str] file format type (ex. 'wav')

`sox.file_info.`**`info`**`(`*filepath*`)`

Get a dictionary of file information

Parameters

filepath [str] File path.

Returns

info_dictionary [dict]

Dictionary of file information. Fields are:

- channels
- sample_rate
- bitdepth
- bitrate
- duration
- num_samples
- encoding
- silent

`sox.file_info.`**`num_samples`**`(`*input_filepath*`)`

Show number of samples, or None if unavailable.

Parameters

input_filepath [str] Path to audio file.

Returns

n_samples [int or None] total number of samples in audio file. Returns None if empty or unavailable.

`sox.file_info.sample_rate(input_filepath)`

Show sample-rate.

Parameters

input_filepath [str] Path to audio file.

Returns

samplerate [float] number of samples/second

`sox.file_info.silent(input_filepath, threshold=0.001)`

Determine if an input file is silent.

Parameters

input_filepath [str] The input filepath.

threshold [float] Threshold for determining silence

Returns

is_silent [bool] True if file is determined silent.

`sox.file_info.stat(filepath)`

Returns a dictionary of audio statistics.

Parameters

filepath [str] File path.

Returns

stat_dictionary [dict] Dictionary of audio statistics.

`sox.file_info.validate_input_file(input_filepath)`

Input file validation function. Checks that file exists and can be processed by SoX.

Parameters

input_filepath [str] The input filepath.

`sox.file_info.validate_input_file_list(input_filepath_list)`

Input file list validation function. Checks that object is a list and contains valid filepaths that can be processed by SoX.

Parameters

input_filepath_list [list] A list of filepaths.

`sox.file_info.validate_output_file(output_filepath)`

Output file validation function. Checks that file can be written, and has a valid file extension. Throws a warning if the path already exists, as it will be overwritten on build.

Parameters

output_filepath [str] The output filepath.

3.4 Core functionality

Base module for calling SoX

exception sox.core.**SoxError**(**args*, ***kwargs*)
Exception to be raised when SoX exits with non-zero status.

exception sox.core.**SoxiError**(**args*, ***kwargs*)
Exception to be raised when SoXI exits with non-zero status.

sox.core.**all_equal**(*list_of_things*)
Check if a list contains identical elements.

Parameters

list_of_things [list] list of objects

Returns

all_equal [bool] True if all list elements are the same.

sox.core.**is_number**(*var*)
Check if variable is a numeric value.

Parameters

var [object]

Returns

is_number [bool] True if var is numeric, False otherwise.

sox.core.**play**(*args*)
Pass an argument list to play.

Parameters

args [iterable] Argument list for play. The first item can, but does not need to, be 'play'.

Returns

status [bool] True on success.

sox.core.**sox**(*args*, *src_array=None*, *decode_out_with_utf=True*)
Pass an argument list to SoX.

Parameters

args [iterable] Argument list for SoX. The first item can, but does not need to, be 'sox'.

src_array [np.ndarray, or None] If src_array is not None, then we make sure it's a numpy array and pass it into stdin.

decode_out_with_utf [bool, default=True] Whether or not sox is outputting a bytestring that should be decoded with utf-8.

Returns

status [bool] True on success.

out [str, np.ndarray, or None] Returns a np.ndarray if src_array was an np.ndarray. Returns the stdout produced by sox if src_array is None. Otherwise, returns None if there's an error.

err [str, or None] Returns stderr as a string.

sox.core.**soxi**(*filepath*, *argument*)
Base call to SoXI.

Parameters

filepath [str] Path to audio file.

argument [str] Argument to pass to SoXI.

Returns

shell_output [str] Command line output of SoXI

CHAPTER 4

Changes

4.1 Changes

4.1.1 v1.4.0

- added *.build_array()* which supports file or in memory inputs and array outputs
- added *.build_file()* - an alias to *.build()*
- refactored *.build()* function to support file or in-memory array inputs and file outputs
- the call to subprocess calls the binary directly (shell=False)
- file_info methods return None instead of 0 when the value is not available
- fixed bug in *file_info.bitrate()*, which was returning bitdepth
- added *file_info.bitdepth()*
- added Windows support for *soxi*
- added configurable logging
- *.trim()* can be called with only the start time specificed

4.1.2 v1.3.0

- patched core sox call to work on Windows
- added remix
- added gain to mcompand
- fixed scientific notation format bug
- allow null output filepaths in *build*
- added ability to capture *build* outputs to stdout and stderr

- added *power_spectrum*
- added *stat*
- added *clear* method
- added *noiseprof* and *noisered* effects
- added *vol* effect
- fixed *Combiner.preview()*

4.1.3 v1.1.8

- Move specification of input/output file arguments from __init__ to .build()

4.1.4 v0.1

- Initial release.

CHAPTER 5

Contribute

- Issue Tracker
- Source Code

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

CHAPTER 7

pysox vs sox

The original command line tool is called SoX

This project (the github repository) is called pysox

The library within python is called *sox*. It can be installed via:

```
$ pip install sox
```

and imported within Python as

```
import sox
```

Python Module Index

S

Index

O

P

R

S

T

U

V