Audiostream Documentation

Release 0.2-alpha

Mathieu Virbel, Dustin Lacewell

August 08, 2015

Contents

Audiostream is a Python extension that provide an easy-to-use API for streaming bytes to the speaker, or read an audio input stream. It use SDL + SDL_Mixer for streaming the audio out, and use platform-api for reading audio input.

This extension works on Android and iOS as well.

CHAPTER 1

Examples

Example to generate a wave form using sin() method, and stream to the speaker:

```
from time import sleep
from audiostream import get_output
from audiostream.sources.wave import SineSource

# get a output stream where we can play samples
stream = get_output(channels=2, rate=22050, buffersize=1024)

# create one wave sin() at 440Hz, attach it to our speaker, and play
sinsource = SineSource(stream, 440)
sinsource.start()

# you can change the frequency of the source during the playtime
for x in xrange(10):
    sinsource.frequency = 440 + x
    sleep(.5)

# ok we are done, stop everything.
sinsource.stop()
```

Example to read microphone bytes:

```
from audiostream import get_input

# declare a callback where we'll receive the data
def callback_mic(data):
    print 'i got', len(data)

# get the microphone (or from another source if available)
mic = get_input(callback=callback_mic)
mic.start()
sleep(5)
mic.stop()
```

Note: To be able to record microphone on Android, you need to have the RECORD_AUDIO permission

CHAPTER 2

API

2.1 Core API

audiostream.**get_output**(*rate : int*, *channels : int*, *encoding : int*) → 'AudioOutput' instance

Initialize the engine and get an output device. This method can be used only once, usually at the start of your application.

Parameters

- **rate** (*integer*) – Rate of the audio, default to 44100
- **channels** (*integer*) – Number of channels, minimum 1, default to 2
- **encoding** (*integer*) – Encoding of the audio stream, can be 8 or 16, default to 16
- **buffersize** (*integer*) – Size of the output buffer. Tiny buffer will use consume more CPU, but will be more reactive.

Return type *AudioOutput* instance

Example:

```
from audiostream import get_output
stream = get_output(channels=2, rate=22050, buffersize=1024)
```

audiostream.**get_input**(*callback : callable*, *source : string*, *rate : int*, *channels : int*, *encoding : int*, *buffersize : int*) → 'AudioInput' instance

Return an *AudioInput* instance. All the data received from the input will be stored in a queue. You need to *AudioInput.poll()* the queue regulary in order to trigger the callback.

Please note that the *callback* will be called in the same thread as the one that call *AudioInput.poll()*.

Parameters

- **callback** (*callable*) – Callback to call when bytes are available on the input, called from the audio thread.
- **source** (*string*) – Source device to read, default to 'default. Depending of the platform, you might read other input source. Check the *get_input_sources()* function.
- **channels** (*integer*) – Number of channels, minimum 1, default to 2
- **encoding** (*integer*) – Encoding of the audio stream, can be 8 or 16, default to 16
- **buffersize** (*integer*) – Size of the input buffer. If <= 0, it will be automatically sized by the system.

Return type *AudioInput* instance

Example:

```
from audiostream import get_input

def mic_callback(buf):
    print 'got', len(buf)

# get the default audio input (mic on most cases)
mic = get_input(callback=mic_callback)
mic.start()

while not quit:
    mic.poll()
    # do something here, like sleep(2)

mic.stop()
```

Note: This function currently work only on Android and iOS.

audiostream.**get_input_sources**() → list of strings

Return a list of available input sources. This list is platform-dependent. You might need some additionnals permissions in order to access to the sources.

- android: 'camcorder', 'default', 'mic', 'voice_call', 'voice_communication', 'voice_downlink', 'voice_recognition', 'voice_uplink'
- ios: 'default'

Note: This function currently work only on Android and iOS.

class audiostream.**AudioInput**(*object*)

Abstract class for handling an audio input. Normally, the default audio source is the microphone. It will be recorded with a rate of 44100hz, mono, with 16bit PCM. Theses defaults are the most used and guaranted to work on Android and iOS. Any others combination might fail.

start()

Start the input to gather data from the source

stop()

Stop the input to gather data from the source

poll()

Read the internal queue and dispatch the data through the callback

callback

Callback to call when bytes are available on the input, called from the audio thread. The callback must have one parameter for receiving the data.

encoding

(readonly) Encoding of the audio, can be 8 or 16, default to 16

source

(readonly) Source device to read, default to 'default. Depending of the platform, you might read other input source. Check the *get_input_sources()* function.

channels

(readonly) Number of channels, minimum 1, default to 2

buffersize

(readonly) Size of the input buffer. If <= 0, it will be automatically sized by the system.

class audiostream.**AudioOutput**(*object*)

Abstract class for handling audio output stream, and handle the mixing of multiple sample. One sample is an instance of *AudioSample* abstract class. You can implement your own sample that generate bytes, and thoses bytes will be mixed in the final output stream.

We also expose multiple *AudioSample* implementation, such as:

- *audiostream.sources.thread.ThreadSource*: base for implementing a generator that run in a thread
- *audiostream.sources.wave.SineSource*: generate a sine wave
- *audiostream.sources.puredata.PatchSource*: sample generator that use a Puredata patch (require pylibpd)

add_sample(*sample : AudioSample*)

Parameters **sample** (*AudioSample*) – sample to manage in the mixer

Add a sample to manage in the internal mixer. This method is usually called in the AudioSample.start()

remove_sample(*sample : AudioSample*)

Parameters **sample** (*AudioSample*) – sample managed by the mixer

Remove a sample from the internal mixer. This method is usually called in the *AudioSample.stop()*

class audiostream.**AudioSample**

AudioSample is a class for generating bytes that will be consumed by AudioOutput'. The data goes first on a RingBuffer, and the buffer is consumed by the speaker, according to the :class:`AudioOutput initialization.

Example:

```
from audiostream import get_output, AudioSample
stream = get_output(channels=1, buffersize=1024, rate=22050)
sample = AudioSample()
stream.add_sample(sample)

sample.play()
while True:
    # audio stuff, this is not accurate.
    sample.write("\\x00\\x00\\x00\\x00\\xff\\xff\\xff\\xff")
```

If you don't write enough data (underrun), the library will fill with *\x00*. If you write too much (overrun), the write method will block, until the data is consumed.

You should use audiostream.sources.ThreadSource instead.

write(*chunk : bytes*)

Parameters **chunk** (*bytes*) – Data chunk to write

Write a data chunk into the ring buffer. It will be consumed later by the speaker.

play()

Play the sample using the internal ring buffer

stop()

Stop the playback

2.2 Sample generators

class audiostream.sources.thread.**ThreadSource**(*AudioSample*)
Sample generator using thread, does nothing by default. It can be used to implement your own generator.

__init__(*stream : AudioOutput*)

Parameters **stream** (AudioOutput) – The AudioOutput instance to use

get_bytes() → bytes
Must return a bytes string with the data to store in the ring buffer.

class audiostream.sources.wave.**SineSource**(*ThreadSource*)
Sample generator that use the ThreadSource, and generate bytes from a sin() generator.

__init__(*stream : AudioOutput*, *frequency : int*)

Parameters

- **stream** (AudioOutput) – The AudioOutput instance to use
- **frequency** (*integer*) – The sin() frequency, for example: 440.

class audiostream.sources.puredata.**PatchSource**(*ThreadSource*)
Load a *PureData* <*http://puredata.info*> patch, and read the generated output.

__init__(*stream : AudioOutput*, *patchfile : string*)

Parameters

- **stream** (AudioOutput) – The AudioOutput instance to use
- **patchfile** (*string*) – The patch filename to load with pylibpd

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

a

Symbols

A

B

C

E

G

P

R

S

T

W