
pastream Documentation

Release 0.2.0.post0

Thomas J. Garcia

Jun 09, 2018

Contents

1	Features	3
2	External Dependencies	5
3	Installation	7
4	Building From Source	9
5	Building Documentation	11
6	Examples	13
7	Command Line Application	15
8	Release Notes	17
9	API Reference	21
	Python Module Index	31

pastream builds on top of *portaudio* and the excellent *sounddevice* python bindings to provide some more advanced functionality right out of the box. Note that in addition to the *pastream library*, *pastream* includes a *command line application* for playing and recording audio files.

Documentation: <http://pastream.readthedocs.io/>

Source code repository and issue tracker: <http://github.com/tgarc/pastream/>

GIL-less Audio Callbacks Having the portaudio callback implemented in C means audio interrupts can be serviced quickly and reliably without ever needing to acquire the Python Global Interpreter Lock (GIL). This is crucial when working with libraries like [Pillow](#) which may greedily grab and hold the GIL subsequently causing audio overruns/underruns.

Input Stream iterators Efficiently retrieve live audio capture data through an iterable. As simple as:

```
import pastream as ps
for chunk in ps.chunks():
    process(chunk)
```

See `pastream.chunks` and `pastream.InputStream.chunks` method.

Built-in support for working with SoundFiles and numpy ndarrays Seamless support for playback/recording of numpy ndarrays, generic buffer types, and SoundFiles.

Reader/Writer Threads `pastream` simplifies the process of implementing stream reader and writer threads to manipulate and/or generate data in the background while leaving the main thread free for higher level management tasks.

External Dependencies

There are a few compiled libraries pastream requires which *may* need to be installed separately depending on your operating system. Windows users are luckiest, they can skip this section entirely.

libffi (Linux/Unix/MacOSX): Under Linux/Unix/MacOSX platforms you'll need to install the ffi library. (For Windows users, ffi is already included with the python cffi package.) libffi is available through most package managers:

```
$ yum install libffi-devel # Red-hat/CentOS Linux
$ apt-get install libffi-dev # Ubuntu/debian derivatives
$ brew install libffi # Homebrew on OSX
```

More information on installing libffi is available in the cffi documentation [here](#).

PortAudio and libsndfile (Linux/Unix): Linux and Unix users will also need to install a recent version of the PortAudio and libsndfile libraries. (For Windows and OSX, the sounddevice and soundfile python packages include prebuilt versions for you.) You can either install the latest available from your package manager (e.g. `apt-get install libportaudio2 libsndfile` for debian/raspbian) or install the latest stable build from the package website (Recommended).

CHAPTER 3

Installation

Once the above dependencies have been resolved, you can install pastream using pip:

```
$ pip install pastream
```

Building From Source

Clone pastream with the `--recursive` flag:

```
$ git clone --recursive http://github.com/tgarc/pastream
```

Or, if you already have a checkout:

```
$ cd <path/to/checkout>  
$ git submodule update --init
```

Finally, do a pip install from your local working copy:

```
$ pip install <path/to/checkout>
```

Building Documentation

Documentation for pastream can be easily generated in a wide variety of formats using Sphinx. Just follow the steps below.

Checkout the repository and cd into it:

```
$ git clone http://github.com/tgarc/pastream
$ cd pastream
```

Install documentation dependencies using requirements file:

```
$ pip install -r docs/requirements.txt
```

Then use the included Makefile/make.bat to generate documentation. (Here we output to the html format):

```
$ cd docs
$ make html
```


CHAPTER 6

Examples

Record one second of audio to memory, then play it back:

```
import pastream as ps

# Use *with* statements to auto-close the stream
with ps.DuplexStream() as stream:
    out = stream.record(int(stream.samplerate), blocking=True)
    stream.play(out, blocking=True)
```

Playback 10 seconds of a file, adding zero padding if the file is shorter, and record the result to memory:

```
import pastream as ps, soundfile as sf

with sf.SoundFile('my-file.wav') as infile, ps.DuplexStream.from_file(infile) as \
↳stream:
    out = stream.playrec(infile, frames=10 * int(stream.samplerate), pad=-1, \
↳blocking=True)
```

Grab (real) frequency transformed live audio stream with 50% overlap:

```
import pastream as ps, numpy as np

chunksize = 1024
window = np.hanning(chunksize)
for x_l in ps.chunks(chunksize, overlap=chunksize//2, channels=1):
    X_l = np.fft.rfft(x_l * window)
```

Generate a pure tone on-the-fly

```
import time
import pastream as ps
import numpy as np

# A simple tone generator
```

(continues on next page)

(continued from previous page)

```
def tone_generator(stream, buffer, f, loop=False):
    fs = stream.samplerate

    # Create a time index
    t = 2*np.pi*f*np.arange(len(buffer), dtype=stream.dtype) / fs

    # Loop until the stream stops
    while not stream.finished:
        frames = buffer.write_available
        if not frames:
            time.sleep(0.010)
            continue

        # Get the write buffers directly to avoid making any extra copies
        frames, part1, part2 = buffer.get_write_buffers(frames)

        out = np.frombuffer(part1, dtype=stream.dtype)
        np.sin(t[:len(out)], out=out)

        if len(part2):
            # part2 will be nonempty whenever we wrap around the end of the ring_
            ↪buffer
            out = np.frombuffer(part2, dtype=stream.dtype)
            np.sin(t[:len(out)], out=out)

        # flag that we've added data to the buffer
        buffer.advance_write_index(frames)

        # advance the time index
        t += 2*np.pi*f*frames / fs

with ps.OutputStream(channels=1) as stream:
    # Set our tone generator as the source and pass along the frequency
    freq = 1000
    stream.set_source(tone_generator, args=(freq,))

    # Busy-wait to allow for keyboard interrupt
    stream.start()
    while stream.active:
        time.sleep(0.1)
```

See also the included examples under /examples.

Command Line Application

Once installed, the `pastream` application should be callable from your command line. If you're familiar with `SoX` you'll notice that some of the command line syntax is quite similar. Here are a few examples to help get you started.

Display the help file:

```
$ pastream -h
```

List available audio devices:

```
$ pastream -l
```

Simultaneous play and record from the default audio device:

```
$ pastream input.wav output.wav
```

Pipe input from `sox` using the AU format and record the playback:

```
$ sox -n -t au - synth sine 440 | pastream - output.wav
```

Play a RAW file:

```
$ pastream -c1 -r48k -e=pcm_16 output.raw
```

Record 10 minutes of audio at 48kHz:

```
$ pastream null output.wav -r48k -d10:00
```


0.2.0:

- (9223b) Fixed bug with loop mode which caused frame droppage
- (5cd35) Added `-channel-select` for ASIO backend
- (110b6-b56ee) Add channel mapping setttable through `tx/rxmapping` properties. Also adds `-map` option to cli

0.1.2:

- (ba45a) improve file read/write efficiency
- (b7b7d) added `-fatal-xruns` option to automatically abort on detected xruns
- (5d7dc) fixed encoding errors with non-ascii device names

0.1.1:

- minor fixes to docs and cli
- (c533a) add 'mode' argument to `to_file`
- (c533a) defer numpy import to improve total import time (as well as the launch time of the pastream cli app)
- (df4bc3) add tone generator example to README
- (df4bc3) bump sounddevice req up to 0.3.9 to get improved import times

0.1.0:

- (c70bda) Dropped `SoundFileStream` classes. This functionality is now integrated into the regular stream classes.
- (c70bda) Added `play/record/playrec` methods
- (dbe076) Added seamless support for `SoundFiles`
- (dbe076) Added built-in support for looping both files and buffers
- (a65cc5) Added `set_source/set_sink` methods as alternatives to `play/record` and as a mechanism to set reader/writer threads.

- (a65cc5) Added to_file/from_file convenience methods
- (7676eb+58472e) offset/pad/duration are all now specified in hours:minutes:seconds by default. samples can still be specified by appending an 's' suffix (just like with SoX).

0.0.8:

- BUG: fixed possible bad behavior when pad >= 0 frames < 0 (06881)
- BUG: pad > 0 can cause too many frame reads (fixed in e917e)
- Receive buffer is no longer automatically flushed when calling start() (cd65b)
- BUG: AttributeError was not correctly being caught and reraised in stream threads (3bc5e)
- Added sphinx documentation (11c13)
- frames attribute changed from long to long long (ee4ebb)
- chunks: eliminated an unnecessary copy when using overlap (b0304)

0.0.7:

- add `-loop` option to the CLI to allow looping playback.
- allow empty string as an alternative to null
- Raise exception when attempting to open stream with RAW playback file if any of samplerate/channels/subtype are not specified.
- change prebuffering behavior slightly: only wait until the first write, not until the buffer fills up. This should avoid potential long pre-buffer times
- fix formatting errors in `__repr__` when using multiple dtypes and/or devices
- no need to vendor `pa_ringbuffer` anymore, it's available on pip! (Thanks @mgeier !)
- if a `SoundFile inpf` is passed to a `SoundFileInputStream` class, it will be used to set the stream samplerate/channels.
- addresses a bug when `BUFFERSIZE < 8192`
- `Stream` and `SoundFileStream` classes renamed to `*DuplexStream`
- Swapped assignments of input/output in `SoundFileStreams` to make it align with the usage in the rest of the library. The order of input/output arguments from the CLI still stays the same though.
- remove `allow_drops` parameters. It can be added back at a later point if it proves to be a more useful feature

0.0.6:

- fix 'null' not properly matching on cmdline
- chunks: check that portaudio has not been terminated before trying to close/stop a stream
- drop `allow_xruns/XRunError`
- `Buffered*Stream -> *Stream`
- `*Buffer{Empty,Full} -> Buffer{Empty,Full}`
- fix remaining issues with wheel building
- Dropped unused exception classes (`PaStreamError`, `AudioBufferError`)
- Added prebuffer argument to `start()` to bypass filling output buffer before stream starts

0.0.5:

- Redirect `sys.stdout` to `devnull` when '-' is used as the output file stream

- Specifying multiple `--file-types` at command line fixed
- `--format` now only accepts a single argument
- `ringbuffersize_t` is of a different type for mac platforms; fixed
- `ps.chunks()` README example fixed
- **frames is now a signed value. The behavior previously reserved for `frames == 0` now is active whenever `frames < 0`**
 - Comma separated arguments are no longer allowed; multiple argument options can only be specified by passing them multiple times
 - dropped support for passing a bool for `pad` parameter
 - `-q` flag for specifying buffersize has been dropped. This is now reserved for the new `--quiet` option.
- add a loopback test for the pastream app using `stdin > stdout`
- improvement: `chunks` function: make sure that stream is closed properly without the performance hit of having an extra yield
- new feature: If both `padding` and `frames` are `< 0`, padding will be added indefinitely
- new feature: `-q/--quiet` option; this drops the deprecated `-q` option for specifying buffersize

0.0.4:

- bugfix: `chunks`: overlap was (accidentally) not allowed if `chunksize` was not non-zero. This should be allowed as long as `stream.blocksize > 0`.
- `chunks` now supports passing a generic `ndarray` to `out` parameter (without having to cast it to a bytes object)
- `nframes` renamed to `frames`
- `padding` renamed to `pad`
- added `allow_drops` option to give user the option to ignore `ReceiveBufferEmpty` error in more atypical use cases
- `raise_on_xruns` changed to `allow_xruns`; inverted behavior
- got rid of undocumented `keep_alive` option; the combination of `allow_drops` and `pad` can give the same functionality
- `--pad` now can be specified without an argument which just sets `pad` to `True`
- added autopadding feature: Now if `frames > 0` and `pad == True` or `pad < 0`, playback will be zero padded out to `frames`. This is a nice feature for the pastream application and `SoundFileStream` since sometimes you want to add extra padding after the file playback.

0.0.3:

- command line options for size parameters now accept `k/K/m/M` suffix
- Backwards compatibility break: multiple argument command line options now accept a comma delimited list
- improved `SoundFileStream` reader writers; nearly zero read/write misses
- bugfix: `__repr__` had a bug for certain cases

0.0.2:

- Improved SoundFileStream interface: remove sfkwards; instead format, endian, and subtype can be passed directly since they don't collide with any of the sounddevice parameters
- Updated examples to allow half or full duplex operation. Also accepts subtype for RAW files
- chunks() updates * better polling behavior greatly decreases read misses * now supports generic buffers so numpy is not required * added `out` option to allow user to pass a preallocated buffer * bugfix: overlap was not overlapping correctly
- MAJOR bugfix: samplerate was not being properly passed up the class chain
- MAJOR bugfix: `lastTime` was not being properly copied in `py_pastream.c` so the value returned was garbage
- bugfix: `assert_chunks_equal`: the 'inframes' buffer was not being allocated enough space for when `chunk-size > blocksize` which was causing mismatch hysteria

0.0.1:

- First tenable release

pastream: GIL-less Portaudio Streams for Python

exception `pastream.BufferEmpty`

exception `pastream.BufferFull`

`pastream.chunks` (*chunksize=None, overlap=0, frames=-1, pad=0, offset=0, atleast_2d=False, playback=None, loop=False, buffersize=None, out=None, **kwargs*)

Read audio data in iterable chunks from a Portaudio stream.

This is simply a convenience function that provides the same functionality as `InputStream.chunks()`.

Parameters

- **overlap, frames, pad, offset, atleast_2d, playback, loop, (chunksize,)** –
- **out** (*buffersize,*) – See `InputStream.chunks()` for description.

Other Parameters ****kwargs** – Additional arguments to pass to Stream constructor.

Yields *ndarray or bytearray or type(out)* – buffer object with `chunksize` elements.

See also:

`InputStream.chunks()`

class `pastream.RingBuffer` (*elementsiz, size=None, buffer=None*)

PortAudio's single-reader single-writer lock-free ring buffer.

C API documentation: http://portaudio.com/docs/v19-doxydocs-dev/pa__ringbuffer_8h.html

Python wrapper: <https://github.com/spatialaudio/python-pa-ringbuffer>

Instances of this class can be used to transport data between Python code and some compiled code running on a different thread.

This only works when there is a single reader and a single writer (i.e. one thread or callback writes to the ring buffer, another thread or callback reads from it).

This ring buffer is *not* appropriate for passing data from one Python thread to another Python thread. For this, the `queue.Queue` class from the standard library can be used.

Parameters

- **elementsiz**e (*int*) – The size of a single data element in bytes.
- **size** (*int*) – The number of elements in the buffer (must be a power of 2). Can be omitted if a pre-allocated buffer is passed.
- **buffer** (*buffer*) – optional pre-allocated buffer to use with RingBuffer. Note that if you pass a read-only buffer object, you still get a writable RingBuffer; it is your responsibility not to write there if the original buffer doesn't expect you to.

advance_read_index (*size*)

Advance the read index to the next location to be read.

Parameters **size** (*int*) – The number of elements to advance.

Returns The new position.

Return type int

Note: This is only needed when using `get_read_buffers()`, the methods `read()` and `readinto()` take care of this by themselves!

advance_write_index (*size*)

Advance the write index to the next location to be written.

Parameters **size** (*int*) – The number of elements to advance.

Returns The new position.

Return type int

Note: This is only needed when using `get_write_buffers()`, the method `write()` takes care of this by itself!

elementsize

Element size in bytes.

flush ()

Reset buffer to empty.

Should only be called when buffer is **not** being read or written.

get_read_buffers (*size*)

Get buffer(s) from which we can read data.

When done reading, use `advance_read_index()` to make the memory available for writing again.

Parameters **size** (*int*) – The number of elements desired.

Returns

- The number of elements available for reading (which might be less than the requested *size*).
- The first buffer.
- The second buffer.

Return type (int, buffer, buffer)

get_write_buffers (*size*)

Get buffer(s) to which we can write data.

When done writing, use `advance_write_index()` to make the written data available for reading.

Parameters **size** (*int*) – The number of elements desired.

Returns

- The room available to be written or the given *size*, whichever is smaller.
- The first buffer.
- The second buffer.

Return type (int, buffer, buffer)

read (*size=-1*)

Read data from the ring buffer into a new buffer.

This advances the read index after reading; calling `advance_read_index()` is *not* necessary.

Parameters **size** (*int, optional*) – The number of elements to be read. If not specified, all available elements are read.

Returns A new buffer containing the read data. Its size may be less than the requested *size*.

Return type buffer

read_available

Number of elements available in the ring buffer for reading.

readinto (*data*)

Read data from the ring buffer into a user-provided buffer.

This advances the read index after reading; calling `advance_read_index()` is *not* necessary.

Parameters **data** (*CData pointer or buffer*) – The memory where the data should be stored.

Returns The number of elements read, which may be less than the size of *data*.

Return type int

write (*data, size=-1*)

Write data to the ring buffer.

This advances the write index after writing; calling `advance_write_index()` is *not* necessary.

Parameters

- **data** (*CData pointer or buffer or bytes*) – Data to write to the buffer.
- **size** (*int, optional*) – The number of elements to be written.

Returns The number of elements written.

Return type int

write_available

Number of elements available in the ring buffer for writing.

class `pastream.Stream` (*kind, device=None, samplerate=None, channels=None, dtype=None, block-size=None, **kwargs*)

Base stream class from which all other stream classes derive.

Note that this class inherits from `sounddevice's _StreamBase` class.

abort ()

aborted

Check whether stream has been aborted.

If True, it is guaranteed that the stream is in a finished state.

close ()

finished

Check whether the stream is in a finished state.

Will only be True if *start ()* has been called and the stream either completed successfully or was stopped/aborted.

frame_count

Running total of frames that have been processed.

Each new starting of the stream resets this number to zero.

classmethod from_file (*file*, **args*, ***kwargs*)

Create a stream using the characteristics of a soundfile

Parameters *file* (*SoundFile* or *str* or *int* or *file-like object*) –

Other Parameters **args*, ***kwargs* – Arguments to pass to Stream constructor

Returns Open stream

Return type *Stream* or Stream subclass instance

See also:

InputStream.to_file ()

isduplex

Return whether this is a full duplex stream or not

start (*prebuffer=True*)

Start the audio stream

Parameters *prebuffer* (*bool* or *int*, *optional*) – Wait for a number of frames to be written to the output buffer before starting the audio stream. If True is given just wait for the first write. If not using threads or the stream is not an output stream this has no effect.

status

The current PaStreamCallbackFlags status of the portaudio stream.

stop ()

wait (*timeout=None*)

Block until stream state changes to finished/aborted/stopped or until the optional timeout occurs.

Parameters *time* (*float*, *optional*) – Optional timeout in seconds.

Returns True unless the timeout occurs.

Return type bool

xruns

Running total of xruns.

Each new starting of the stream resets this number to zero.

class `pastream.InputStream` (**args*, ***kwargs*)

Record only stream.

Other Parameters **args, **kwargs* – Arguments to pass to *Stream*.

chunks (*chunksize=None, overlap=0, frames=-1, pad=-1, offset=0, atleast_2d=False, playback=None, loop=False, buffersize=None, out=None*)
Read audio data in iterable chunks from a Portaudio stream.

Similar in concept to PySoundFile library's `blocks()` method. Returns an iterator over buffered audio chunks read from a Portaudio stream. By default a direct view into the stream's ringbuffer is returned whenever possible. Setting an `out` buffer will of course incur an extra copy.

Parameters

- **chunksize** (*int, optional*) – Size of iterator chunks. If not specified the stream blocksize will be used. Note that if the blocksize is zero the yielded audio chunks may be of variable length depending on the audio backend.
- **overlap** (*int, optional*) – Number of frames to overlap across blocks.
- **frames** (*int, optional*) – Number of frames to play/record.
- **pad** (*int, optional*) – Playback padding. See *OutputStream.play()*. Only applicable when playback is given.
- **offset** (*int, optional*) – Recording offset. See *InputStream.record()*.
- **atleast_2d** (*bool, optional*) – (numpy only) Always return chunks as 2 dimensional arrays.
- **playback** (*buffer or SoundFile, optional*) – Set playback audio. Only works for full duplex streams.
- **loop** (*bool, optional*) – Loop the playback audio.
- **out** (*ndarray or buffer object, optional*) – Alternative output buffer in which to store the result. Note that any buffer object - with the exception of *ndarray* - is expected to have single-byte elements as would be provided by e.g., *bytearray*.

Yields *ndarray* or *memoryview* or *ffi.buffer* – Buffer object with `chunksize` frames. If numpy is available defaults to *ndarray* otherwise a buffer of bytes is yielded (which is either a *ffi.buffer* object or a *memoryview*).

See also:

chunks()

record (*frames=None, offset=0, atleast_2d=False, buffersize=None, blocking=False, out=None*)
Record audio data to a buffer or file

Parameters

- **frames** (*int, sometimes optional*) – Number of frames to record. Can be omitted if `out` is specified.
- **offset** (*int, optional*) – Number of frames to discard from beginning of recording.
- **atleast_2d** (*bool, optional*) – (numpy only) Always return output as a 2 dimensional array.
- **buffersize** (*int, optional*) – Buffer size to use for (double) buffering audio data to file. Only applicable when `out` is a file. Must be a power of 2.
- **blocking** (*bool, optional*) – Wait for recording to finish before returning.
- **out** (*buffer or SoundFile, optional*) – Output sink.

Returns Recording destination.

Return type ndarray or bytearray or type(out)

See also:

InputStream.set_sink(), *DuplexStream.playrec()*

rxmapping

Mutable mapping of input data channel to output channels

A mutable property specifying what input data channels are mapped to each output channel. Note that input channels may only be mapped once (i.e., channel mixing is not supported). Can also be specified as a dict of input:output channel key value pairs.

See also:

OutputStream.txmapping

set_sink (*sink*, *buffer_size=None*, *args=()*, *kwargs={}*)

Set the recording sink for the audio stream

Parameters

- **sink** (*function or RingBuffer or SoundFile or buffer type*) – Recording sink. If *sink* is a function it must be of the form: `function(stream, ringbuffer, *args, **kwargs)`. Function sources are useful if you want to handle capturing of audio data in some custom way. For example, *sink* could be a function that writes audio data directly to a socket. This function will be called from a separate thread whenever the stream is started and is expected to close itself whenever the stream becomes inactive. For an example see the `_soundfilerecorder` function in the source code for this module.
- **buffer_size** (*int, optional*) – RingBuffer size to use for (double) buffering audio data. Only applicable when *sink* is either a file or function. Must be a power of 2.
- **kwargs** (*args,*) – Additional arguments to pass if *sink* is a function.

Returns RingBuffer wrapper interface to which audio device will write audio data.

Return type RingBuffer instance

See also:

OutputStream.set_source()

to_file (*file*, *mode='w'*, ***kwargs*)

Open a SoundFile for writing based on stream properties

Parameters

- **file** (*str or int or file-like object*) – File to open as SoundFile
- ****kwargs** – Additional arguments to pass to SoundFile constructor

Raises `TypeError` – If no subtype was given and an appropriate subtype could not be guessed.

Returns

Return type SoundFile

See also:

Stream.from_file()

class `pastream.OutputStream` (**args*, ***kwargs*)

Playback only stream.

Other Parameters **args, **kwargs* – Arguments to pass to *Stream*.

classmethod `from_file` (*file, *args, **kwargs*)

Create a stream using the characteristics of a soundfile

Parameters *file* (*SoundFile* or *str* or *int* or *file-like object*)–

Other Parameters **args, **kwargs* – Arguments to pass to Stream constructor

Returns Open stream

Return type *Stream* or Stream subclass instance

See also:

InputStream.to_file()

play (*playback, frames=-1, pad=0, loop=False, buffersize=None, blocking=False*)

Play back audio data from a buffer or file

Parameters

- **playback** (*buffer* or *SoundFile*) – Playback source.
- **frames** (*int, optional*) – Number of frames to play. (Note: This does *not* include the length of any additional padding). A negative value (the default) will cause the stream to continue until the send buffer is empty.
- **pad** (*int, optional*) – Number of zero frames to pad the playback with. A negative value causes padding to be automatically chosen so that the total playback length matches *frames* (or, if *frames* is negative, zero padding will be added indefinitely).
- **blocking** (*bool, optional*) – Wait for playback to finish before returning.

Other Parameters *loop, buffersize* – See *OutputStream.set_source()*

See also:

OutputStream.set_source(), DuplexStream.playrec()

set_source (*source, loop=False, buffersize=None, args=(), kwargs={}*)

Set the playback source for the audio stream

Parameters

- **source** (*function* or *RingBuffer* or *SoundFile* or *buffer type*) – Playback source. If *source* is a function it must be of the form: `function(stream, ringbuffer, *args, loop=<bool>, **kwargs)`. Function sources are useful if you want to handle generating playback in some custom way. For example, *source* could be a function that reads audio data from a socket. This function will be called from a separate thread whenever the stream is started and is expected to close itself whenever the stream becomes inactive. For an example see the `_soundfileplayer` function in the source code for this module.
- **loop** (*bool, optional*) – Whether to enable playback looping.
- **buffersize** (*int, optional*) – RingBuffer size to use for double buffering audio data. Only applicable if *source* is a function or *SoundFile*. Must be a power of 2.

Other Parameters *args, kwargs* – Additional arguments to pass if *source* is a function.

Returns RingBuffer wrapper interface from which audio device will read audio data.

Return type RingBuffer instance

See also:

InputStream.set_sink()

txmapping

Mutable mapping of input data channel to output channels

A mutable property specifying what input data channels are mapped to each output channel. Note that input channels may only be mapped once (i.e., channel duplication is not supported). Can also be specified as a dict of input:output channel key value pairs.

See also:

InputStream.rxmapping

class `pastream.DuplexStream(*args, **kwargs)`

Full duplex audio streamer.

Other Parameters **args, **kwargs* – Arguments to pass to *Stream*.

See also:

OutputStream, InputStream

classmethod `from_file(playback, *args, **kwargs)`

Open a stream using the properties of a playback Soundfile

Parameters `playback` (*SoundFile or str or int or file-like object*) – Playback audio file from which to create stream.

Other Parameters **args, **kwargs* – Arguments to pass to Stream constructor

Returns Open stream

Return type *Stream*

See also:

InputStream.to_file()

playrec (*playback, frames=None, pad=0, offset=0, atleast_2d=False, loop=False, buffersize=None, blocking=False, out=None*)

Simultaneously record and play audio data

Parameters

- **playback** (*buffer or SoundFile*) – Playback source.
- **frames** (*int, sometimes optional*) – Number of frames to play/record. This is required whenever playback is a file and out is not given.
- **buffersize** (*int*) – Buffer size to use for (double) buffering audio data to/from file. Only applicable when one or both of {playback, out} is a file. Must be a power of 2.
- **offset, atleast_2d, loop, blocking, out** (*pad,*) – See description of *InputStream.record()* and *OutputStream.play()*.

Returns Recording destination.

Return type ndarray or bytearray or type(out)

See also:

OutputStream.play(), InputStream.record()

rxmapping

Mutable mapping of input data channel to output channels

A mutable property specifying what input data channels are mapped to each output channel. Note that input channels may only be mapped once (i.e., channel mixing is not supported). Can also be specified as a dict of input:output channel key value pairs.

See also:

OutputStream.txmapping

to_file (*file*, *mode='w'*, ***kwargs*)

Open a SoundFile for writing based on stream properties

Parameters

- **file** (*str or int or file-like object*) – File to open as SoundFile
- ****kwargs** – Additional arguments to pass to SoundFile constructor

Raises `TypeError` – If no subtype was given and an appropriate subtype could not be guessed.

Returns

Return type `SoundFile`

See also:

Stream.from_file()

txmapping

Mutable mapping of input data channel to output channels

A mutable property specifying what input data channels are mapped to each output channel. Note that input channels may only be mapped once (i.e., channel duplication is not supported). Can also be specified as a dict of input:output channel key value pairs.

See also:

InputStream.rxmapping

p

`pastream`, 21

A

abort() (pastream.Stream method), 23
aborted (pastream.Stream attribute), 24
advance_read_index() (pastream.RingBuffer method), 22
advance_write_index() (pastream.RingBuffer method), 22

B

BufferEmpty, 21
BufferFull, 21

C

chunks() (in module pastream), 21
chunks() (pastream.InputStream method), 25
close() (pastream.Stream method), 24

D

DuplexStream (class in pastream), 28

E

elementsize (pastream.RingBuffer attribute), 22

F

finished (pastream.Stream attribute), 24
flush() (pastream.RingBuffer method), 22
frame_count (pastream.Stream attribute), 24
from_file() (pastream.DuplexStream class method), 28
from_file() (pastream.OutputStream class method), 27
from_file() (pastream.Stream class method), 24

G

get_read_buffers() (pastream.RingBuffer method), 22
get_write_buffers() (pastream.RingBuffer method), 23

I

InputStream (class in pastream), 24
isduplex (pastream.Stream attribute), 24

O

OutputStream (class in pastream), 26

P

pastream (module), 21
play() (pastream.OutputStream method), 27
playrec() (pastream.DuplexStream method), 28

R

read() (pastream.RingBuffer method), 23
read_available (pastream.RingBuffer attribute), 23
readinto() (pastream.RingBuffer method), 23
record() (pastream.InputStream method), 25
RingBuffer (class in pastream), 21
rxmapping (pastream.DuplexStream attribute), 28
rxmapping (pastream.InputStream attribute), 26

S

set_sink() (pastream.InputStream method), 26
set_source() (pastream.OutputStream method), 27
start() (pastream.Stream method), 24
status (pastream.Stream attribute), 24
stop() (pastream.Stream method), 24
Stream (class in pastream), 23

T

to_file() (pastream.DuplexStream method), 29
to_file() (pastream.InputStream method), 26
txmapping (pastream.DuplexStream attribute), 29
txmapping (pastream.OutputStream attribute), 28

W

wait() (pastream.Stream method), 24
write() (pastream.RingBuffer method), 23
write_available (pastream.RingBuffer attribute), 23

X

xruns (pastream.Stream attribute), 24