

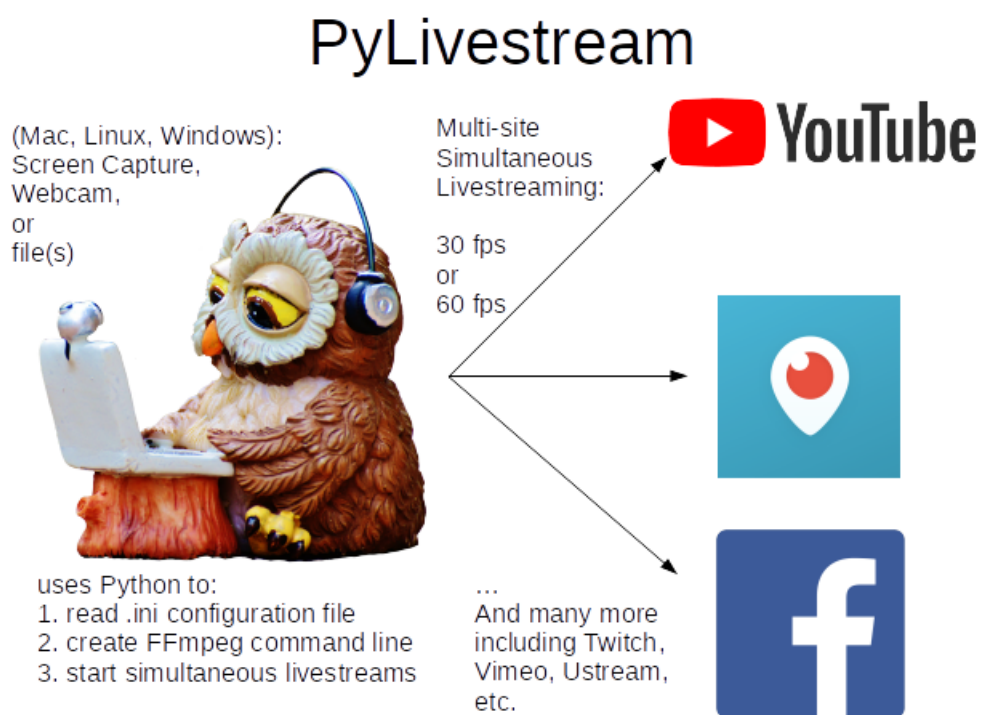
---

## Python scripted livestreaming using FFmpeg

DOI [10.5281/zenodo.11123952](https://doi.org/10.5281/zenodo.11123952)  ci [passing](#)  python 3  downloads 164k

Streams to one or **multiple** streaming sites simultaneously, using pure object-oriented Python (no extra packages) and FFmpeg. Tested with [flake8](#), [mypy](#) type checking and [pytest](#). [visual\\_tests.py](#) is a quick check of several command line scripting scenarios on your laptop. FFmpeg is used from Python [subprocess](#) to stream to sites including:

- Facebook Live (requires FFmpeg  $\geq 4.2$  due to mandatory RTMPS)
- YouTube Live
- Twitch
- also Ustream, Vimeo, Restream.io and more for streaming broadcasts.



Troubleshooting

---

## PyLivestream benefits

- Python scripts compute good streaming parameters, and emit the command used to copy and paste if desired.
- Works on any OS (Mac, Linux, Windows) and computing platform, including PC, Mac, and Raspberry Pi.
- Uses single JSON file pylivestream.json to adjust parameters.

## PyLivestream limitations

- does *not* auto-restart if network connection glitches
- is intended as a bare minimum command generator to run the FFmpeg program
- is not intended for bidirectional robust streaming—consider a program/system based on Jitsi for that.

## Design rationale

Why not do things without the command line, via linking libffmpeg, libstreamer or libav?

- the command-line approach does not require a compiler or OS-dependent libraries
- once you get a setup working once, you don't even need Python anymore—just copy and paste the command line

## Alternatives

Other projects using FFmpeg from Python include:

- python-ffmpeg lower level use of FFmpeg with Python asyncio
- asyncio-subprocess-ffmpeg simple asyncio subprocess example that could also be used as a template for general asyncio subprocess Python use.
- ffmpeg FFmpeg subprocess without asyncio

## Install

Requires FFmpeg  $\geq 3.0$  ( $\geq 4.2$  for Facebook Live RTMPS)

Latest release:

```
1 python3 -m pip install PyLivestream
```

---

Development version:

```
1 git clone https://github.com/scivision/PyLivestream
2
3 cd PyLivestream
4
5 python3 -m pip install -e .
```

FFmpeg can be obtained like:

- Windows: `winget install ffmpeg`
- Linux: `sudo apt install ffmpeg`
- MacOS: `brew install ffmpeg`

If errors result from FFmpeg not in PATH environment variable, optionally set environment variable “FFMPEG\_ROOT” to the directory containing FFmpeg executable.

### Configuration: pylivestream.json

You can skip past this section to “stream start” if it’s confusing. The defaults might work to get you started.

The pylivestream.json file you create has parameters relevant to the live stream. We suggest copying the example pylivestream.json and editing, then specify it for your streams.

- `screencap_origin`: origin (upper left corner) of screen capture region in pixels.
- `screencap_size`: resolution of screen capture (area to capture, starting from origin)
- `screencap_fps`: frames/sec of screen capture
- `video_kbps`: override automatic video bitrate in kbps
- `audio_rate`: audio sampling frequency. Typically 44100 Hz (CD quality).
- `audio_bps`: audio data rate—**leave blank if you want no audio** (usually used for “file”, to make an animated GIF in post-processing)
- `preset`: `veryfast` or `ultrafast` if CPU not able to keep up.
- `exe`: override path to desired FFmpeg executable. In case you have multiple FFmpeg versions installed (say, from Anaconda Python).

Next are `sys.platform` specific parameters.

Seek help in FFmpeg documentation, try capturing to a file first and then update `~/pylivestream.json` for `sys.platform`.

---

## Deduce inputs

Each computer will need distinct pylivestream.json device input parameters:

- audio\_chan: audio device
- camera\_chan: camera device
- screen\_chan: desktop capture software port name

Loopback devices that let you “record what you hear” are operating system dependent. You may need to search documentation for your operating system to enable such a virtual loopback device.

### Windows

```
1 ffmpeg -list_devices true -f dshow -i dummy
```

### MacOS

```
1 ffmpeg -f avfoundation -list_devices true -i ""
```

### Linux

```
1 v4l2-ctl --list-devices
```

## API

There are two ways to start a stream (assuming you’ve configured as per following sections). Both do the same thing.

- command line
  - python -m pylivestream.glob
  - python -m pylivestream.screen
  - python -m pylivestream.loopfile
  - python -m pylivestream.screen2disk
  - python -m pylivestream.camera
  - python -m pylivestream.microphone
- **import** pylivestream.api as pls from within your Python script. For more information type `help(pls)` or `help(pls.stream_microphone)`
  - pls.stream\_file()
  - pls.stream\_microphone()
  - pls.stream\_camera()

---

## Authentication

The program loads a JSON file with the stream URL and hexadecimal stream key for the website(s) used. The user must specify this JSON file location.

### YouTube Live

1. configure YouTube Live.
2. Edit “pylivestream.json” to have the YouTube streamid
3. Run Python script and chosen input will stream on YouTube Live.

```
1 python -m pylivestream.screen youtube ./pylivestream.json
```

### Facebook Live

Facebook Live requires FFmpeg >= 4.2 due to mandatory RTMPS

1. configure your Facebook Live stream
2. Put stream ID into the JSON file
3. Run Python script for Facebook with chosen input

```
1 python -m pylivestream.screen facebook ./pylivestream.json
```

### Twitter

TODO

### Twitch

Create stream from Twitch Dashboard. Edit pylivestream.json file with “url” and “streamid” for Twitch. Run Python script for Twitch with chosen input:

```
1 python -m pylivestream.screen twitch ./pylivestream.json
```

## Usage

Due to the complexity of streaming and the non-specific error codes FFmpeg emits, the default behavior is that if FFmpeg detects one stream has failed, ALL streams will stop streaming and the program ends.

---

Setup a pylivestream.json for computer and desired parameters. Copy the provided pylivestream.json and edit with values you determine.

File-Streaming

## Camera

Note: your system may not have a camera, particularly if it's a virtual machine.

JSON:

- `camera_size`: camera resolution – find from `v4l2-ctl --list-formats-ext` or camera spec sheet.
- `camera_fps`: camera fps – found from command above or camera spec sheet

Stream to multiple sites, in this example Facebook Live and YouTube Live simultaneously:

```
1 python -m pylivestream.camera youtube facebook ./pylivestream.json
```

## Screen Share Livestream

Stream to multiple sites, in this example Facebook Live and YouTube Live simultaneously:

```
1 python -m pylivestream.screen youtube facebook ./pylivestream.json
```

## Image + Audio Livestream

Microphone audio + static image is accomplished by

```
1 python -m pylivestream.microphone youtube facebook ./pylivestream.json
   -image doc/logo.jpg
```

or wherever your image file is.

## Audio-only Livestream

Audio-only streaming is not typically allowed by the Video streaming sites. It may fail to work altogether, or may fail when one file is done and another starts. That's not something we can fix within the scope of this project. You can test it to your own computer by:

```
1 python -m pylivestream.microphone localhost ./pylivestream.json
```

---

## Screen capture to disk

This script saves your screen capture to a file on your disk:

```
1 python -m pylivestream.screen2disk myvid.avi ./pylivestream.json
```

## Utilities

- `PyLivestream.get_framerate(vidfn)` gives the frames/sec of a video file.
- `PyLivestream.get_resolution(vidfn)` gives the resolution (width x height) of video file.

## Notes

- Linux requires X11, not Wayland (choose at login)
- `x11grab` was deprecated in FFmpeg 3.3, was previously replaced by `xcbgrab`
- Reference webpage

## FFmpeg References

- streaming
- camera
- Camera overlay

## Windows

- `gdigrab`

DirectShow didn't work for me on Windows 10, so I used `gdigrab` instead.

- DirectShow device selection
- DirectShow examples

## Stream References

- Twitch parameters
- Twitch ingest servers

- 
- Twitch encoding
  - Twitter Live parameters
  - YouTube Live parameters
  - Facebook Live parameters
  - Ustream parameters
  - Vimeo config
  - Vimeo parameters

### **Logo Credits**

- Owl PC: Creative Commons no attrib. commercial
- YouTube: YouTube Brand Resources
- Facebook: Wikimedia Commons