Livestreamer

Release 1.12.2

Nov 11, 2017

Contents

Livestreamer is a *command-line utility* that pipes video streams from various services into a video player, such as VLC. The main purpose of Livestreamer is to allow the user to avoid buggy and CPU heavy flash plugins but still be able to enjoy various streamed content. There is also an *API* available for developers who want access to the video stream data.

- Latest release: 1.12.2 (changelog)
- GitHub: https://github.com/chrippa/livestreamer
- Issue tracker: https://github.com/chrippa/livestreamer/issues
- PyPI: https://pypi.python.org/pypi/livestreamer
- Discussions: https://groups.google.com/forum/#!forum/livestreamer
- IRC: #livestreamer @ Freenode
- Free software: Simplified BSD license

CHAPTER 1

Features

Livestreamer is built upon a plugin system which allows support for new services to be easily added. Currently most of the big streaming services are supported, such as:

- Dailymotion
- Livestream
- Twitch
- UStream
- YouTube Live

... and many more. A full list of plugins currently included can be found on the *Plugins* page.

CHAPTER 2

Quickstart

The default behaviour of Livestreamer is to playback a stream in the default player (VLC).

```
# pip install livestreamer
$ livestreamer twitch.tv/day9tv best
[cli][info] Found matching plugin twitch for URL twitch.tv/day9tv
[cli][info] Opening stream: source (hls)
[cli][info] Starting player: vlc
```

For more in-depth usage and install instructions see the *User guide*.

CHAPTER 3

User guide

Livestreamer is made up of two parts, a *Command-Line Interface* and a library *API*. See their respective sections for more information on how to use them.

3.1 Installation

3.1.1 Linux and BSD packages

Distribution	Installing
Arch Linux (package)	`# pacman -S livestreamer`
Arch Linux (aur, git)	Installing AUR packages

CRUX	`$ cd /usr/ports/contrib/livestreamer`
	`# pkgmk -d -i`

Debian	`# apt-get install livestreamer`
Exherbo Linux	
Fedora	`# yum install livestreamer`
FreeBSD (package)	`# pkg install multimedia/livestreamer`

FreeBSD (ports)	`$ cd /usr/ports/multimedia/livestreamer`
	`# make install clean`

Gentoo Linux	`# emerge net-misc/livestreamer`

NetBSD (pkgsrc)	`$ cd /usr/pkgsrc/multimedia/livestreamer`
	`# make install clean`

OpenBSD (package)	`# pkg_add livestreamer`

OpenBSD (ports)	`$ cd /usr/ports/multimedia/livestreamer`
	`# make install clean`

Slackware Linux	Installing Slackbuilds
Ubuntu	`# apt-get install livestreamer`

3.1.2 Other platforms

Platform	Installing
Mac OS X	`# easy_install -U livestreamer`
Microsoft Windows	See *Windows binaries*.

3.1.3 Source code

If a package is not available for your platform (or it's out of date) you can install Livestreamer via source.

There are a few different methods to do this, pip the Python package manager, **`easy_install`** the older package manager included with python-setuptools or by checking out the latest code with Git.

The commands listed here will also upgrade any existing version of Livestreamer.

Version	Installing
Latest release (pip)	`# pip install -U livestreamer`
Latest release (easy_install)	`# easy_install -U livestreamer`

Development version (pip)	`# pip install -U git+https://github.com/`
	`↪chrippa/livestreamer.git`

Development version (git)	`$ git clone git://github.com/chrippa/`
	`↪livestreamer.git`
	`$ cd livestreamer`
	`# python setup.py install`

Dependencies

To install Livestreamer from source you will need these dependencies.

Name	Notes
Python	At least version **2.6** or **3.3**.
python-setuptools	
Automatically installed by the setup script	
python-argparse	Only needed on Python **2.6**.
python-futures	Only needed on Python **2.x**.
python-requests	At least version **1.0**.
python-singledispatch	Only needed on Python versions older than **3.4**.
Optional	
RTMPDump	Required to play RTMP streams.
PyCrypto	Required to play some encrypted streams.
python-librtmp	Required by the *ustreamtv* plugin to be able to use non-mobile streams.

Installing without root permissions

If you do not wish to install Livestreamer globally on your system it's recommended to use virtualenv to create a user owned Python environment instead.

```
Creating an environment
$ virtualenv ~/myenv

Activating the environment
$ source ~/myenv/bin/activate

Installing livestreamer into the environment
(myenv)$ pip install livestreamer

Using livestreamer in the enviroment
(myenv)$ livestreamer ...

Deactivating the enviroment
(myenv)$ deactivate

Using livestreamer without activating the environment
$ ~/myenv/bin/livestreamer ...
```

Note: This may also be required on some OS X versions that seems to have weird permission issues (see issue #401).

3.1.4 Windows binaries

Installer

This is a installer which contains:

- A compiled version of Livestreamer that does not require an existing Python installation
- RTMPDump for viewing RTMP streams

and performs the following tasks:

- Generates a default *configuration file*
- Adds Livestreamer to your `$PATH` (making it possible to use **`livestreamer`** directly from the command prompt without specifying its directory)

Zip archive

This is minimal zip archive containing a compiled version of Livestreamer that does not require an existing Python installation.

Nightly build

This is an automatically generated build of the latest development code from the git repo.

Note: The binaries requires Microsoft Visual C++ 2008 Redistributable Package to be installed.

3.2 Command-Line Interface

3.2.1 Tutorial

Livestreamer is command-line application, this means the commands described here should be typed into a terminal. On Windows this means you should open the command prompt or PowerShell, on Mac OS X open the Terminal app and if you're on Linux or BSD you probably already know the drill.

The way Livestreamer works is that it's only a means to extract and transport the streams, and the playback is done by an external video player. Livestreamer works best with VLC or mpv, which are also cross-platform, but other players may be compatible too, see the *Players* page for a complete overview.

Now to get into actually using Livestreamer, let's say you want to watch the stream located on http://twitch.tv/day9tv, you start off by telling Livestreamer where to attempt to extract streams from. This is done by giving the URL to the command **livestreamer** as the first argument:

```
$ livestreamer twitch.tv/day9tv
[cli][info] Found matching plugin twitch for URL twitch.tv/day9tv
Available streams: audio, high, low, medium, mobile (worst), source (best)
```

Note: You don't need to include the protocol when dealing with HTTP URLs, e.g. just `twitch.tv/day9tv` is enough and quicker to type.

This command will tell Livestreamer to attempt to extract streams from the URL specified, and if it's successful, print out a list of available streams to choose from.

To select a stream and start playback, we simply add the stream name as a second argument to the **livestreamer** command:

```
$ livestreamer twitch.tv/day9tv source
[cli][info] Found matching plugin twitch for URL twitch.tv/day9tv
[cli][info] Opening stream: source (hls)
[cli][info] Starting player: vlc
```

The stream you chose should now be playing in the player. It's a common use case to just want start the highest quality stream and not be bothered with what it's named. To do this just specify `best` as the stream name and Livestreamer will attempt to rank the streams and open the one of highest quality. You can also specify `worst` to get the lowest quality.

Now that you have a basic grasp of how Livestreamer works, you may want to look into customizing it to your own needs, such as:

- Creating a *configuration file* of options you want to use
- Setting up your player to *cache some data* before playing the stream to help avoiding buffering issues

3.2.2 Configuration file

Writing the command-line options every time is inconvenient, that's why Livestreamer is capable of reading options from a configuration file instead.

Livestreamer will look for config files in different locations depending on your platform:

Platform	Location
Unix-like (POSIX)	• $XDG_CONFIG_HOME/livestreamer/config • ~/.livestreamerrc
Windows	%APPDATA%\livestreamer\livestreamerrc

You can also specify the location yourself using the `--config` option.

Note:

- *$XDG_CONFIG_HOME* is `~/.config` if it has not been overridden
- *%APPDATA%* is usually `<your user directory>\Application Data`

Note: On Windows there is a default config created by the installer but on any other platform you must create the file yourself.

Syntax

The config file is a simple text file and should contain one *command-line option* (omitting the dashes) per line in the format:

```
option=value
```

or for a option without value:

```
option
```

Note: Any quotes used will be part of the value, so only use when the value needs them, e.g. specifiying a player with a path containing spaces.

Example

```
# Player options
player=mpv --cache 2048
player-no-close

# Authenticate with Twitch
twitch-oauth-token=mytoken
```

3.2.3 Plugin specific configuration file

You may want to use specific options for some plugins only. This can be accomplished by placing those settings inside a plugin specific config file. Options inside these config files will override the main config file when a URL matching the plugin is used.

Livestreamer expects this config to be named like the main config but with `.<plugin name>` attached to the end.

Examples

Platform	Location
Unix-like (POSIX)	• $XDG_CONFIG_HOME/livestreamer/config.**twitch** • ~/.livestreamerrc.**ustreamtv**
Windows	%APPDATA%\livestreamer\livestreamerrc.**youtube**

Have a look at the *list of plugins* to see the name of each built-in plugin.

3.2.4 Plugin specific usage

Authenticating with Twitch

It's possible to access subscription content on Twitch by giving Livestreamer access to your account.

Authentication is done by creating an OAuth token that Livestreamer will use to access your account. It's done like this:

```
$ livestreamer --twitch-oauth-authenticate
```

This will open a web browser where Twitch will ask you if you want to give Livestreamer permission to access your account, then forwards you to a page with further instructions on how to use it.

Authenticating with Crunchyroll

Crunchyroll requires authenticating with a premium account to access some of their content. To do so, the plugin provides a couple of options to input your information, *--crunchyroll-username* and *--crunchyroll-password*.

You can login like this:

```
$ livestreamer --crunchyroll-username=xxxx --crunchyroll-password=xxx http://
↪crunchyroll.com/a-crunchyroll-episode-link
```

Note: If you omit the password, livestreamer will ask for it.

Once logged in, the plugin makes sure to save the session credentials to avoid asking your username and password again.

Neverthless, these credentials are valid for a limited amount of time, so it might be a good idea to save your username and password in your *configuration file* anyway.

Warning: The API this plugin uses isn't supposed to be available to use it on computers. The plugin tries to blend in as a valid device using custom headers and following the API usual flow (e.g. reusing credentials), but this does not assure that your account will be safe from being spotted for unusual behavior.

HTTP proxy with Crunchyroll

You can use the *--http-proxy* **and** *--https-proxy* options (you need both since the plugin uses both protocols) to access the Crunchyroll servers through a proxy to be able to stream region locked content.

When doing this, it's very probable that you will get denied to access the stream; this occurs because the session and credentials used by the plugin where obtained when logged from your own region, and the server still assumes you're in that region.

For this, the plugin provides the `--crunchyroll-purge-credentials` option, which removes your saved session and credentials and tries to log in again using your username and password.

3.2.5 Sideloading plugins

Livestreamer will attempt to load standalone plugins from these directories:

Platform	Location
Unix-like (POSIX)	$XDG_CONFIG_HOME/livestreamer/plugins
Windows	%APPDATA%\livestreamer\plugins

Note: If a plugin is added with the same name as a built-in plugin then the added plugin will take precedence. This is useful if you want to upgrade plugins independently of the Livestreamer version.

3.2.6 Playing built-in streaming protocols directly

There are many types of streaming protocols used by services today and Livestreamer supports most of them. It's possible to tell Livestreamer to access a streaming protocol directly instead of relying on a plugin to extract the streams from a URL for you.

A protocol can be accessed directly by specifying it in the URL format:

```
protocol://path [key=value]
```

Accessing a stream that requires extra parameters to be passed along (e.g. RTMP):

```
$ livestreamer "rtmp://streaming.server.net/playpath live=1 swfVfy=http://server.net/
↪flashplayer.swf"
```

Most streaming technologies simply requires you to pass a HTTP URL, this is a Adobe HDS stream:

```
$ livestreamer hds://streaming.server.net/playpath/manifest.f4m
```

Supported streaming protocols

Name	Prefix
Adobe HTTP Dynamic Streaming	hds://
Akamai HD Adaptive Streaming	akamaihd://
Apple HTTP Live Streaming	hls:// hlsvariant://
Real Time Messaging Protocol	rtmp:// rtmpe:// rtmps:// rtmpt:// rtmpte://
Progressive HTTP, HTTPS, etc	httpstream://

3.2.7 Command-line usage

```
$ livestreamer [OPTIONS] [URL] [STREAM]
```

Positional arguments

`[URL]`
A URL to attempt to extract streams from.

If it's a HTTP URL then "http://" can be omitted.

`[STREAM]`
Stream to play.

Use "best" or "worst" for highest or lowest quality available.

Fallback streams can be specified by using a comma-separated list:

```
"720p,480p,best"
```

If no stream is specified and *`--default-stream`* is not used then a list of available streams will be printed.

General options

`-h, --help`
Show this help message and exit.

`-V, --version`
Show version number and exit.

`--plugins`
Print a list of all currently installed plugins.

`--can-handle-url` `URL`
Check if Livestreamer has a plugin that can handle the specified URL.

Returns status code 1 for false and 0 for true.

Useful for external scripting.

`--config` `FILENAME`
Load options from this config file.

Can be repeated to load multiple files, in which case the options are merged on top of each other where the last config has highest priority.

`-l` `LEVEL`**`, --loglevel`** `LEVEL`
Set the log message threshold.

Valid levels are: none, error, warning, info, debug

`-Q, --quiet`
Hide all log output.

Alias for "*`--loglevel`* none".

`-j, --json`
Output JSON representations instead of the normal text output.

Useful for external scripting.

`--no-version-check`
Do not check for new Livestreamer releases.

`--version-check`
Runs a version check and exits.

Player options

-p COMMAND, **--player** COMMAND

Player to feed stream data to. This is a shell-like syntax to support passing options to the player. For example:

```
"vlc --file-caching=5000"
```

To use a player that is located in a path with spaces you must quote the path:

```
"'/path/with spaces/vlc' --file-caching=5000"
```

By default VLC will be used if it can be found in its default location.

-a ARGUMENTS, **--player-args** ARGUMENTS

This option allows you to customize the default arguments which are put together with the value of `--player` to create a command to execute.

This value can contain formatting variables surrounded by curly braces, { and }. If you need to include a brace character, it can be escaped by doubling, e.g. {{ and }}.

Formatting variables available:

filename This is the filename that the player will use. It's usually "-" (stdin), but can also be a URL or a file depending on the options used.

It's usually enough to use `--player` instead of this unless you need to add arguments after the filename.

Default is: **"{filename}"**.

-v, --verbose-player

Allow the player to display its console output.

-n, --player-fifo, --fifo

Make the player read the stream through a named pipe instead of the stdin pipe.

--player-http

Make the player read the stream through HTTP instead of the stdin pipe.

--player-continuous-http

Make the player read the stream through HTTP, but unlike `--player-http` it will continuously try to open the stream if the player requests it.

This makes it possible to handle stream disconnects if your player is capable of reconnecting to a HTTP stream. This is usually done by setting your player to a "repeat mode".

--player-external-http

Serve stream data through HTTP without running any player. This is useful to allow external devices like smartphones or streaming boxes to watch streams they wouldn't be able to otherwise.

Behavior will be similar to the continuous HTTP option, but no player program will be started, and the server will listen on all available connections instead of just in the local (loopback) interface.

The URLs that can be used to access the stream will be printed to the console, and the server can be interrupted using CTRL-C.

--player-external-http-port PORT

A fixed port to use for the external HTTP server if that mode is enabled. Omit or set to 0 to use a random high (>1024) port.

--player-passthrough TYPES

A comma-delimited list of stream types to pass to the player as a URL to let it handle the transport of the stream instead.

Stream types that can be converted into a playable URL are:

- hls
- http
- rtmp

Make sure your player can handle the stream type when using this.

--player-no-close
By default Livestreamer will close the player when the stream ends. This is to avoid "dead" GUI players lingering after a stream ends.

It does however have the side-effect of sometimes closing a player before it has played back all of its cached data.

This option will instead let the player decide when to exit.

File output options

-o FILENAME, **--output** FILENAME
Write stream data to FILENAME instead of playing it.

You will be prompted if the file already exists.

-f, --force
When using -o, always write to file even if it already exists.

-O, --stdout
Write stream data to stdout instead of playing it.

Stream options

--default-stream STREAM
Open this stream when no stream argument is specified, e.g. "best".

--retry-streams DELAY
Will retry fetching streams until streams are found while waiting DELAY (seconds) between each attempt.

--retry-open ATTEMPTS
Will try ATTEMPTS times to open the stream until giving up.

Default is: **1**.

--stream-types TYPES, **--stream-priority** TYPES
A comma-delimited list of stream types to allow.

The order will be used to separate streams when there are multiple streams with the same name but different stream types.

Default is: **"rtmp,hls,hds,http,akamaihd"**.

--stream-sorting-excludes STREAMS
Fine tune best/worst synonyms by excluding unwanted streams.

Uses a filter expression in the format:

```
[operator]<value>
```

Valid operators are >, >=, < and <=. If no operator is specified then equality is tested.

For example this will exclude streams ranked higher than "480p":

```
">480p"
```

Multiple filters can be used by separating each expression with a comma.

For example this will exclude streams from two quality types:

```
">480p,>medium"
```

Stream transport options

`--hds-live-edge` `SECONDS`
The time live HDS streams will start from the edge of stream.

Default is: **10.0**.

`--hds-segment-attempts` `ATTEMPTS`
How many attempts should be done to download each HDS segment before giving up.

Default is: **3**.

`--hds-segment-threads` `THREADS`
The size of the thread pool used to download HDS segments. Minimum value is 1 and maximum is 10.

Default is: **1**.

`--hds-segment-timeout` `TIMEOUT`
HDS segment connect and read timeout.

Default is: **10.0**.

`--hds-timeout` `TIMEOUT`
Timeout for reading data from HDS streams.

Default is: **60.0**.

`--hls-live-edge` `SEGMENTS`
How many segments from the end to start live HLS streams on.

The lower the value the lower latency from the source you will be, but also increases the chance of buffering.

Default is: **3**.

`--hls-segment-attempts` `ATTEMPTS`
How many attempts should be done to download each HLS segment before giving up.

Default is: **3**.

`--hls-segment-threads` `THREADS`
The size of the thread pool used to download HLS segments. Minimum value is 1 and maximum is 10.

Default is: **1**.

`--hls-segment-timeout` `TIMEOUT`
HLS segment connect and read timeout.

Default is: **10.0**.

`--hls-timeout` `TIMEOUT`
Timeout for reading data from HLS streams.

Default is: **60.0**.

`--http-stream-timeout` `TIMEOUT`
Timeout for reading data from HTTP streams.

Default is: **60.0**.

`--ringbuffer-size` `SIZE`
The maximum size of ringbuffer. Add a M or K suffix to specify mega or kilo bytes instead of bytes.

The ringbuffer is used as a temporary storage between the stream and the player. This is to allows us to download the stream faster than the player wants to read it.

The smaller the size, the higher chance of the player buffering if there are download speed dips and the higher size the more data we can use as a storage to catch up from speed dips.

It also allows you to temporary pause as long as the ringbuffer doesn't get full since we continue to download the stream in the background.

Note: A smaller size is recommended on lower end systems (such as Raspberry Pi) when playing stream types that require some extra processing (such as HDS) to avoid unnecessary background processing.

Default is: **"16M"**.

`--rtmp-proxy` `PROXY`, **`--rtmpdump-proxy`** `PROXY`
A SOCKS proxy that RTMP streams will use.

Example: 127.0.0.1:9050

`--rtmp-rtmpdump` `FILENAME`, **`--rtmpdump`** `FILENAME`, **`-r`** `FILENAME`
RTMPDump is used to access RTMP streams. You can specify the location of the rtmpdump executable if it is not in your PATH.

Example: "/usr/local/bin/rtmpdump"

`--rtmp-timeout` `TIMEOUT`
Timeout for reading data from RTMP streams.

Default is: **60.0**.

`--stream-segment-attempts` `ATTEMPTS`
How many attempts should be done to download each segment before giving up.

This is generic option used by streams not covered by other options, such as stream protocols specific to plugins, e.g. UStream.

Default is: **3**.

`--stream-segment-threads` `THREADS`
The size of the thread pool used to download segments. Minimum value is 1 and maximum is 10.

This is generic option used by streams not covered by other options, such as stream protocols specific to plugins, e.g. UStream.

Default is: **1**.

`--stream-segment-timeout` `TIMEOUT`
Segment connect and read timeout.

This is generic option used by streams not covered by other options, such as stream protocols specific to plugins, e.g. UStream.

Default is: **10.0**.

`--stream-timeout` `TIMEOUT`
Timeout for reading data from streams.

This is generic option used by streams not covered by other options, such as stream protocols specific to plugins, e.g. UStream.

Default is: **60.0**.

`--stream-url`
If possible, translate the stream to a URL and print it.

`--subprocess-cmdline, --cmdline, -c`
Print command-line used internally to play stream.

This is only available on RTMP streams.

`--subprocess-errorlog, --errorlog, -e`
Log possible errors from internal subprocesses to a temporary file. The file will be saved in your systems temporary directory.

Useful when debugging rtmpdump related issues.

HTTP options

`--http-proxy` `HTTP_PROXY`
A HTTP proxy to use for all HTTP requests.

Example: http://hostname:port/

`--https-proxy` `HTTPS_PROXY`
A HTTPS capable proxy to use for all HTTPS requests.

Example: http://hostname:port/

`--http-cookie` `KEY=VALUE`
A cookie to add to each HTTP request.

Can be repeated to add multiple cookies.

`--http-header` `KEY=VALUE`
A header to add to each HTTP request.

Can be repeated to add multiple headers.

`--http-query-param` `KEY=VALUE`
A query parameter to add to each HTTP request.

Can be repeated to add multiple query parameters.

`--http-ignore-env`
Ignore HTTP settings set in the environment such as environment variables (HTTP_PROXY, etc) or ~/.netrc authentication.

`--http-no-ssl-verify`
Don't attempt to verify SSL certificates.

Usually a bad idea, only use this if you know what you're doing.

`--http-ssl-cert` `FILENAME`
SSL certificate to use.

Expects a .pem file.

--http-ssl-cert-crt-key `CRT_FILENAME KEY_FILENAME`
SSL certificate to use.

Expects a .crt and a .key file.

--http-timeout `TIMEOUT`
General timeout used by all HTTP requests except the ones covered by other options.

Default is: **20.0**.

Plugin options

--plugin-dirs `DIRECTORY`
Attempts to load plugins from these directories.

Multiple directories can be used by separating them with a semi-colon.

--twitch-oauth-token `TOKEN`
An OAuth token to use for Twitch authentication. Use `--twitch-oauth-authenticate` to create a token.

--twitch-oauth-authenticate
Open a web browser where you can grant Livestreamer access to your Twitch account which creates a token for use with `--twitch-oauth-token`.

--twitch-cookie `COOKIES`
Twitch cookies to authenticate to allow access to subscription channels.

Example:

```
"_twitch_session_id=xxxxxx; persistent=xxxxx"
```

Note: This method is the old and clunky way of authenticating with Twitch, using `--twitch-oauth-authenticate` is the recommended and simpler way of doing it now.

--ustream-password `PASSWORD`
A password to access password protected UStream.tv channels.

--crunchyroll-username `USERNAME`
A Crunchyroll username to allow access to restricted streams.

--crunchyroll-password `[PASSWORD]`
A Crunchyroll password for use with `--crunchyroll-username`.

If left blank you will be prompted.

--crunchyroll-purge-credentials
Purge cached Crunchyroll credentials to initiate a new session and reauthenticate.

--livestation-email `EMAIL`
A Livestation account email to access restricted or premium quality streams.

--livestation-password `PASSWORD`
A Livestation account password to use with `--livestation-email`.

3.3 Plugins

This is a list of the currently built in plugins and what URLs and features they support. Livestreamer's primary focus is live streams, so VOD support is limited.

Name	URL(s)	Live	VOD	Notes
afreeca	afreecatv.com	Yes	No	
afreecatv	afreeca.tv	Yes	No	
aftonbladet	aftonbladet.se	Yes	Yes	
alieztv	aliez.tv	Yes	Yes	
ard_live	live.daserste.de	Yes	–	Streams may be geo-restricted to Germany.
ard_mediathek	ardmediathek.de	Yes	Yes	Streams may be geo-restricted to Germany.
artetv	arte.tv	Yes	Yes	
azubutv	azubu.tv	Yes	No	
beattv	be-at.tv	Yes	Yes	Playlist not implemented yet.
bambuser	bambuser.com	Yes	Yes	
chaturbate	chaturbate.com	Yes	No	
connectcast	connectcast.tv	Yes	Yes	
crunchyroll	crunchyroll.com	–	Yes	
cybergame	cybergame.tv	Yes	Yes	
dailymotion	dailymotion.com	Yes	Yes	
disney_de	• video.disney.de • disneychannel.de	Yes	Yes	Streams may be geo-restricted to Germany.
dommune	dommune.com	Yes	–	
douyutv	douyutv.com	Yes	–	
dmcloud	api.dmcloud.net	Yes	–	
drdk	dr.dk	Yes	Yes	Streams may be geo-restricted to Denmark.
euronews	euronews.com	Yes	No	
filmon	filmon.com	Yes	Yes	Only SD quality streams.
filmon_us	filmon.us	Yes	Yes	
furstream	furstre.am	Yes	No	
gaminglive	gaminglive.tv	Yes	Yes	
gomexp	gomexp.com	Yes	No	
goodgame	goodgame.ru	Yes	No	Only HLS streams are available.
hitbox	hitbox.tv	Yes	Yes	
itvplayer	itv.com/itvplayer	Yes	Yes	Streams may be geo-restricted to Great Britain.

Continued on next page

Table 3.1 – continued from previous page

Name	URL(s)	Live	VOD	Notes
letontv	leton.tv	Yes	–	
livestation	livestation.com	Yes	–	
livestream	new.livestream.com	Yes	–	
media_ccc_de	• media.ccc.de • streaming...[4]	Yes	Yes	Only mp4 and HLS ar are supported.
mips	mips.tv	Yes	–	Requires rtmpdump with K-S-V patches.
mlgtv	mlg.tv	Yes	–	
nhkworld	nhk.or.jp/nhkworld	Yes	No	
nos	nos.nl	Yes	Yes	Streams may be geo-restricted to Netherlands.
npo	npo.nl	Yes	Yes	Streams may be geo-restricted to Netherlands.
nrk	• tv.nrk.no • radio.nrk.no	Yes	Yes	Streams may be geo-restricted to Norway.
oldlivestream	original.liv...[3]	Yes	No	Only mobile streams are supported.
periscope	periscope.tv	Yes	–	
picarto	picarto.tv	Yes	–	
rtve	rtve.es	Yes	No	
sbsdiscovery	• kanal5play.se • kanal9play.se • kanal11play.se	–	Yes	
seemeplay	seemeplay.ru	Yes	Yes	
speedrunslive	speedrunslive.com	Yes	–	URL forwarder to Twitch channels.
ssh101	ssh101.com	Yes	No	
streamingvi...[1]	streamingvid...[2]	Yes	–	RTMP streams requires rtmpdump with K-S-V patches.
streamlive	streamlive.to	Yes	–	
svtplay	• svtplay.se • svtflow.se • oppetarkiv.se	Yes	Yes	Streams may be geo-restricted to Sweden.
tga	• star.plu.cn • star.tga.plu.cn	Yes	No	

Continued on next page

Table 3.1 – continued from previous page

Name	URL(s)	Live	VOD	Notes
tv3cat	tv3.cat	Yes	Yes	Streams may be geo-restricted to Spain.
tv4play	• tv4play.se • fotbollskanalen.se	Yes	Yes	Streams may be geo-restricted to Sweden. Only non-premium streams currently supported.
tvcatchup	• tvcatchup.com	Yes	No	Streams may be geo-restricted to Great Britain.
tvplayer	tvplayer.com	Yes	No	
twitch	twitch.tv	Yes	Yes	Possible to authenticate for access to subscription streams.
ustreamtv	ustream.tv	Yes	Yes	
vaughnlive	• vaughnlive.tv • breakers.tv • instagib.tv • vapers.tv	Yes	–	
veetle	veetle.com	Yes	Yes	
vgtv	vgtv.no	Yes	Yes	
viagame	viagame.com			
viasat	• tv3play.se • tv3play.no • tv3play.dk • tv3play.ee • tv3play.lt • tv3play.lv • tv6play.se • tv6play.no • tv8play.se • tv10play.se • viasat4play.no • play.tv3.lt • juicyplay.se	Yes	Yes	Streams may be geo-restricted.
wattv	wat.tv	–	Yes	
weeb	weeb.tv	Yes	–	Requires rtmpdump with K-S-V patches.
youtube	• youtube.com • youtu.be	Yes	Yes	Protected videos are not supported.

Continued on next page

Table 3.1 – continued from previous page

Name	URL(s)	Live	VOD	Notes
zdf_mediathek	zdf.de	Yes	Yes	

3.4 Players

3.4.1 Transport modes

There are three different modes of transporting the stream to the player.

Name	Description
Standard input pipe	This is the default behaviour when there are no other options specified.
Named pipe (FIFO)	Use the *--player-fifo* option to enable.
HTTP	Use the *--player-http* or *--player-continuous-http* options to enable.

3.4.2 Player compatibility

This is a list of video players and their compatibility with the transport modes.

Name	Stdin Pipe	Named Pipe	HTTP
Daum Pot Player	No	No	Yes[1]
MPC-HC	Yes[2]	No	Yes[1]
MPlayer	Yes	Yes	Yes
MPlayer2	Yes	Yes	Yes
mpv	Yes	Yes	Yes
QuickTime	No	No	No
VLC media player	Yes[3]	Yes	Yes

3.4.3 Known issues and workarounds

MPC-HC reports "File not found"

Upgrading to version 1.7 or newer will solve this issue since reading data from standard input is not supported in version 1.6.x of MPC-HC.

MPC-HC only plays sound on Twitch streams

Twitch sometimes returns badly muxed streams which may confuse players. The following workaround was contributed by MPC-HC developer @kasper93:

[4] streaming.media.ccc.de

[3] original.livestream.com

[1] streamingvideoprovider

[2] streamingvideoprovider.co.uk

[1] *--player-continuous-http* must be used. Using HTTP with players that rely on Windows' codecs to access HTTP streams may have a long startup time since Windows tend to do multiple HTTP requests and Livestreamer will attempt to open the stream for each request.

[2] Stdin requires MPC-HC 1.7 or newer.

[3] Some versions of VLC might be unable to use the stdin pipe and prints the error message:

```
VLC is unable to open the MRL 'fd://0'
```

Use one of the other transport methods instead to work around this.

> *To fix this problem go to options -> internal filters -> open splitter settings and increase "Stream Analysis Duration" this will let ffmpeg to properly detect all streams.*

Using `--player-passthrough hls` has also been reported to work.

MPlayer tries to play Twitch streams at the wrong FPS

This is a bug in MPlayer, using the MPlayer fork mpv instead is recommended.

VLC hangs when buffering and no playback starts

Some versions of 64-bit VLC seem to be unable to read the stream created by rtmpdump. Using the 32-bit version of VLC might help.

3.5 Common issues

3.5.1 Streams are buffering/lagging

Enable caching in your player

By default most players do not cache the data they receieve from Livestreamer. Caching can reduce the amount of buffering you run into because the player will have some breathing room between receving the data and playing it.

Player	Parameter	Note
MPC-HC	–	Currently no way of configuring the cache
MPlayer	`-cache <kbytes>`	Between 1024 and 8192 is recommended
MPlayer2	`-cache <kbytes>`	Between 1024 and 8192 is recommended
mpv	`--cache <kbytes>`	Between 1024 and 8192 is recommended
VLC	`--file-caching <ms> --network-caching <ms>`	Between 1000 and 10000 is recommended

Use the `--player` option to pass these options to your player.

Multi-threaded streaming

On segmented streaming protocols (such as HLS and HDS) it's possible to use multiple threads to potentially increase the throughput. Each stream type has it's own option, these options are currently available:

Option	Used by these plugins
`--hls-segment-threads`	*twitch*, *youtube* and many more.
`--hds-segment-threads`	*dailymotion*, *mlgtv* and many more.
`--stream-segment-threads`	*ustreamtv*, *beattv* and any other plugins implementing their own segmented streaming protocol.

Note: Using 2 or 3 threads should be enough to see an impact on live streams, any more will likely not show much effect.

3.6 API Guide

This API is what powers the *Command-Line Interface* but is also available to developers that wish to make use of the data Livestreamer can retrieve in their own application.

3.6.1 Extracting streams

The simplest use of the Livestreamer API looks like this:

```
>>> import livestreamer
>>> streams = livestreamer.streams("http://twitch.tv/day9tv")
```

This simply attempts to find a plugin and use it to extract streams from the URL. This works great in simple cases but if you want more fine tuning you need to use a *session object* instead.

The returned value is a dict containing `Stream` objects:

```
>>> streams
{'best': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>,
 'high': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>,
 'low': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>,
 'medium': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>,
 'mobile': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>,
 'source': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>,
 'worst': <HLSStream('http://video11.fra01.hls.twitch.tv/ ...')>}
```

If no plugin for the URL is found, a `NoPluginError` will be raised. If an error occurs while fetching streams, a `PluginError` will be raised.

3.6.2 Opening streams to read data

Now that you have extracted some streams we might want to read some data from one of them. When you call *open()* on a stream, a file-like object will be returned, which you can call *.read(size)* and *.close()* on.

```
>>> stream = streams["source"]
>>> fd = stream.open()
>>> data = fd.read(1024)
>>> fd.close()
```

If an error occurs while opening a stream, a `StreamError` will be raised.

3.6.3 Inspecting streams

It's also possible to inspect streams internal parameters, see *Stream subclasses* to see what attributes are available for inspection for each stream type.

For example this is a `HLSStream` object which contains a *url* attribute.

```
>>> stream.url
'http://video38.ams01.hls.twitch.tv/hls11/ ...'
```

3.6.4 Session object

The session allows you to set various options and is more efficient when extracting streams more than once. You start by creating a *Livestreamer* object:

```
>>> from livestreamer import Livestreamer
>>> session = Livestreamer()
```

You can then extract streams like this:

```
>>> streams = session.streams("http://twitch.tv/day9tv")
```

or set options like this:

```
>>> session.set_option("rtmp-rtmpdump", "/path/to/rtmpdump")
```

See *Livestreamer.set_option()* to see which options are available.

3.6.5 Examples

Simple player

This example uses the PyGObject module to playback a stream using the GStreamer framework.

```
#!/usr/bin/env python

from __future__ import print_function

import sys

import gi

from gi.repository import GObject as gobject, Gst as gst
from livestreamer import Livestreamer, StreamError, PluginError, NoPluginError


def exit(msg):
    print(msg, file=sys.stderr)
    sys.exit()


class LivestreamerPlayer(object):
    def __init__(self):
        self.fd = None
        self.mainloop = gobject.MainLoop()

        # This creates a playbin pipeline and using the appsrc source
        # we can feed it our stream data
        self.pipeline = gst.ElementFactory.make("playbin", None)
        self.pipeline.set_property("uri", "appsrc://")

        # When the playbin creates the appsrc source it will call
        # this callback and allow us to configure it
        self.pipeline.connect("source-setup", self.on_source_setup)

        # Creates a bus and set callbacks to receive errors
        self.bus = self.pipeline.get_bus()
        self.bus.add_signal_watch()
        self.bus.connect("message::eos", self.on_eos)
        self.bus.connect("message::error", self.on_error)

    def exit(self, msg):
        self.stop()
        exit(msg)

    def stop(self):
        # Stop playback and exit mainloop
        self.pipeline.set_state(gst.State.NULL)
        self.mainloop.quit()

        # Close the stream
        if self.fd:
            self.fd.close()

    def play(self, stream):
        # Attempt to open the stream
        try:
            self.fd = stream.open()
        except StreamError as err:
            self.exit("Failed to open stream: {0}".format(err))

        # Start playback
        self.pipeline.set_state(gst.State.PLAYING)
        self.mainloop.run()

    def on_source_setup(self, element, source):
        # When this callback is called the appsrc expects
        # us to feed it more data
        source.connect("need-data", self.on_source_need_data)

    def on_source_need_data(self, source, length):
        # Attempt to read data from the stream
        try:
            data = self.fd.read(length)
        except IOError as err:
            self.exit("Failed to read data from stream: {0}".format(err))

        # If data is empty it's the end of stream
        if not data:
            source.emit("end-of-stream")
            return

        # Convert the Python bytes into a GStreamer Buffer
        # and then push it to the appsrc
        buf = gst.Buffer.new_wrapped(data)
        source.emit("push-buffer", buf)

    def on_eos(self, bus, msg):
        # Stop playback on end of stream
        self.stop()

    def on_error(self, bus, msg):
        # Print error message and exit on error
        error = msg.parse_error()[1]
        self.exit(error)
```

```
def main():
    if len(sys.argv) < 3:
        exit("Usage: {0} <url> <quality>".format(sys.argv[0]))

    # Initialize and check GStreamer version
    gi.require_version("Gst", "1.0")
    gobject.threads_init()
    gst.init(None)

    # Collect arguments
    url = sys.argv[1]
    quality = sys.argv[2]

    # Create the Livestreamer session
    livestreamer = Livestreamer()

    # Enable logging
    livestreamer.set_loglevel("info")
    livestreamer.set_logoutput(sys.stdout)

    # Attempt to fetch streams
    try:
        streams = livestreamer.streams(url)
    except NoPluginError:
        exit("Livestreamer is unable to handle the URL '{0}'".format(url))
    except PluginError as err:
        exit("Plugin error: {0}".format(err))

    if not streams:
        exit("No streams found on URL '{0}'".format(url))

    # Look for specified stream
    if quality not in streams:
        exit("Unable to find '{0}' stream on URL '{1}'".format(quality, url))

    # We found the stream
    stream = streams[quality]

    # Create the player and start playback
    player = LivestreamerPlayer()

    # Blocks until playback is done
    player.play(stream)

if __name__ == "__main__":
    main()
```

3.7 API Reference

This ia reference of all the available API methods in Livestreamer.

3.7.1 Livestreamer

`livestreamer.`**`streams`**(*url*, ***params*)

Attempts to find a plugin and extract streams from the *url*.

params are passed to `Plugin.streams()`.

Raises *`NoPluginError`* if no plugin is found.

3.7.2 Session

class `livestreamer.`**`Livestreamer`**

A Livestreamer session is used to keep track of plugins, options and log settings.

`get_option`(*key*)

Returns current value of specified option.

Parameters **`key`** – key of the option

`get_plugin_option`(*plugin*, *key*)

Returns current value of plugin specific option.

Parameters

- **`plugin`** – name of the plugin
- **`key`** – key of the option

`get_plugins`()

Returns the loaded plugins for the session.

`load_plugins`(*path*)

Attempt to load plugins from the path specified.

Parameters **`path`** – full path to a directory where to look for plugins

`resolve_url`(*url*)

Attempts to find a plugin that can use this URL.

The default protocol (http) will be prefixed to the URL if not specified.

Raises *`NoPluginError`* on failure.

Parameters **`url`** – a URL to match against loaded plugins

`set_loglevel`(*level*)

Sets the log level used by this session.

Valid levels are: “none”, “error”, “warning”, “info” and “debug”.

Parameters **`level`** – level of logging to output

`set_logoutput`(*output*)

Sets the log output used by this session.

Parameters **`output`** – a file-like object with a write method

`set_option`(*key*, *value*)

Sets general options used by plugins and streams originating from this session object.

Parameters

- **`key`** – key of the option
- **`value`** – value to set the option to

Available options:

hds-live-edge	(float) Specify the time live HDS streams will start from the edge of stream, default: `10.0`
hds-segment-attempts	(int) How many attempts should be done to download each HDS segment, default: `3`
hds-segment-threads	(int) The size of the thread pool used to download segments, default: `1`
hds-segment-timeout	(float) HDS segment connect and read timeout, default: `10.0`
hds-timeout	(float) Timeout for reading data from HDS streams, default: `60.0`
hls-live-edge	(int) How many segments from the end to start live streams on, default: `3`
hls-segment-attempts	(int) How many attempts should be done to download each HLS segment, default: `3`
hls-segment-threads	(int) The size of the thread pool used to download segments, default: `1`
hls-segment-timeout	(float) HLS segment connect and read timeout, default: `10.0`
hls-timeout	(float) Timeout for reading data from HLS streams, default: `60.0`
http-proxy	(str) Specify a HTTP proxy to use for all HTTP requests
https-proxy	(str) Specify a HTTPS proxy to use for all HTTPS requests
http-cookies	(dict or str) A dict or a semi-colon (;) delimited str of cookies to add to each HTTP request, e.g. `foo=bar;baz=qux`
http-headers	(dict or str) A dict or semi-colon (;) delimited str of headers to add to each HTTP request, e.g. `foo=bar;baz=qux`
http-query-params	(dict or str) A dict or a ampersand (&) delimited string of query parameters to add to each HTTP request, e.g. `foo=bar&baz=qux`
http-trust-env	(bool) Trust HTTP settings set in the environment, such as environment variables (HTTP_PROXY, etc) and ~/.netrc authentication
http-ssl-verify	(bool) Verify SSL certificates, default: `True`
http-ssl-cert	(str or tuple) SSL certificate to use, can be either a .pem file (str) or a .crt/.key pair (tuple)
http-timeout	(float) General timeout used by all HTTP requests except the ones covered by other options, default: `20.0`
http-stream-timeout	(float) Timeout for reading data from HTTP streams, default: `60.0`
subprocess-errorlog	(bool) Log errors from subprocesses to a file located in the temp directory
ringbuffer-size	(int) The size of the internal ring buffer used by most stream types, default: `16777216` (16MB)
rtmp-proxy	(str) Specify a proxy (SOCKS) that RTMP streams will use
rtmp-rtmpdump	(str) Specify the location of the rtmpdump executable used by RTMP streams, e.g. `/usr/local/bin/rtmpdump`
rtmp-timeout	(float) Timeout for reading data from RTMP streams, default: `60.0`
stream-segment-attempts	(int) How many attempts should be done to download each segment, default: `3`. General option used by streams not covered by other options.
stream-segment-threads	(int) The size of the thread pool used to download segments, default: `1`. General option used by streams not covered by other options.
stream-segment-timeout	(float) Segment connect and read timeout, default: `10.0`. General option used by streams not covered by other options.
stream-timeout	(float) Timeout for reading data from stream, default: `60.0`. General option used by streams not covered by other options.

`set_plugin_option`(*plugin*, *key*, *value*)
Sets plugin specific options used by plugins originating from this session object.

Parameters

- **`plugin`** – name of the plugin
- **`key`** – key of the option
- **`value`** – value to set the option to

`streams`(*url*, ***params*)
Attempts to find a plugin and extract streams from the *url*.

params are passed to `Plugin.streams()`.

Raises *`NoPluginError`* if no plugin is found.

3.7.3 Plugins

class `livestreamer.plugin.`**`Plugin`**(*url*)
A plugin can retrieve stream information from the URL specified.

Parameters **`url`** – URL that the plugin will operate on

`get_streams`(**args*, ***kwargs*)
Deprecated since version 1.9.0.

Has been renamed to *`Plugin.streams()`*, this is an alias for backwards compatibility.

`streams`(*stream_types=None*, *sorting_excludes=None*)
Attempts to extract available streams.

Returns a `dict` containing the streams, where the key is the name of the stream, most commonly the quality and the value is a `Stream` object.

The result can contain the synonyms **best** and **worst** which points to the streams which are likely to be of highest and lowest quality respectively.

If multiple streams with the same name are found, the order of streams specified in *stream_types* will determine which stream gets to keep the name while the rest will be renamed to "<name>_<stream type>".

The synonyms can be fine tuned with the *sorting_excludes* parameter. This can be either of these types:

- A list of filter expressions in the format *[operator]<value>*. For example the filter ">480p" will exclude streams ranked higher than "480p" from the list used in the synonyms ranking. Valid operators are >, >=, < and <=. If no operator is specified then equality will be tested.
- A function that is passed to filter() with a list of stream names as input.

Parameters

- **`stream_types`** – A list of stream types to return.
- **`sorting_excludes`** – Specify which streams to exclude from the best/worst synonyms.

Changed in version 1.4.2: Added *priority* parameter.

Changed in version 1.5.0: Renamed *priority* to *stream_types* and changed behaviour slightly.

Changed in version 1.5.0: Added *sorting_excludes* parameter.

Changed in version 1.6.0: *sorting_excludes* can now be a list of filter expressions or a function that is passed to filter().

3.7.4 Streams

All streams inherit from the `Stream` class.

class `livestreamer.stream.`**`Stream`**(*session*)

`open`()
Attempts to open a connection to the stream. Returns a file-like object that can be used to read the stream data.

Raises `StreamError` on failure.

Stream subclasses

You are able to inspect the parameters used by each stream, different properties are available depending on stream type.

class `livestreamer.stream.`**`AkamaiHDStream`**(*session*, *url*, *swf=None*, *seek=None*)
Implements the AkamaiHD Adaptive Streaming protocol

Attributes:

- `url` URL to the stream
- `swf` URL to a SWF used by the handshake protocol
- `seek` Position to seek to when opening the stream

class `livestreamer.stream.`**`HDSStream`**(*session*, *baseurl*, *url*, *bootstrap*, *metadata=None*, *timeout=60*, ***request_params*)
Implements the Adobe HTTP Dynamic Streaming protocol

Attributes:

- `baseurl` Base URL
- `url` Base path of the stream, joined with the base URL when fetching fragments
- `bootstrap` Either a URL pointing to the bootstrap or a bootstrap `Box` object used for initial information about the stream
- `metadata` Either *None* or a `ScriptData` object that contains metadata about the stream, such as height, width and bitrate

classmethod **`parse_manifest`**(*session*, *url*, *timeout=60*, *pvswf=None*, ***request_params*)
Parses a HDS manifest and returns its substreams.

Parameters

- **`url`** – The URL to the manifest.
- **`timeout`** – How long to wait for data to be returned from from the stream before raising an error.
- **`pvswf`** – URL of player SWF for Akamai HD player verification.

class `livestreamer.stream.`**`HLSStream`**(*session_*, *url*, ***args*)
Implementation of the Apple HTTP Live Streaming protocol

Attributes:

- `url` The URL to the HLS playlist.
- `args` A `dict` containing keyword arguments passed to `requests.request()`, such as headers and cookies.

Changed in version 1.7.0: Added *args* attribute.

classmethod **`parse_variant_playlist`**(*session_*, *url*, *name_key='name'*, *name_prefix=''*, *check_streams=False*, ***request_params*)
Attempts to parse a variant playlist and return its streams.

Parameters

- **`url`** – The URL of the variant playlist.
- **`name_key`** – Prefer to use this key as stream name, valid keys are: name, pixels, bitrate.
- **`name_prefix`** – Add this prefix to the stream names.
- **`check_streams`** – Only allow streams that are accesible.

class `livestreamer.stream.`**`HTTPStream`**(*session_*, *url*, *buffered=True*, ***args*)
A HTTP stream using the requests library.

Attributes:

- `url` The URL to the stream, prepared by requests.
- `args` A `dict` containing keyword arguments passed to `requests.request()`, such as headers and cookies.

class `livestreamer.stream.`**`RTMPStream`**(*session*, *params*, *redirect=False*)
RTMP stream using rtmpdump.

Attributes:

- `params` A `dict` containing parameters passed to rtmpdump

3.7.5 Exceptions

Livestreamer has three types of exceptions:

exception `livestreamer.`**`LivestreamerError`**
Any error caused by Livestreamer will be caught with this exception.

exception `livestreamer.`**`PluginError`**
Plugin related error.

exception `livestreamer.`**`NoPluginError`**
No relevant plugin has been loaded.

exception `livestreamer.`**`StreamError`**
Stream related error.

Python Module Index

Index