avtk

Release 0.1.0

Senko Rasic

May 10, 2019

CONTENTS

AVTK provides functionality for inspecting and manipulating audio and video contents in an easy to use manner.

Here's a quick example:

```
>>> from avtk.backends.ffmpeg.shortcuts import convert_to_webm

>>> convert_to_webm('test-media/video/sintel.mkv', '/tmp/output.webm')
```

CHAPTER

ONE

CONTENTS

1.1 AVTK - Python Audio/Video Toolkit

AVTK provides functionality for inspecting and manipulating audio and video contents in an easy to use manner.

Current version supports working with local files using the FFmpeg command line tools (`ffmpeg` and `ffprobe`). Future versions will support various online video providers by wrapping their respective low-level SDKs.

AVTK aims to make it easy to most of the things you might want to do with audio and video, while allowing you to use the underlying tools and libraries for edge cases where you need full low-level functionality.

WARNING: AVTK is still in beta phase - bugs, missing documentation and frequent API changes are likely. Please report bugs, suggest use cases and feature ideas, and any other feedback you might have, to GitHub issues.

1.1.1 Resources

- Python Package Index: https://pypi.org/project/avtk/
- Documentation: https://avtk.readthedocs.io/en/latest/
- Source code: https://github.com/senko/avtk/
- Bug and feature tracker: https://github.com/senko/avtk/issues

1.1.2 Quickstart

Install from Python Package Index:

```
pip install avtk
```

Make sure you have the `ffmpeg` and `ffprobe` tools available on the system. For detailed installation instructions please see the *Installation* page in the documentation.

1.1.3 Examples

Converting a video:

```
>>> from avtk.backends.ffmpeg.shortcuts import convert_to_webm

>>> convert_to_webm('test-media/video/sintel.mkv', '/tmp/output.webm')
```

Creating a thumbnail:

```
>>> from datetime import timedelta
>>> from avtk.backends.ffmpeg.shortcuts import get_thumbnail

>>> png_data = get_thumbnail('test-media/video/sintel.mkv', timedelta(seconds=2))
```

Inspecting a media file:

```
>>> from avtk.backends.ffmpeg.shortcuts import inspect

>>> info = inspect('test-media/video/sintel.mkv')
>>> info.format.duration
datetime.timedelta(seconds=5, microseconds=24000)
>>> info.format.bit_rate
2657539
>>> info.has_video
True
>>> info.video_streams[0].codec
<Codec(name=h264, type=video)>
>>> info.has_audio
True
>>> info.audio_streams[0].codec
<Codec(name=ac3, type=audio)>
>>> info.has_subtitles
True
```

For more examples, see the *avtk.backends.ffmpeg.shortcuts* documentation.

1.1.4 Bugs and feature requests

To report a bug or suggest a feature, please use the GitHub issue tracker.

When reporting a bug, please include a detailed description, any logging output and exception tracebacks, and media files (direcly or links to) which caused the problem.

When suggesting a feature, please explain both the feature (how it would work and what it would do) and rationale (why it is needed, especially if the same can be combined with the existing API).

If contributing code, either bugfixes or feature implementation, please include tests for the new or fixed functionality, and update documentation accordingly.

1.2 Installation

The recommended way to install AVTK is from the Python Package Index:

```
pip install avtk
```

If you're contributing to the project or want to check out the latest source code, you can install directly from GitHub:

```
pip install git+https://github.com/senko/avtk.git
```

Or you can manually download the source code and run `setup.py` to install:

```
python setup.py build
python setup.pby install
```

1.2.1 Requirements

AVTK depends on the FFmpeg command-line tools for manipulating local media files. Follow the official installation instructions to install FFmpeg.

Note that due to licensing and patent issues, the pre-built ffmpeg package on your operating system may have limited or no support for certain codecs, formats or protocols. If you're unsure, we recomend you download a static build that comes with most functionality out of the box.

1.3 Local file support via FFmpeg

The `avtk.backends.ffmpeg` module implements inspection and manipulation of local audio and video files using FFmpeg. In order to use this module, you'll need to have `ffmpeg` and `ffprobe` tools installed.

1.3.1 Convenience shortcut functions

The module *avtk.backends.ffmpeg.shortcuts* contains convenience helpers for often-used functionality of the ffmpeg backend.

These functions are just wrappers around the lower-level FFmpeg and Probe modules.

THUMBNAIL_FORMATS = ['png', 'jpg', 'gif', 'tiff', 'bmp']
Supported thumbnail formats

inspect(*source*)
Inspects a media file and returns information about it.

See the *MediaInfo* documentation for a detailed description of the returned information.

Parameters **source** (*str*) – Local file path or stream URL to inspect

Returns Information about the inspected file or stream

Return type MediaInfo

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

Example:

```
>>> from avtk.backends.ffmpeg.shortcuts import inspect

>>> info = inspect('test-media/video/sintel.mkv')
>>> info.format.duration
datetime.timedelta(seconds=5, microseconds=24000)
>>> info.format.bit_rate
2657539
>>> info.has_video
True
>>> info.video_streams[0].codec
<Codec(name=h264, type=video)>
>>> info.has_audio
True
>>> info.audio_streams[0].codec
<Codec(name=ac3, type=audio)>
>>> info.has_subtitles
True
```

get_thumbnail(*source*, *seek*, *fmt='png'*)
Extracts a thumbnail from a video file.

Parameters

- **source** (*str*) – file path
- **seek** (*timedelta*, *int* or *float*) – position (in seconds) from which to get the thumbnail
- **fmt** (*str*) – output image format (one of *THUMBNAIL_FORMATS*)

Returns thumbnail data as a binary string in the specified format

Return type bytes

Raises

- **ValueError** – if image format is not supported
- ***NoMediaError*** – if source doesn't exist or is of unknown format

Example:

```
from avtk.backends.ffmpeg.shortcuts import get_thumbnail

png_data = get_thumbnail('test-media/video/sintel.mkv', timedelta(seconds=2))
with open('/tmp/thumb.png', 'wb') as fp:
    fp.write(png_data)
```

extract_audio(*source*, *output*, *fmt=None*, *codec=None*, *channels=None*)
Extracts audio from the source media file and saves it to a separate output file.

If codec is not specified, audio data will be copied directly from the input file, preserving quality. If the codec is specified, audio will be re-encoded. The transcode process is slower than pure copy and some quality loss is unavoidable.

Note that the *codec* parameter is actually an encoder name, and should be one of the supported encoders from *get_available_encoders()*. The somewhat-confusing *codec* name for the parameter is kept to be consistent with ffmpeg naming.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **fmt** (*str*) – output file format (optional, default: guess from output file name)
- **codec** (*str*) – output file codec (optional, default: copy from input)
- **channels** (*int*) – downmix to specified number of channels (optional)

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

Example:

```
from avtk.backends.ffmpeg.shortcuts import extract_audio

extract_audio('test-media/video/sintel.mkv', '/tmp/output.ogg', codec='libopus',␣
↪channels=2)
```

remove_audio(*source*, *output*, *fmt=None*, *remove_subtitles=True*)
Creates an output video file from the source with all audio streams removed.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **fmt** (*str*) – output file format (optional, default: guess from output file name)
- **remove_subtitles** (*bool*) – whether to remove subtitle streams as well (optional, default: True)

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

Example:

```
from avtk.backends.ffmpeg.shortcuts import remove_audio

remove_audio('test-media/video/sintel.mkv', '/tmp/silent.mkv')
```

convert_to_h264(*source*, *output*, *preset=None*, *crf=None*, *video_bitrate=None*, *audio_bitrate=None*, ***kwargs*)

Converts a video file to MP4 format using H264 for video and AAC for audio.

See https://trac.ffmpeg.org/wiki/Encode/H.264 for tips on H.264 encoding with ffmpeg, and for description of CRF and preset parameters.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **preset** (*str*) – encoder preset (quality / encoding speed tradeoff) - optional
- **crf** (*str*) – constant rate factor (determines target quality) - optional
- **video_bitrate** (*int or str*) – target video bitrate - optional
- **audio_bitrate** (*int or str*) – target audio bitrate - optional
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments - optional

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

If *scale* is set, either width or height may be `-1` to use the optimal size for preserving aspect ratio, or `0` to keep the original size.

Bitrates should be specified as integers or as strings in 'NUMk' or 'NUMm' format.

Example:

```
from avtk.backends.ffmpeg.shortcuts import convert_to_h264

convert_to_h264(
    'test-media/video/sintel.mkv',
    '/tmp/out.mkv',
    preset='fast',
    crf=23,
    video_bitrate='2m',
    audio_bitrate='128k'
)
```

convert_to_webm(*source*, *output*, *crf=None*, *audio_bitrate=None*, ***kwrags*)

Converts a video file to WebM format using VP9 for video and Opus for audio.

Uses a single-pass constant quality encode process for VP9. See https://trac.ffmpeg.org/wiki/Encode/VP9 for tips on VP9 encoding with ffmpeg, and for description of the CRF parameter.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **crf** (*str*) – constant rate factor (determines target quality) - optional, default is 31
- **audio_bitrate** (*int or str*) – target audio bitrate - optional
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments - optional

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

If *scale* is set, either width or height may be `-1` to use the optimal size for preserving aspect ratio, or `0` to keep the original size.

Audio bitrate should be specified as integer or as string in 'NUMk' format.

Example:

```
from avtk.backends.ffmpeg.shortcuts import convert_to_webm

convert_to_webm(
    'test-media/video/sintel.mkv',
    '/tmp/out.webm',
    crf=31,
    audio_bitrate='128k'
)
```

convert_to_hevc(*source*, *output*, *preset=None*, *crf=None*, *video_bitrate=None*, *audio_bitrate=None*, ***kwargs*)

Converts a video file to MP4 format using H.265 (HEVC) for video and AAC for audio.

Uses a single-pass constant quality encode process for HEVC. See https://trac.ffmpeg.org/wiki/Encode/H.265 for tips on HEVC encoding with ffmpeg, and for description of the CRF and preset parameters.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **preset** (*str*) – encoder preset (quality / encoding speed tradeoff) - optional
- **crf** (*str*) – constant rate factor (determines target quality) - optional
- **audio_bitrate** (*int or str*) – target audio bitrate - optional
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments - optional

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

If *scale* is set, either width or height may be `-1` to use the optimal size for preserving aspect ratio, or `0` to keep the original size.

Bitrates should be specified as integers or as strings in 'NUMk' or 'NUMm' format.

Example:

```
from avtk.backends.ffmpeg.shortcuts import convert_to_hevc

convert_to_hevc(
    'test-media/video/sintel.mkv',
    '/tmp/out.mp4'
)
```

convert_to_aac(*source*, *output*, *bit_rate=None*, ***kwargs*)

Converts a media file to audio MP4 using AAC codec.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **bit_rate** (*int or str*) – target audio bitrate - optional
- **channels** (*int*) – number of channels to downmix to - optional

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

Bit rate should be specified as integers or as strings in 'NUMk' or 'NUMm' format.

Example:

```
from avtk.backends.ffmpeg.shortcuts import convert_to_aac

convert_to_aac(
    'test-media/audio/stereo.mp3',
    '/tmp/out.aac',
    bit_rate='96k'
)
```

convert_to_opus(*source*, *output*, *bit_rate=None*, ***kwargs*)

Converts a media file to Opus-encoded Ogg file.

Parameters

- **source** (*str*) – input file path or stream URL
- **output** (*str*) – output file path
- **bit_rate** (*int or str*) – target audio bitrate - optional
- **channels** (*int*) – number of channels to downmix to - optional

Raises ***NoMediaError*** – if source doesn't exist or is of unknown format

Bit rate should be specified as integers or as strings in 'NUMk' or 'NUMm' format.

Example:

```
from avtk.backends.ffmpeg.shortcuts import convert_to_opus

convert_to_opus(
    'test-media/audio/stereo.mp3',
    '/tmp/out.ogg',
    bit_rate='96k'
)
```

1.3.2 Converting media files

This module contains wrappers for the `ffmpeg` command-line utility. The aim is to expose most-often used functionality in a concise and Pythonic way, while still allowing passing extra options unsupported by AVTK to the underlying ffmpeg invocation.

Synopsis:

```
FFmpeg(inputs, outputs).run()
```

Inputs and outputs can either be lists (in case of multiple inputs or outputs), or a single item (in case of single input or output).

Input can be either a string (input file path or stream URL), or an instance of *Input*. Output can either be a string (output file path) or an instance of *Output* specifying the format, codecs to use and other options.

Examples:

```
# Convert MP4 to MKV with default codecs and options
FFmpeg('input.mp4', 'output.mkv').run()

# Convert 1 minute of input video starting at 5min mark into 1080p HEVC video
# with 160kbit AAC audio, stripping any subtitles
FFmpeg(
    Input('input.mkv', seek=300, duration=60),
    Output('output.mp4',
        Video('libx265', scale=(-1, 1080), preset='veryfast', crf=20),
        Audio('aac', bit_rate='160k'),
        NoSubtitles
    )
).run()

# Split video into separate video and audio files
FFmpeg(['input.mp4', [
    Output('video.mp4', streams=[H264(), NoAudio]),
    Output('audio.m4a', streams=[AAC(), NoVideo])
]).run()
```

Stream, format, input and output definitions all take an optional *extra* parameter - a list of strings that are passed directly to ffmpeg in appropriate places. This allows you to use all functionality of ffmpeg without explicit support in AVTK.

class Duration(*val*)

Bases: `object`

Helper class for parsing duration and time values - don't use it directly.

Supported duration types: *timedelta*, *Decimal*, *float*, *int*

Examples:

```
Duration(timedelta(seconds=3.14))
Duration('5.25')
Duration(60.5)
Duration(3600)
```

class Stream(*encoder*)

Bases: `object`

Output stream definition - base class

Don't use this class directly. Instead use *Video*, *Audio* or *Subtitle* subclass, or one of the convenience helper classes for specific codecs.

class Video(*encoder*, *scale=None*, *bit_rate=None*, *frames=None*, *extra=None*)
Bases: *avtk.backends.ffmpeg.convert.Stream*

Output video stream definition

Parameters

- **encoder** (*str*) – encoder to use
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **bit_rate** (*int or str*) – target video bitrate - optional
- **frames** (*int*) – number of frames to output - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

Use *avtk.backends.ffmpeg.cap.get_available_encoders()* to get information on supported encoders. You can also use *None* to disable the video stream (see *NoVideo*), or `copy` to copy the video stream without re-encoding (see *CopyVideo*).

If *scale* is set, either width or height may be `-1` to use the optimal size for preserving aspect ratio, or `0` to keep the original size.

Bitrates should be specified as integers or as strings in 'NUMk' or 'NUMm' format.

Examples:

```
# Encode to 480p H.264 at 2mbps, keeping aspect ratio
Video('libx264', scale=(-1:480), bit_rate='2m')

# Convert only one frame to PNG
Video('png', frames=1)

# Passthrough (copy video stream from input to output)
Video('copy')
# or
CopyVideo

# Disable (ignore) video streams
NoVideo
```

class H264(*preset=None*, *crf=None*, ***kwargs*)
Bases: *avtk.backends.ffmpeg.convert.Video*

Video stream using H.264 (AVC) codec and libx264 encoder

Parameters

- **preset** (*str*) – encoder preset (quality / encoding speed tradeoff) - optional
- **crf** (*str*) – constant rate factor (determines target quality) - optional
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **bit_rate** (*int or str*) – target video bitrate - optional
- **frames** (*int*) – number of frames to output - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

See https://trac.ffmpeg.org/wiki/Encode/H.264 for tips on H.264 encoding with ffmpeg, and for description of CRF and preset parameters.

Examples:

```
# Encode to high quality 1080p, taking as much time as needed
H264(preset='veryslow', crf=18)
```

class H265(*preset=None*, *crf=None*, ***kwargs*)

Bases: *avtk.backends.ffmpeg.convert.Video*

Video stream using H.265 (HEVC) codec and libx265 encoder

Parameters

- **preset** (*str*) – encoder preset (quality / encoding speed tradeoff) - optional
- **crf** (*str*) – constant rate factor (determines target quality) - optional
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **bit_rate** (*int or str*) – target video bitrate - optional
- **frames** (*int*) – number of frames to output - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

Uses a single-pass constant quality encode process for HEVC. See https://trac.ffmpeg.org/wiki/Encode/H.265 for tips on HEVC encoding with ffmpeg, and for description of the CRF and preset parameters.

Example:

```
# Encode to high quality, taking as much time as needed
H265(preset='veryslow', crf=20)
```

class VP9(*crf=None*, ***kwargs*)

Bases: *avtk.backends.ffmpeg.convert.Video*

Video stream using VP9 codec and libvpx-vp9 encoder

Parameters

- **crf** (*str*) – constant rate factor (determines target quality) - optional, default is 31
- **scale** (*tuple*) – resize output to specified size (width, height) - optional
- **frames** (*int*) – number of frames to output - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

Uses a single-pass constant quality encode process for VP9. See https://trac.ffmpeg.org/wiki/Encode/VP9 for tips on VP9 encoding with ffmpeg, and for description of the CRF parameter.

Example:

```
# Encode using default parameters (CRF of 31)
VP9()
```

class AV1(*crf=None*, ***kwargs*)

Bases: *avtk.backends.ffmpeg.convert.Video*

Video stream using AV1 codec and libaom-av1 encoder

Based on AV1 wiki guide: https://trac.ffmpeg.org/wiki/Encode/AV1

> **Warning:** Uses an extremely slow and unoptimized reference AV1 encoder implementation. You probably don't want to use this.

class Audio(*encoder*, *channels=None*, *bit_rate=None*, *extra=None*)

Bases: *avtk.backends.ffmpeg.convert.Stream*

Output audio stream definition

Parameters

- **encoder** (*str*) – encoder to use
- **channels** (*int*) – number of channels to downmix to - optional
- **bit_rate** (*int or str*) – target audio bitrate - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

Use *avtk.backends.ffmpeg.cap.get_available_encoders()* to get information on supported encoders. You can also use *None* to disable the audio streams (see *NoAudio*), or `copy` to copy the stream without re-encoding (see *CopyAudio*).

Bitrates should be specified as integers or as strings in 'NUMk' format.

Examples:

```
# Encode to AAC at 256kbps, while downmixing to stereo
Audio('aac', channels=2, bit_rate='256k')

# Passthrough (copy audio stream directly to output)
Audio('copy')
# or
CopyAudio

# Disable (ignore) audio stream
NoAudio
```

class AAC(***kwargs*)

Bases: *avtk.backends.ffmpeg.convert.Audio*

Audio stream using AAC

Parameters

- **channels** (*int*) – number of channels to downmix to - optional
- **bit_rate** (*int or str*) – target audio bitrate - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

class Opus(***kwargs*)

Bases: *avtk.backends.ffmpeg.convert.Audio*

Audio stream using Opus codec and libopus encoder

Parameters

- **channels** (*int*) – number of channels to downmix to - optional
- **bit_rate** (*int or str*) – target audio bitrate - optional

- **extra** (*list(str)*) – additional ffmpeg command line arguments for the stream - optional

class Subtitle(*encoder*)
Bases: *avtk.backends.ffmpeg.convert.Stream*

Output subtitle stream definition

Use *avtk.backends.ffmpeg.cap.get_available_encoders()* to get information on supported encoders. You can also use *None* to disable the subtitle streams (see *NoSubtitles*), or `copy` to copy the stream without converting (see *CopySubtitles*).

NoVideo = <avtk.backends.ffmpeg.convert.Video object>
Disable video

CopyVideo = <avtk.backends.ffmpeg.convert.Video object>
Dopy video without re-encoding

NoAudio = <avtk.backends.ffmpeg.convert.Audio object>
Disable audio

CopyAudio = <avtk.backends.ffmpeg.convert.Audio object>
Copy audio without re-encoding

NoSubtitles = <avtk.backends.ffmpeg.convert.Subtitle object>
Disable subtitles

CopySubtitles = <avtk.backends.ffmpeg.convert.Subtitle object>
Copy subtitles

class Input(*source*, *seek=None*, *duration=None*, *extra=None*)
Bases: `object`

Define input source

Parameters

- **source** (*str*) – input file path or stream URL
- **seek** (see *Duration*) – seek in source before processing - optional
- **duration** (see *Duration*) – how much of source to process - optional
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the input - optional

Raises ***NoMediaError*** – if source doesn't exist

Source can either be a local file, capture device or network stream supported by the underlying `ffmpeg` tool.

Examples:

```
# Use local file, seek to 2min mark and process 5 minutes
Input('test-media/video/sintel.mkv', seek=120, duration=300)

# Process one minute of an internet radio stream
Input('https://example.radio.fm/stream.aac', duration=60)
```

class Format(*name*, *extra=None*)
Bases: `object`

Output container format definition

Parameters

- **name** (*str*) – format name

- **extra** (*list(str)*) – additional ffmpeg command line arguments for the input - optional

Use *avtk.backends.ffmpeg.cap.get_available_formats()* to get information on supported formats.

class MP4 (*faststart=False*)
Bases: *avtk.backends.ffmpeg.convert.Format*

MP4 output format

Parameters **faststart** (*bool*) – place the *moov atom* metadata at the beginning - optional, default *False*

This option is required if you want to support playback of incomplete MP4 file (that is, start playback before the entire file is downloaded).

class WebM
Bases: *avtk.backends.ffmpeg.convert.Format*

WebM output format

class Matroska
Bases: *avtk.backends.ffmpeg.convert.Format*

Matroska (MKV) output format

class Ogg
Bases: *avtk.backends.ffmpeg.convert.Format*

Ogg output format

class Output (*target*, *streams=None*, *fmt=None*, *extra=None*)
Bases: object

Defines an output to the transcoding process.

Parameters

- **target** (*str*) – output file path
- **streams** (*Stream* or *None*) – output stream definitions - optional
- **fmt** – output format - optional
- **fmt** – *Format*, *str* or *None*
- **extra** (*list(str)*) – additional ffmpeg command line arguments for the output - optional

If streams are not specified, all input streams will be transcoded using default codecs for the specified format.

If format is not specified, it is guessed from the output file name. If specified, it can either be a format name or an instance of *Format*.

class FFmpeg (*inputs*, *outputs*)
Bases: object

Convert audio/video

Parameters

- **inputs** (*Input*, *str*, list(*Input*) or *list(str)*) – one or more inputs
- **outputs** (*Output*, *str*, list(*Output*) or *list(str)*) – one or more outputs

Input(s) and output(s) can either be specified as strings representing input and output files respectively, or as *Input* and *Output* objects with all the configuration exposed by those classes.

If only one input is used, it can be specified directly. If multiple inputs are used, specify a list of inputs. Likewise for outputs.

Examples:

```
# Convert input to output
FFmpeg('input.mp4', 'output.mkv').run()

# Combine audio and video
FFmpeg(['video.mp4', 'audio.m4a'], 'output.mkv').run()

# Split audio and video
FFmpeg(['input.mp4', [
    Output('video.mp4', streams=[H264(), NoAudio]),
    Output('audio.m4a', streams=[AAC(), NoVideo])
]).run()
```

get_args()

Builds ffmpeg command line arguments

Returns ffmpeg command line arguments for the specified process

Return type list of strings

Example:

```
>>> FFmpeg(['input.mp4', [
...     Output('video.mp4', streams=[H264(), NoAudio]),
...     Output('audio.m4a', streams=[AAC(), NoVideo])
... ]).get_args()
[
    '-i', 'input.mp4',
    '-c:v', 'libx264', '-an', 'video.mp4',
    '-c:a', 'aac', '-vn', 'audio.m4a'
]
```

run(*text=True*)

Runs the conversion process

Uses *get_args()* to build the command line and runs it using *avtk.backends.ffmpeg.run.ffmpeg()*.

Parameters **text** (*bool*) – whether to return the output as text - optional, default true

Returns output (stdout) from `ffmpeg` invocation

Return type *str* if *text=True* (default), *bytes* if *text=False*

1.3.3 Inspecting media files

A wrapper around ffprobe that gathers and makes available information about the inspected media file or stream in a Pythonic way.

class MediaInfo(*source*)

Bases: `object`

Represents all available information about the inspected media file or stream.

Attributes

- raw (*dict*) - raw `ffprobe` JSON output
- format (*Format*) - container (format) information
- streams (list(*Stream*)) - audio, video and subtitle streams information

The *avtk.backends.ffmpeg.shortcuts.inspect()* function is a simple wrapper around the MediaInfo constructor.

Example:

```
>>> info = MediaInfo('test-media/video/sintel.mkv')
```

audio_streams
List of audio streams in the media object

video_streams
List of video streams in the media object

subtitle_streams
List of subtitle streams in the media objects

has_audio
Whether there is at least one audio stream present

has_video
Whether there is at least one video stream present

has_subtitles
Whether there is at least one subtitles stream present

class Codec(*raw*)
Bases: `object`

Information about the codec used in a stream.

Attributes

- name (*str*) - codec (short) name
- type (*str*) - codec type, one of *TYPE_AUDIO*, *TYPE_VIDEO*, *TYPE_SUBTITLE* or *TYPE_DATA*
- description (*str*) - codec description (long name)
- profile (*str* or *None*) - profile used, if applicable

TYPE_AUDIO = 'audio'
Audio codec

TYPE_VIDEO = 'video'
Video codec

TYPE_SUBTITLE = 'subtitle'
Subtitles codec

TYPE_DATA = 'data'
Raw (unknown) data

class Stream(*raw*)
Bases: `object`

Information about a stream in a media file.

This is a base class handling information common to most stream types. Streams returned in *MediaInfo* object will be one of the subclasses holding more information.

Subclasses

- *VideoStream* - for video streams
- *AudioStream* - for audio streams
- *SubtitleStream* - for subtitle streams
- *DataStream* - raw or unrecognized data

Attributes

- codec (*Codec*) - codec used in this stream
- index (*int*) - 0-based index of this stream in the container
- time_base (*Fraction*) - unit of time for timestamp calculations
- nb_frames (*int* or *None*) - total number of frames in the stream
- start_time (*timedelta* or *None*) - start timestamp of the stream
- duration (*timedelta* or *None*) - duration of the stream
- duration_ts (*int* or *None*) - duration in *time_base* units
- bit_rate (*int* or *None*) - bit rate of the stream
- tags (*dict*) - stream tags, if any

class VideoStream(*raw*)

Bases: *avtk.backends.ffmpeg.probe.Stream*

Holds video-specific information about a stream.

Attributes

- width (*int*) - display width, in pixels
- height (*int*) - display height, in pixels
- display_aspect_ratio (*str*) - aspect ratio, in 'W:H' format
- pix_fmt (*str*) - image pixel format name
- has_b_frames (*int*) - how many frames might need reordering for B-frame decoding or 0 if B-frames are not used

For a list of all known pixel formats and more information about them, run `ffmpeg -pix_fmts`.

class AudioStream(*raw*)

Bases: *avtk.backends.ffmpeg.probe.Stream*

Holds video-specific information about a stream.

Attributes

- channels (*int*) - number of channels
- channel_layout (*str*) - channel layout
- sample_fmt (*str*) - sample format
- sample_rate (*int*) - audio sample rate in Hz

For a list of all known channel names and standard layouts, run `ffmpeg -layouts`. To get a list of all known sample formats, run `ffmpeg -sample_fmts`.

class SubtitleStream(*raw*)
Bases: *avtk.backends.ffmpeg.probe.Stream*

Holds information about a subtitle stream.

Attributes

- language (*str*) - language

class DataStream(*raw*)
Bases: *avtk.backends.ffmpeg.probe.Stream*

Holds information about unknown or raw data stream.

Attributes

- raw (*dict*) - raw data parsed from `ffprobe`

class Format(*raw*)
Bases: `object`

Information about the container format.

Attributes

- name (*str*) - format name, or multiple comma-separated format names (aliases)
- names (*list(str)*) - list of format names
- description (*str*) - format description (long name)
- start_time (*timedelta* or *None*) - start timestamp of the entire media object
- duration (*timedelta* or *None*) - duration of the entire media object
- size (*int* or *None*) - size in bytes, if known
- bit_rate (*int* or *None*) - bit rate of the entire media object
- tags (*dict*) - stream tags, if any

1.3.4 Codecs, encoders and formats support in FFmpeg

The *avtk.backends.ffmpeg.cap* module can be used to discover versions of the available `ffmpeg` and `ffprobe` command-line tools, and list codecs, encoders and formats they support.

Example usage:

```
>>> from avtk.backends.ffmpeg import cap

>>> cap.get_ffmpeg_version()
'4.1.2'
>>> cap.get_ffprobe_version()
'4.1.2'
>>> 'vp9' in cap.get_available_codecs()
True
>>> 'libx264' in cap.get_availble_encoders()
True
>>> 'webm' in cap.get_available_formats()
True
```

class Codec(*name*, *type*, *can_encode*, *can_decode*, *can_all_i*, *lossy*, *lossless*, *desc*)
Describes a codec known to FFmpeg

This doesn't mean it can be encoded or decoded. For encoding support, look at *Encoder*.

Attributes:

- `name` - Codec (short) name
- `description` - Codec description (long name)
- `type` - Codec type, one of *TYPE_AUDIO*, *TYPE_VIDEO* or *TYPE_SUBTITLE*
- `can_encode` - Whether ffmpeg can encode to this codec
- `can_decode` - Whether ffmpeg can decode this codec
- `can_all_i` - Whether the codec is all-intra (can have only I frames)
- `lossy` - Whether the codec supports lossy encoding
- `lossless` - Whether the codec supports lossless encoding

TYPE_AUDIO = 'audio'
Audio codec

TYPE_VIDEO = 'video'
Video codec

TYPE_SUBTITLE = 'subtitle'
Subtitle codec

class Encoder(*name*, *type*, *desc*)
Describes an available encoder

Attributes:

- `name` - Encoder name
- `description` - Encoder description
- `type` - Encoder type, one of *Codec.TYPE_AUDIO*, *Codec.TYPE_VIDEO* or *Codec.TYPE_SUBTITLE*

class Format(*name*, *can_mux*, *can_demux*, *desc*)
Describes a format known to FFmpeg

Attributes:

- `name` - Format name
- `description` - Format description (long name)
- `can_mux` - Whether ffmpeg can mux (pack) to this format
- `can_demux` - Whether ffmpeg can demux (unpack) this format

get_available_codecs()
Discovers and returns a list of available codecs.

Result is cached between runs so the results will only be discovered once.

Returns A list of available codecs

Return type list(*Codec*)

Raises **ValueError** – if `ffmpeg` utility cannot be found

get_available_formats()
Discovers and returns a list of available formats

Result is cached between runs so the results will only be discovered once.

Returns A list of available formats

Return type list(*Format*)

Raises **ValueError** – if `ffmpeg` utility cannot be found

get_available_encoders()
Discovers and returns a list of available encoders

Result is cached between runs so the results will only be discovered once.

Returns A list of available encoders

Return type list(*Encoder*)

Raises **ValueError** – if `ffmpeg` utility cannot be found

get_ffmpeg_version()
Returns version of installed ffmpeg tool

Returns FFmpeg version in 'x.y.z' format

Return type str

Raises **ValueError** – if `ffmpeg` utility cannot be found

get_ffprobe_version()
Returns version of installed ffprobe tool

Returns FFprobe version in 'x.y.z' format

Return type str

Raises **ValueError** – if `ffprobe` utility cannot be found

1.3.5 FFmpeg module index

Submodules

ffmpeg.shortcuts module

See *avtk.backends.ffmpeg.shortcuts*.

ffmpeg.probe module

See *avtk.backends.ffmpeg.probe*.

ffmpeg.convert module

See *avtk.backends.ffmpeg.probe*.

ffmpeg.exceptions module

exception NoMediaError
Bases: `Exception`

ffmpeg.run module

ffprobe(*args*, *parse_json=True*)

ffmpeg(*args*, *quick=False*, *text=True*)

1.4 Commercial support

AVTK is a free and open source software and comes with no support guarantee. I will try to address bug reports and requests promptly, but there are only so much hours in a day, and my available time for open source is limited.

Commercial support for AVTK, audio/video consulting and related software development is also available - please visit https://senkorasic.com/ for details.

CHAPTER

TWO

LICENCE

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

N

O

R

S

T

V

W