vardautomation

Release 1.2.4

lchunjo

Dec 12, 2022

CONTENTS

CHAPTER

ONE

API REFERENCE

Collection of classes and helper functions to automate encoding

1.1 Configuration

class vardautomation.config.**FileInfo**(*path, /, trims_or_dfs=None, *, idx=None, preset=[Preset(idx=vapoursynth.core.lsmas.LWLibavSource, a_src=VPath('{work_filename:s}_track_{track_number:s}.wav'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.wav'), a_enc_cut=None, chapter=None, preset_type=<PresetType.VIDEO: 10>), Preset(idx=vapoursynth.core.lsmas.LWLibavSource, a_src=VPath('{work_filename:s}_track_{track_number:s}.w64'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.w64'), a_enc_cut=None, chapter=None, preset_type=<PresetType.VIDEO: 10>)], workdir=VPath('/home/docs/checkouts/readthedocs.org/user_builds/vardautomation/ch*

FileInfo object. This is the first thing you should initialise.

Helper which allows to store the data related to your file to be encoded

Parameters

- **path** (PathLike[str] | str) – Path to your source file.
- **trims_or_dfs** (Union[List[Union[Trim, DuplicateFrame]], Trim, None]) – Adjust the clip length by trimming or duplicating frames. Python slicing. Defaults to None
- **idx** (Optional[Callable[[str], VideoNode]]) – Indexer used to index the video track, defaults to None
- **preset** (Union[Preset, Sequence[Preset]]) – Preset used to fill idx, a_src, a_src_cut, a_enc_cut and chapter attributes, defaults to PresetGeneric
- **workdir** (PathLike[str] | str) – Work directory. Default to the current directorie where the script is launched.

path: *VPath*

Path of the video file

path_without_ext: *VPath*

Path of the video file without the extension

work_filename: str

Work directory filename

idx: Optional[Callable[[str], vapoursynth.VideoNode]]

Vapoursynth Indexer

preset: List[Preset]

Preset(s) used

name: str

Name of the script

workdir: *VPath*

Work directory

a_src: Optional[*VPath*]

Audio source path

a_src_cut: Optional[*VPath*]

Audio source trimmed/cut path

a_enc_cut: Optional[*VPath*]

Audio source encoded (and trimmed) path

clip: vapoursynth.VideoNode

VideoNode object loaded by the indexer

clip_cut: vapoursynth.VideoNode

Clip trimmed

name_clip_output: *VPath*

Clip output path name

name_file_final: *VPath*

Final file output path

set_name_clip_output_ext(*extension*, /)

Set the extension of *FileInfo.name_clip_output*

Parameters

extension (str) – Extension in string format, eg. ".265"

Return type

None

property chapter: Optional[*VPath*]

Chapter file path

Setter

Set the chapter path

Return type

Optional[*VPath*]

property trims_or_dfs: Optional[Union[List[Union[vardefunc.types.Trim, vardefunc.types.DuplicateFrame]], vardefunc.types.Trim]]

Trims or DuplicateFrame objects of the current FileInfo

Setter

Set trims or duplicate frames

Return type

Union[List[Union[Trim, DuplicateFrame]], Trim, None]

property media_info: MediaInfo

Get the MediaInfo of the video file loaded

Return type
MediaInfo

property num_prop: bool

If the frame number is added to props

Setter
Add a prop FileInfoFrameNumber to the frame properties of *FileInfo.clip* and *FileInfo.clip_cut*

Return type
bool

class vardautomation.config.**FileInfo2**(*path, /, trims_or_dfs=None, *, idx=None, preset=[Preset(idx=vapoursynth.core.lsmas.LWLibavSource, a_src=VPath('{work_filename:s}_track_{track_number:s}.wav'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.wav'), a_enc_cut=None, chapter=None, preset_type=<PresetType.VIDEO: 10>), Preset(idx=vapoursynth.core.lsmas.LWLibavSource, a_src=VPath('{work_filename:s}_track_{track_number:s}.w64'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.w64'), a_enc_cut=None, chapter=None, preset_type=<PresetType.VIDEO: 10>)], workdir=VPath('/home/docs/checkouts/readthedocs.org/user_builds/vardautomation/*

Second version of FileInfo adding audio support

Helper which allows to store the data related to your file to be encoded

Parameters

- **path** (PathLike[str] | str) – Path to your source file.
- **trims_or_dfs** (Union[List[Union[Trim, DuplicateFrame]], Trim, None]) – Adjust the clip length by trimming or duplicating frames. Python slicing. Defaults to None
- **idx** (Optional[Callable[[str], VideoNode]]) – Indexer used to index the video track, defaults to None
- **preset** (Union[Preset, Sequence[Preset]]) – Preset used to fill idx, a_src, a_src_cut, a_enc_cut and chapter attributes, defaults to PresetGeneric
- **workdir** (PathLike[str] | str) – Work directory. Default to the current directorie where the script is launched.

audios: List[vapoursynth.AudioNode]

List of AudioNode indexed by BestAudioSource in the file

audios_cut: List[vapoursynth.AudioNode]

List of AudioNode cut with the specified trims

property trims_or_dfs: Optional[Union[List[Union[vardefunc.types.Trim, vardefunc.types.DuplicateFrame]], vardefunc.types.Trim]]

Trims or DuplicateFrame objects of the current FileInfo

Setter
Set trims or duplicate frames

Return type
`Union[List[Union[Trim, DuplicateFrame]], Trim, None]`

property audio: vapoursynth.AudioNode

Return the first AudioNode track of the file.

Return type
`AudioNode`

Returns
AudioNode

property audio_cut: vapoursynth.AudioNode

Return the first trimmed AudioNode track of the file.

Return type
`AudioNode`

Returns
AudioNode

write_a_src(*index*, *offset=-1*)

Using *audio_async_render* write the AudioNodes of the file as a WAV file to *a_src* path

Return type
`None`

Parameters

- **index** (*int*) –
- **offset** (*int*) –

write_a_src_cut(*index*, *offset=-1*)

Using *audio_async_render* write the AudioNodes of the file as a WAV file to *a_src_cut* path

Return type
`None`

Parameters

- **index** (*int*) –
- **offset** (*int*) –

class vardautomation.config.**BluraySho**w(*episodes*, *global_trims=None*, ***, *idx=None*, *preset=[PresetBD, PresetBDWAV64]*, *lang=UNDEFINED*, *fps=Fraction(24000, 1001)*)

Helper class for batching shows

Parameters

- **episodes** (`Dict[VPath, List[VPath]]`) – A dictionnary of episodes. Keys are the path of each bdmv folder. Values are the episodes inside the current bdmv folder key.
- **global_trims** (`Union[List[Union[Trim, DuplicateFrame]], Trim, None]`) – Adjust the clips length by trimming or duplicating frames. Python slicing. Defaults to None
- **idx** (`Optional[Callable[[str], VideoNode]]`) – Indexer used to index the video track, defaults to None
- **preset** (`Union[Preset, Sequence[Preset]]`) – Preset used to fill idx, a_src, a_src_cut, a_enc_cut and chapter attributes, defaults to `PresetGeneric`

- **lang** (*Lang*) – Chapters language, defaults to UNDEFINED
- **fps** (*Fraction*) –

register_ncops(**path*)

Add NCOP paths to the class

Return type
None

Parameters
path (VPath) –

register_nceds(**path*)

Add NCED paths to the class

Return type
None

Parameters
path (VPath) –

ncops(*file_info_t*)

Get all the NCOPs

Return type
List[TypeVar(_FileInfoType, bound= *FileInfo*)]

Returns
List of FileInfo

Parameters
file_info_t (*Type[_FileInfoType]*) –

ncop(*num*, */*, *file_info_t*, ***, *start_from=1*)

Get a specified NCOP

Parameters

- **num** (int) – Numero of the NCOP
- **start_from** (int) – Indexing starting value, defaults to 1
- **file_info_t** (*Type[_FileInfoType]*) –

Return type
TypeVar(_FileInfoType, bound= *FileInfo*)

Returns
FileInfo object

nceds(*file_info_t*)

Get all the NCEDs

Return type
List[TypeVar(_FileInfoType, bound= *FileInfo*)]

Returns
List of FileInfo

Parameters
file_info_t (*Type[_FileInfoType]*) –

nced(*num*, /, *file_info_t*, *, *start_from=1*)

Get a specified NCED

Parameters

- **num** (int) – Numero of the NCED
- **start_from** (int) – Indexing starting value, defaults to 1
- **file_info_t** (*Type[_FileInfoType]*) –

Return type
TypeVar(_FileInfoType, bound= *FileInfo*)

Returns
FileInfo object

episodes(*file_info_t*)

Get all the episodes

Return type
List[TypeVar(_FileInfoType, bound= *FileInfo*)]

Returns
List of FileInfo

Parameters
file_info_t (*Type[_FileInfoType]*) –

episode(*num*, /, *file_info_t*, *, *start_from=1*)

Get a specified episode

Parameters

- **num** (int) – Numero of the episode
- **start_from** (int) – Indexing starting value, defaults to 1
- **file_info_t** (*Type[_FileInfoType]*) –

Return type
TypeVar(_FileInfoType, bound= *FileInfo*)

Returns
FileInfo object

1.1.1 Presets

vardautomation.config.**PresetBD = Preset(idx=vapoursynth.core.lsmas.LWLibavSource, a_src=VPath('{work_filename:s}_track_{track_number:s}.wav'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.wav'), a_enc_cut=None, chapter=None, preset_type=<PresetType.VIDEO: 10>)**

Preset for BD encode. The indexer is core.lsmas.LWLibavSource and audio sources are .wav

vardautomation.config.**PresetWEB = Preset(idx=vapoursynth.core.ffms2.Source, a_src=None, a_src_cut=None, a_enc_cut=VPath('.'), chapter=None, preset_type=<PresetType.VIDEO: 10>)**

Preset for WEB encode. The indexer is core.ffms2.Source and a_enc_cut is blocked.

vardautomation.config.**PresetAAC = Preset(idx=None, a_src=VPath('{work_filename:s}_track_{track_number:s}.aac'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.aac'), a_enc_cut=VPath('{work_filename:s}_cut_enc_track_{track_number:s}.m4a'), chapter=None, preset_type=<PresetType.AUDIO: 20>)**

Preset for AAC encode.

vardautomation.config.**PresetOpus = Preset(idx=None, a_src=VPath('{work_filename:s}_track_{track_number:s}.opus'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.opus'), a_enc_cut=VPath('{work_filename:s}_cut_enc_track_{track_number:s}.opus'), chapter=None, preset_type=<PresetType.AUDIO: 20>)**

Preset for Opus encode.

vardautomation.config.**PresetEAC3 = Preset(idx=None, a_src=VPath('{work_filename:s}_track_{track_number:s}.eac3'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.eac3'), a_enc_cut=VPath('{work_filename:s}_cut_enc_track_{track_number:s}.eac3'), chapter=None, preset_type=<PresetType.AUDIO: 20>)**

Preset for EAC3 encode.

vardautomation.config.**PresetFLAC = Preset(idx=None, a_src=VPath('{work_filename:s}_track_{track_number:s}.flac'), a_src_cut=VPath('{work_filename:s}_cut_track_{track_number:s}.flac'), a_enc_cut=VPath('{work_filename:s}_cut_enc_track_{track_number:s}.flac'), chapter=None, preset_type=<PresetType.AUDIO: 20>)**

Preset for FLAC encode.

vardautomation.config.**PresetChapOGM = Preset(idx=None, a_src=None, a_src_cut=None, a_enc_cut=None, chapter=VPath('chapters/{name:s}.txt'), preset_type=<PresetType.CHAPTER: 30>)**

Preset for OGM based chapters.

vardautomation.config.**PresetChapXML = Preset(idx=None, a_src=None, a_src_cut=None, a_enc_cut=None, chapter=VPath('chapters/{name:s}.xml'), preset_type=<PresetType.CHAPTER: 30>)**

Preset for XML based chapters.

1.2 Tools

class vardautomation.tooling.abstract.**Tool**(*binary*, *settings*, *, *check_binary=True*)

Bases: ABC

Abstract Tool interface.

Most of the tools inherit from it.

```
# This
>>> cat settings
-o {clip_output:s} - --y4m --preset slower --crf 51

# is equivalent to this:
settings: List[str] = ['-o', '{clip_output:s}', '-', '--y4m', '--preset', 'slower',
```

(continues on next page)

(continued from previous page)

```
↪'--crf', '51']

# and is equivalent to this:
settings: Dict[str, Any] = {
    '-o': '{clip_output:s}',
    '-': None,
    '--y4m': None,
    '--preset': 'slower',
    '--crf': 51
}
```

Parameters

- **binary** (PathLike[str] | str) – Path to your binary file.
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – Path to your settings file or list of string or a dict containing your settings Special variable names can be specified and are replaced at runtime. Supported variable names are defined in *set_variable()* docstring.
- **check_binary** (bool) – Check binary's availability.

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

abstract run()

Tooling chain

Return type
Optional[NoReturn]

abstract set_variable()

Set variables in the settings

Return type
Dict[str, Any]

class vardautomation.tooling.base.**BasicTool**(*binary*, *settings*, */*, *file=None*, *check_binary=True*)

Bases: *Tool*

BasicTool interface.

Helper allowing the use of CLI programs for basic tasks like video or audio track extraction.

Parameters

- **binary** (PathLike[str] | str) – See Tool.binary
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – See Tool.settings
- **file** (Optional[*FileInfo*]) – Not used in BasicTool implementation, defaults to None
- **check_binary** (bool) – Check binary's availability.

file: Optional[*FileInfo*]

FileInfo object.

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

1.2.1 Video encoders

class vardautomation.tooling.video.**VideoEncoder**(*binary*, *settings*)

Bases: *Tool*

General VideoEncoder interface

```
# This
>>> cat settings
-o {clip_output:s} - --y4m --preset slower --crf 51

# is equivalent to this:
settings: List[str] = ['-o', '{clip_output:s}', '-', '--y4m', '--preset', 'slower',
↪'--crf', '51']

# and is equivalent to this:
settings: Dict[str, Any] = {
    '-o': '{clip_output:s}',
    '-': None,
    '--y4m': None,
    '--preset': 'slower',
    '--crf': 51
}
```

Parameters

- **binary** (PathLike[str] | str) – Path to your binary file.
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – Path to your settings file or list of string or a dict containing your settings Special variable names can be specified and are replaced at runtime. Supported variable names are defined in *set_variable()* docstring.

file: *FileInfo*

FileInfo object

clip: **vapoursynth.VideoNode**

Your filtered VideoNode clip

y4m: **bool = True**

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

progress_update: **Optional[Callable[[int, int], None]] = None**

Progress update function to be used in *vapoursynth.VideoNode.output*

prefetch: **int = 0**

Max number of concurrent rendered frames

backlog: **int = -1**

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

run_enc(*clip: vs.VideoNode*, *file:* FileInfo) → None

run_enc(*clip: vs.VideoNode*, *file: None*) → None

Run encoding toolchain

Parameters

- **clip** (VideoNode) – Clip to be encoded
- **file** (Optional[*FileInfo*]) – FileInfo object

Return type

None

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type

NoReturn

set_variable()

Set variables in the settings

Replaces {clip_output:s} by self.file.name_clip_output

Replaces {filename:s} by self.file.name

Return type

Dict[str, Any]

binary: *VPath*

Binary path

params: **List[str]**

Settings normalised and parsed

class vardautomation.tooling.video.**VideoLanEncoder**(*settings*, /, *zones=None*, *override_params=None*, *progress_update=progress_update_func*)

Bases: SupportManualVFR, SupportResume, SupportQpfile, HasZone, HasOverrideParams, *VideoEncoder*, ABC

Abstract VideoEncoder interface for VideoLan based encoders such as x265 and x264.

```
# This
>>> cat settings
-o {clip_output:s} - --y4m --preset slower --crf 51

# is equivalent to this:
settings: List[str] = ['-o', '{clip_output:s}', '-', '--y4m', '--preset', 'slower',
↪'--crf', '51']

# and is equivalent to this:
settings: Dict[str, Any] = {
    '-o': '{clip_output:s}',
    '-': None,
    '--y4m': None,
    '--preset': 'slower',
    '--crf': 51
}
```

Parameters

- **binary** – Path to your binary file.
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – Path to your settings file or list of string or a dict containing your settings Special variable names can be specified and are replaced at runtime. Supported variable names are defined in *set_variable()* docstring.
- **zones** (*Optional[Dict[Tuple[int, int], Dict[str, Any]]]*) –
- **override_params** (*Optional[Dict[str, Any]]*) –
- **progress_update** (*Optional[Callable[[int, int], None]]*) –

```
override_params: Dict[str, Any] = {'--crf': 10, '--preset': 'ultrafast'}
```

Parameters

- **override_params** (Optional[Dict[str, Any]]) – Parameters to be overrided in settings
- **settings** (*Union[PathLike[str], str, List[str], Dict[str, Any]]*) –
- **zones** (*Optional[Dict[Tuple[int, int], Dict[str, Any]]]*) –
- **progress_update** (*Optional[Callable[[int, int], None]]*) –

backlog: int = -1

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

property params_asdict: Dict[str, Any]

Get *params* as a dictionnary

Return type
Dict[str, Any]

prefetch: int = 0

Max number of concurrent rendered frames

progress_update: Optional[UpdateFunc] = None

Progress update function to be used in *vapoursynth.VideoNode.output*

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type
NoReturn

run_enc(*clip*, *file*, *, *qpfile_clip=None*, *qpfile_func=<function make_qpfile>*)

Run encoding toolchain

Parameters

- **clip** (*Union[vapoursynth.VideoNode, Sequence[vapoursynth.VideoNode]]*) – Clip to be encoded
- **file** (vardautomation.config.FileInfo | *None*) – FileInfo object
- **qpfile_clip** (*vs.VideoNode* | *None*) – Clip to be used to generate the Qpfile
- **qpfile_func** (*Callable[[vapoursynth.VideoNode, os.PathLike[str] | str], Qpfile]*) – Function to be used to generate the Qpfile

Return type
None

y4m: bool = True

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

file: *FileInfo*

FileInfo object

clip: vs.VideoNode

Your filtered VideoNode clip

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

resumable: bool = False

Enable resumable encodes

set_variable()

Set variables in the settings

Replaces `{clip_output:s}` with `self.file.name_clip_output`

Replaces `{filename:s}` with `self.file.name`

Replaces {frames:d} with self.clip.num_frames

Replaces {fps_num:d} with self.clip.fps.numerator

Replaces {fps_den:d} with self.clip.fps.denominator

Replaces {bits:d} with Properties.get_depth(self.clip)

Replaces {min_keyint:d} with round(self.clip.fps)

Replaces {keyint:d} with round(self.clip.fps) * 10

Return type
Dict[str, Any]

class vardautomation.tooling.video.**X265**(*settings*, /, *zones=None*, *override_params=None*, *progress_update=progress_update_func*)

Bases: *VideoLanEncoder*

Video encoder using x265 for HEVC

```
# This
>>> cat settings
-o {clip_output:s} - --y4m --preset slower --crf 51

# is equivalent to this:
settings: List[str] = ['-o', '{clip_output:s}', '-', '--y4m', '--preset', 'slower',
↪'--crf', '51']

# and is equivalent to this:
settings: Dict[str, Any] = {
    '-o': '{clip_output:s}',
    '-': None,
    '--y4m': None,
    '--preset': 'slower',
    '--crf': 51
}
```

Parameters

- **binary** – Path to your binary file.
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – Path to your settings file or list of string or a dict containing your settings Special variable names can be specified and are replaced at runtime. Supported variable names are defined in *set_variable()* docstring.
- **zones** (*Optional[Dict[Tuple[int, int], Dict[str, Any]]]*) –
- **override_params** (*Optional[Dict[str, Any]]*) –
- **progress_update** (*Optional[Callable[[int, int], None]]*) –

```
override_params: Dict[str, Any] = {'--crf': 10, '--preset': 'ultrafast'}
```

Parameters

- **override_params** (Optional[Dict[str, Any]]) – Parameters to be overrided in settings

- **settings** (*Union[PathLike[str], str, List[str], Dict[str, Any]]*) –
- **zones** (*Optional[Dict[Tuple[int, int], Dict[str, Any]]]*) –
- **progress_update** (*Optional[Callable[[int, int], None]]*) –

backlog: int = -1

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

property params_asdict: Dict[str, Any]

Get *params* as a dictionnary

Return type
Dict[str, Any]

prefetch: int = 0

Max number of concurrent rendered frames

progress_update: Optional[UpdateFunc] = None

Progress update function to be used in *vapoursynth.VideoNode.output*

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type
NoReturn

run_enc(*clip*, *file*, **, *qpfile_clip=None*, *qpfile_func=<function make_qpfile>*)

Run encoding toolchain

Parameters

- **clip** (*Union[vapoursynth.VideoNode, Sequence[vapoursynth.VideoNode]]*) – Clip to be encoded
- **file** (vardautomation.config.FileInfo | *None*) – FileInfo object
- **qpfile_clip** (*vs.VideoNode | None*) – Clip to be used to generate the Qpfile
- **qpfile_func** (*Callable[[vapoursynth.VideoNode, os.PathLike[str] | str], Qpfile]*) – Function to be used to generate the Qpfile

Return type
None

y4m: bool = True

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

file: *FileInfo*

FileInfo object

clip: vs.VideoNode

Your filtered VideoNode clip

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

resumable: bool = False

Enable resumable encodes

set_variable()

Set variables in the settings

Replaces `{clip_output:s}` with `self.file.name_clip_output`

Replaces `{filename:s}` with `self.file.name`

Replaces `{frames:d}` with `self.clip.num_frames`

Replaces `{fps_num:d}` with `self.clip.fps.numerator`

Replaces `{fps_den:d}` with `self.clip.fps.denominator`

Replaces `{bits:d}` with `Properties.get_depth(self.clip)`

Replaces `{min_keyint:d}` with `round(self.clip.fps)`

Replaces `{keyint:d}` with `round(self.clip.fps) * 10`

Replaces `{min_luma:d}` and `{max_luma:d}` with `Properties.get_colour_range(self.params, self.clip)`

Replaces `{matrix:d}` with `Properties.get_prop(self.clip.get_frame(0), '_Matrix', int)`

Replaces `{primaries:d}` with `Properties.get_prop(self.clip.get_frame(0), '_Primaries', int)`

Replaces `{transfer:d}` with `Properties.get_prop(self.clip.get_frame(0), '_Transfer', int)`

Return type
`Dict[str, Any]`

class vardautomation.tooling.video.**X264**(*settings, /, zones=None, override_params=None, progress_update=progress_update_func*)

Bases: *VideoLanEncoder*

Video encoder using x264 for AVC

```
# This
>>> cat settings
-o {clip_output:s} - --y4m --preset slower --crf 51

# is equivalent to this:
settings: List[str] = ['-o', '{clip_output:s}', '-', '--y4m', '--preset', 'slower',
↪'--crf', '51']

# and is equivalent to this:
settings: Dict[str, Any] = {
    '-o': '{clip_output:s}',
    '-': None,
    '--y4m': None,
    '--preset': 'slower',
    '--crf': 51
}
```

Parameters

- **binary** – Path to your binary file.
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – Path to your settings file or list of string or a dict containing your settings Special variable names can be specified and are replaced at runtime. Supported variable names are defined in *set_variable()* docstring.
- **zones** (*Optional[Dict[Tuple[int, int], Dict[str, Any]]]*) –
- **override_params** (*Optional[Dict[str, Any]]*) –
- **progress_update** (*Optional[Callable[[int, int], None]]*) –

```
override_params: Dict[str, Any] = {'--crf': 10, '--preset': 'ultrafast'}
```

Parameters

- **override_params** (Optional[Dict[str, Any]]) – Parameters to be overrided in settings
- **settings** (*Union[PathLike[str], str, List[str], Dict[str, Any]]*) –
- **zones** (*Optional[Dict[Tuple[int, int], Dict[str, Any]]]*) –
- **progress_update** (*Optional[Callable[[int, int], None]]*) –

backlog: int = -1

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

property params_asdict: Dict[str, Any]

Get *params* as a dictionnary

Return type
Dict[str, Any]

prefetch: int = 0

Max number of concurrent rendered frames

progress_update: Optional[UpdateFunc] = None

Progress update function to be used in *vapoursynth.VideoNode.output*

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type
NoReturn

run_enc(*clip*, *file*, *, *qpfile_clip=None*, *qpfile_func=<function make_qpfile>*)

Run encoding toolchain

Parameters

- **clip** (*Union[vapoursynth.VideoNode, Sequence[vapoursynth.VideoNode]]*) – Clip to be encoded
- **file** (vardautomation.config.FileInfo | *None*) – FileInfo object

- **qpfile_clip** (*vs.VideoNode | None*) – Clip to be used to generate the Qpfile
- **qpfile_func** (*Callable[[vapoursynth.VideoNode, os.PathLike[str] | str], Qpfile]*) – Function to be used to generate the Qpfile

Return type
None

y4m: bool = True

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

file: *FileInfo*

FileInfo object

clip: vs.VideoNode

Your filtered VideoNode clip

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

resumable: bool = False

Enable resumable encodes

set_variable()

Set variables in the settings

Replaces `{clip_output:s}` with `self.file.name_clip_output`

Replaces `{filename:s}` with `self.file.name`

Replaces `{frames:d}` with `self.clip.num_frames`

Replaces `{fps_num:d}` with `self.clip.fps.numerator`

Replaces `{fps_den:d}` with `self.clip.fps.denominator`

Replaces `{bits:d}` with `Properties.get_depth(self.clip)`

Replaces `{min_keyint:d}` with `round(self.clip.fps)`

Replaces `{keyint:d}` with `round(self.clip.fps) * 10`

Replaces `{csp:s}` with `Properties.get_csp(self.clip)`

Replaces `{matrix:s}` with `Properties.get_prop(self.clip.get_frame(0), '_Matrix', int)`

Replaces `{primaries:s}` with `Properties.get_prop(self.clip.get_frame(0), '_Primaries', int)`

Replaces `{transfer:s}` with `Properties.get_prop(self.clip.get_frame(0), '_Transfer', int)`

Return type
`Dict[str, Any]`

class vardautomation.tooling.video.**LosslessEncoder**(*binary*, *settings*)

Bases: *VideoEncoder*

Video encoder for lossless encoding

```
# This
>>> cat settings
-o {clip_output:s} - --y4m --preset slower --crf 51

# is equivalent to this:
settings: List[str] = ['-o', '{clip_output:s}', '-', '--y4m', '--preset', 'slower',
↪'--crf', '51']

# and is equivalent to this:
settings: Dict[str, Any] = {
    '-o': '{clip_output:s}',
    '-': None,
    '--y4m': None,
    '--preset': 'slower',
    '--crf': 51
}
```

Parameters

- **binary** (PathLike[str] | str) – Path to your binary file.
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – Path to your settings file or list of string or a dict containing your settings Special variable names can be specified and are replaced at runtime. Supported variable names are defined in *set_variable()* docstring.

suffix_name: str = '_lossless'

Suffix name for the lossless output

set_variable()

Set variables in the settings

Replaces {clip_output_lossless:s} by self.file.name_clip_output.append_stem(self.suffix_name)

Replaces {bits:s} by Properties.get_depth(self.clip)

Return type
Dict[str, Any]

backlog: int = -1

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

prefetch: int = 0

Max number of concurrent rendered frames

progress_update: Optional[UpdateFunc] = None

Progress update function to be used in *vapoursynth.VideoNode.output*

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type
NoReturn

run_enc(*clip*, *file*)

Run encoding toolchain

Parameters

- **clip** (VideoNode) – Clip to be encoded
- **file** (Optional[*FileInfo*]) – FileInfo object

Return type
None

y4m: bool = True

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

file: *FileInfo*

FileInfo object

clip: vs.VideoNode

Your filtered VideoNode clip

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.video.**NVEncCLossless**(*, *hevc=True*)

Bases: *LosslessEncoder*

Built-in NvencC encoder.

Use NvencC to output a lossless encode in HEVC

Parameters
hevc (bool) – If True use HEVC codec for output. Keep in mind that 10 bit support depends on your NVEnc, driver and CUDA version

suffix_name: str = '_lossless.mkv'

Suffix name for the lossless output

progress_update: Optional[UpdateFunc] = None

Progress update function to be used in *vapoursynth.VideoNode.output*

backlog: int = -1

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

prefetch: int = 0

Max number of concurrent rendered frames

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type
NoReturn

run_enc(*clip*, *file*)

Run encoding toolchain

Parameters

- **clip** (VideoNode) – Clip to be encoded
- **file** (Optional[*FileInfo*]) – FileInfo object

Return type
None

set_variable()

Set variables in the settings

Replaces {clip_output_lossless:s} by self.file.name_clip_output.append_stem(self.suffix_name)

Replaces {bits:s} by Properties.get_depth(self.clip)

Return type
Dict[str, Any]

y4m: bool = True

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

file: *FileInfo*

FileInfo object

clip: vs.VideoNode

Your filtered VideoNode clip

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.video.**FFV1**(*, *threads=0*)

Bases: *LosslessEncoder*

Built-in FFV1 encoder.

Use FFmpeg to output a lossless encode in FFV1

Parameters
threads (int) – Number of threads to be used, defaults to 0 (auto selection)

suffix_name: str = '_lossless.mkv'

Suffix name for the lossless output

progress_update: Optional[UpdateFunc] = None

Progress update function to be used in *vapoursynth.VideoNode.output*

backlog: int = -1

Defines how many unconsumed frames (including those that did not finish rendering yet) vapoursynth buffers at most before it stops rendering additional frames.

This argument is there to limit the memory this function uses storing frames.

prefetch: int = 0

Max number of concurrent rendered frames

run()

Shouldn't be used in VideoEncoder object. Use *run_enc()* instead

Return type
NoReturn

run_enc(*clip*, *file*)

Run encoding toolchain

Parameters

- **clip** (VideoNode) – Clip to be encoded
- **file** (Optional[*FileInfo*]) – FileInfo object

Return type
None

set_variable()

Set variables in the settings

Replaces {clip_output_lossless:s} by self.file.name_clip_output.append_stem(self.suffix_name)

Replaces {bits:s} by Properties.get_depth(self.clip)

Return type
Dict[str, Any]

y4m: bool = True

YUV4MPEG2 headers

More informations http://www.vapoursynth.com/doc/pythonreference.html#VideoNode.output

file: *FileInfo*

FileInfo object

clip: vs.VideoNode

Your filtered VideoNode clip

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

1.2.2 Audio extracters

class vardautomation.tooling.audio.**AudioExtracter**(*binary*, *settings*, */*, *file*)

Bases: *BasicTool*

Audio base extracter interface for audio extration

Parameters

- **binary** (PathLike[str] | str) – See Tool.binary

- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – See Tool.settings
- **file** (*FileInfo*) – FileInfo object, needed

file: *FileInfo*

FileInfo object

track_in: Sequence[int]

Track number(s) of the input file

track_out: Sequence[int]

Track number(s) of the output file

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**MKVAudioExtracter**(*file*, */*, ***, *track_in=-1*, *track_out=-1*, *mkvextract_args=None*)

AudioExtracter using MKVExtract

Parameters

- **file** (*FileInfo*) – FileInfo object, needed
- **track_in** (Union[int, Sequence[int]]) – Input track(s) number
- **track_out** (Union[int, Sequence[int]]) – Output track(s) number
- **mkvextract_args** (Optional[List[str]]) – https://mkvtoolnix.download/doc/mkvextract.html, defaults to None

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track_in: Sequence[int]

Track number(s) of the input file

track_out: Sequence[int]

Track number(s) of the output file

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**Eac3toAudioExtracter**(*file*, */*, ***, *track_in=-1*, *track_out=-1*, *eac3to_args=None*)

AudioExtracter using Eac3to

Parameters

- **file** (*FileInfo*) – FileInfo object, needed
- **track_in** (Union[int, Sequence[int]]) – Input track(s) number
- **track_out** (Union[int, Sequence[int]]) – Output track(s) number
- **eac3to_args** (Optional[List[str]]) – https://en.wikibooks.org/wiki/Eac3to/How_to_Use, eg. ['-log=nul']

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track_in: Sequence[int]

Track number(s) of the input file

track_out: Sequence[int]

Track number(s) of the output file

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**FFmpegAudioExtracter**(*file*, /, *, *track_in=-1*, *track_out=-1*)

AudioExtracter using Ffmpeg

Parameters

- **file** (*FileInfo*) – FileInfo object, needed
- **track_in** (Union[int, Sequence[int]]) – Input track(s) number
- **track_out** (Union[int, Sequence[int]]) – Output track(s) number

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track_in: **Sequence[int]**

Track number(s) of the input file

track_out: **Sequence[int]**

Track number(s) of the output file

binary: *VPath*

Binary path

params: **List[str]**

Settings normalised and parsed

1.2.3 Audio cutters

class vardautomation.tooling.audio.**AudioCutter**(*file*, /, *, *track*, ***kwargs*)

Bases: ABC

Abstract interface implementing audio trimming

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **kwargs** (Any) – Additionnal arguments

file: *FileInfo*

FileInfo object

track: **int**

Track number

kwargs: Dict[str, Any]

Additionnal arguments

abstract run()

Trimming toolchain

Return type
None

abstract classmethod generate_silence(*s*, *output*, *num_ch=2*, *sample_rate=48000*, *bitdepth=16*)

Generate silence if supported by the current interface

Parameters

- **s** (float) – Seconds
- **output** (PathLike[str] | str) – Output file path
- **num_ch** (int) – Number of channels, defaults to 2
- **sample_rate** (int) – Sample rate in Hz, defaults to 48000
- **bitdepth** (int) – Bit depth, defaults to 16

Return type
Optional[NoReturn]

class vardautomation.tooling.audio.**ScipyCutter**(*file*, */*, ***, *track*, ***kwargs*)

Bases: *AudioCutter*

Audio cutter using scipy.io.wavfile module

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **kwargs** (Any) – Additionnal arguments

file: *FileInfo*

FileInfo object

track: int

Track number

kwargs: Dict[str, Any]

Additionnal arguments

run()

Trimming toolchain

Return type
None

classmethod scipytrim(*src*, *output*, */*, *trims*, *ref_clip*, ***, *combine=True*)

Simple trimming function that follows VapourSynth/Python slicing syntax. End frame is NOT inclusive.

Parameters

- **src** (PathLike[str] | str) – Input file
- **output** (PathLike[str] | str) – Output file

- **trims** (Union[Trim, DuplicateFrame, List[Trim], List[Union[Trim, DuplicateFrame]]]) – Either a list of 2-tuples, one tuple of 2 ints, a DuplicateFrame object or a list of of 2-tuples and/or DuplicateFrame object.
- **ref_clip** (VideoNode) – Vapoursynth clip used to determine framerate AND the number of frames.
- **combine** (bool) – Keep all performed trims in the same file, defaults to True

Return type
None

classmethod generate_silence(*s*, *output*, *num_ch=2*, *sample_rate=48000*, *bitdepth=16*)

Generate silence if supported by the current interface

Parameters

- **s** (float) – Seconds
- **output** (PathLike[str] | str) – Output file path
- **num_ch** (int) – Number of channels, defaults to 2
- **sample_rate** (int) – Sample rate in Hz, defaults to 48000
- **bitdepth** (int) – Bit depth, defaults to 16

Return type
None

class vardautomation.tooling.audio.**EztrimCutter**(*file*, */*, ***, *track*, ***kwargs*)

Bases: *AudioCutter*

AudioCutter using acsuite.eztrim().

It fallbacks on *EztrimCutter.ezpztrim()* if some DuplicateFrame objects are detected in the FileInfo object specified.

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **kwargs** (Any) – Additionnal arguments

file: *FileInfo*

FileInfo object

track: int

Track number

kwargs: Dict[str, Any]

Additionnal arguments

run()

Trimming toolchain

Return type
None

classmethod ezpztrim(*src*, *output*, */*, *trims*, *ref_clip*, ***, *combine=True*, *cleanup=True*)

Simple trimming function that follows VapourSynth/Python slicing syntax. End frame is NOT inclusive.

Parameters

- **src** (PathLike[str] | str) – Input file
- **output** (PathLike[str] | str) – Output file
- **trims** (Union[Trim, DuplicateFrame, List[Trim], List[Union[Trim, DuplicateFrame]]]) – Either a list of 2-tuples, one tuple of 2 ints, a DuplicateFrame object or a list of of 2-tuples and/or DuplicateFrame object.
- **ref_clip** (VideoNode) – Vapoursynth clip used to determine framerate AND the number of frames.
- **combine** (bool) – Keep all performed trims in the same file, defaults to True
- **cleanup** (bool) – Delete temporary files, defaults to True

Return type
None

classmethod generate_silence(*cls*, *s*, *output*, *num_ch=2*, *sample_rate=48000*, *bitdepth=16*)

Generate silence if supported by the current interface

Parameters

- **s** (float) – Seconds
- **output** (PathLike[str] | str) – Output file path
- **num_ch** (int) – Number of channels, defaults to 2
- **sample_rate** (int) – Sample rate in Hz, defaults to 48000
- **bitdepth** (int) – Bit depth, defaults to 16

Return type
None

class vardautomation.tooling.audio.**SoxCutter**(*file*, */*, ***, *track*, ***kwargs*)

Bases: *AudioCutter*

Audio cutter using Sox

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **kwargs** (Any) – Additionnal arguments

file: *FileInfo*

FileInfo object

track: int

Track number

kwargs: Dict[str, Any]

Additionnal arguments

run()

Trimming toolchain

Return type
None

classmethod soxtrim(*src*, *output*, */*, *trims*, *ref_clip*, ***, *combine=True*, *cleanup=True*)

Simple trimming function that follows VapourSynth/Python slicing syntax. End frame is NOT inclusive.

Parameters

- **src** (PathLike[str] | str) – Input file
- **output** (PathLike[str] | str) – Output file
- **trims** (Union[Trim, DuplicateFrame, List[Trim], List[Union[Trim, DuplicateFrame]]]) – Either a list of 2-tuples, one tuple of 2 ints, a DuplicateFrame object or a list of of 2-tuples and/or DuplicateFrame object.
- **ref_clip** (VideoNode) – Vapoursynth clip used to determine framerate AND the number of frames.
- **combine** (bool) – Keep all performed trims in the same file, defaults to True
- **cleanup** (bool) – Delete temporary files, defaults to True

Return type
None

classmethod generate_silence(*s*, *output*, *num_ch=2*, *sample_rate=48000*, *bitdepth=16*)

Generate silence if supported by the current interface

Parameters

- **s** (float) – Seconds
- **output** (PathLike[str] | str) – Output file path
- **num_ch** (int) – Number of channels, defaults to 2
- **sample_rate** (int) – Sample rate in Hz, defaults to 48000
- **bitdepth** (int) – Bit depth, defaults to 16

Return type
None

class vardautomation.tooling.audio.**PassthroughCutter**(*file*, */*, ***, *track*, ***kwargs*)

Bases: *AudioCutter*

Special AudioCutter that will copy *vardautomation.config.FileInfo.a_src* to *vardautomation.config.FileInfo.a_src_cut*

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **kwargs** (Any) – Additionnal arguments

file: *FileInfo*

FileInfo object

track: int

Track number

kwargs: Dict[str, Any]

Additionnal arguments

run()

Trimming toolchain

Return type
None

classmethod generate_silence(*cls*, *s*, *output*, *num_ch=2*, *sample_rate=48000*, *bitdepth=16*)

You can't generate silence from this class

Return type
NoReturn

Parameters

- **s** (*float*) –
- **output** (*os.PathLike[str] | str*) –
- **num_ch** (*int*) –
- **sample_rate** (*int*) –
- **bitdepth** (*int*) –

1.2.4 Audio encoders

class vardautomation.tooling.audio.**AudioEncoder**(*binary*, *settings*, */*, *file*, ***, *track=-1*, *xml_tag=None*, *check_binary=True*)

Bases: *BasicTool*

BasicTool interface helper for audio encoding

Parameters

- **binary** (PathLike[str] | str) – See Tool.binary
- **settings** (Union[PathLike[str], str, List[str], Dict[str, Any]]) – See Tool.settings
- **file** (*FileInfo*) – FileInfo object, needed.
- **track** (int) – Track number
- **xml_tag** (Union[PathLike[str], str, None]) – See *AudioEncoder.xml_tag*, defaults to None

  If specified, will write a file containing the encoder info to be passed to the muxer.
- **check_binary** (bool) – Check binary's availability.

file: *FileInfo*

FileInfo object

track: **int**

Track number

xml_tag: **Optional[Union[PathLike[str], str]]**

XML tags suitable for mkvmerge

Curently only write the encoder name

More info here: https://mkvtoolnix.download/doc/mkvmerge.html#mkvmerge.tags

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**PassthroughAudioEncoder**(*file*, *, *track=-1*)

Bases: *AudioEncoder*

Special AudioEncoder that will copy FileInfo.a_src_cut to FileInfo.a_enc_cut

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **xml_tag** – See *AudioEncoder.xml_tag*, defaults to None

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track: int

Track number

xml_tag: Optional[AnyPath]

XML tags suitable for mkvmerge

Curently only write the encoder name

More info here: https://mkvtoolnix.download/doc/mkvmerge.html#mkvmerge.tags

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**QAACEncoder**(*file*, ***, *track=-1*, *mode=BitrateMode.TVBR*, *bitrate=127*, *xml_tag=None*, *qaac_args=None*)

Bases: *AudioEncoder*

AudioEncoder using QAAC, an open-source wrapper for Core Audio's AAC and ALAC encoder.

These following options are automatically added:

- `--no-delay --no-optimize --threading`

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **mode** (Literal[<BitrateMode.ABR>, <BitrateMode.CBR>, <BitrateMode.CVBR>, <BitrateMode.TVBR>]) – Bitrate mode. QAAC supports ABR, CBR, CVBR and TVBR, defaults to BitrateMode.TVBR
- **bitrate** (int) – Matches the bitrate for ABR, CBR and CVBR in kbit/s and quality Q for TVBR, defaults to 127
- **xml_tag** (Union[PathLike[str], str, None]) – See *AudioEncoder.xml_tag*, defaults to None
- **qaac_args** (Optional[List[str]]) – Additional options, see https://github.com/nu774/qaac/wiki/Command-Line-Options, defaults to None

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track: int

Track number

xml_tag: Optional[AnyPath]

XML tags suitable for mkvmerge

Curently only write the encoder name

More info here: https://mkvtoolnix.download/doc/mkvmerge.html#mkvmerge.tags

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**OpusEncoder**(*file*, ***, *track=-1*, *mode=BitrateMode.VBR*, *bitrate=160*, *xml_tag=None*, *use_ffmpeg=True*, *opus_args=None*)

Bases: *AudioEncoder*

AudioEncoder using Opus, open, royalty-free, highly versatile audio codec.

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **mode** (Literal[<BitrateMode.VBR>, <BitrateMode.CVBR>, <BitrateMode.CBR>]) – Bitrate mode. libopus supports VBR, CVBR, and HARD_CBR (aka CBR), defaults to BitrateMode.VBR
- **bitrate** (int) – Target bitrate in kbit/s, defaults to 160
- **xml_tag** (Union[PathLike[str], str, None]) – See *AudioEncoder.xml_tag*, defaults to None
- **use_ffmpeg** (bool) – Use opusenc if False, defaults to True
- **opus_args** (Optional[List[str]]) – Additionnal arguments, defaults to None

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track: int

Track number

xml_tag: Optional[AnyPath]

XML tags suitable for mkvmerge

Curently only write the encoder name

More info here: https://mkvtoolnix.download/doc/mkvmerge.html#mkvmerge.tags

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**FDKAACEncoder**(*file*, ***, *track=-1*, *mode=BitrateMode.CBR*, *bitrate=256*, *cutoff=20000*, *xml_tag=None*, *use_ffmpeg=True*, *fdk_args=None*)

Bases: *AudioEncoder*

AudioEncoder using fdkaac.

The libfdk-aac library is based on the Fraunhofer FDK AAC code from the Android project

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **mode** (Literal[<BitrateMode.CBR>, <BitrateMode.VBR>]) – Bitrate mode, fdkaac supports CBR and VBR, defaults to BitrateMode.CBR
- **bitrate** (int) – Matches the bitrate for CBR in kbit/s and quality Q for VBR, defaults to 256
- **cutoff** (int) – Set cutoff frequency. If not specified (or explicitly set to 0) it will use a value automatically computed by the library. Correspond to frequency bandwidth in Hz in fdkaac library, defaults to 20000
- **xml_tag** (Union[PathLike[str], str, None]) – See *AudioEncoder.xml_tag*, defaults to None
- **use_ffmpeg** (bool) – Use fdkaac if False, defaults to True
- **fdk_args** (Optional[List[str]]) – Additional options see https://www.ffmpeg.org/ffmpeg-codecs.html#Options-11 or https://github.com/nu774/fdkaac/blob/master/README, defaults to None

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track: int

Track number

xml_tag: Optional[AnyPath]

XML tags suitable for mkvmerge

Curently only write the encoder name

More info here: https://mkvtoolnix.download/doc/mkvmerge.html#mkvmerge.tags

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**FlacEncoder**(*file*, ***, *track=-1*, *xml_tag=None*, *level=FlacCompressionLevel.VARDOU*, *use_ffmpeg=True*, *flac_args=None*)

Bases: *AudioEncoder*

AudioEncoder using FLAC, Free Lossless Audio Codec

Parameters

- **file** (*FileInfo*) – FileInfo object
- **track** (int) – Track number
- **xml_tag** (Union[PathLike[str], str, None]) – See *AudioEncoder.xml_tag*, defaults to None
- **level** (*FlacCompressionLevel*) – See *FlacCompressionLevel* for all levels available, defaults to FlacCompressionLevel.VARDOU
- **use_ffmpeg** (bool) – Will use flac if false, defaults to True
- **flac_args** (Optional[List[str]]) – Additionnal arguments, defaults to None

run()

Tooling chain

Return type
None

set_variable()

Set variables in the settings

No variable are replaced there.

Return type
Dict[str, Any]

file: *FileInfo*

FileInfo object

track: int

Track number

xml_tag: Optional[AnyPath]

XML tags suitable for mkvmerge

Curently only write the encoder name

More info here: https://mkvtoolnix.download/doc/mkvmerge.html#mkvmerge.tags

binary: *VPath*

Binary path

params: List[str]

Settings normalised and parsed

class vardautomation.tooling.audio.**BitrateMode**(*value*)

Common bitrate mode enumeration

ABR = 1

Average BitRate

CBR = 2

Constant BitRate

VBR = 3

Variable BitRate

CVBR = 4

Constrained Variable BitRate

TVBR = 5

True Variable BitRate

HARD_CBR = 2

Hard Constant BitRate

class vardautomation.tooling.audio.**FlacCompressionLevel**(*value*)

Flac compression level.

Keep in mind that the max FLAC can handle is 8 and ffmpeg 12

ZERO = 0

ffmpeg: compression_level 0

flac: –compression-level-0

ONE = 1

ffmpeg: compression_level 1

flac: –compression-level-1

TWO = 2

ffmpeg: compression_level 2

flac: –compression-level-2

THREE = 3

ffmpeg: compression_level 3

flac: –compression-level-3

FOUR = 4

ffmpeg: compression_level 4

flac: –compression-level-4

FIVE = 5

ffmpeg: compression_level 5

flac: –compression-level-5

This is the default for both ffmpeg and flac encoders

SIX = 6

ffmpeg: compression_level 6

flac: –compression-level-6

SEVEN = 7

ffmpeg: compression_level 7

flac: –compression-level-7

EIGHT = 8

ffmpeg: compression_level 8

flac: –compression-level-8

NINE = 9

ffmpeg: compression_level 9

TEN = 10

ffmpeg: compression_level 10

ELEVEN = 11

ffmpeg: compression_level 11

TWELVE = 12

ffmpeg: compression_level 12

FAST = 0

Fastest compression. Currently synonymous with 0

ffmpeg: compression_level 0

flac: –compression-level-0

BEST = 8

Highest compression. Currently synonymous with -8

ffmpeg: compression_level 0

flac: –compression-level-0

VARDOU = 99

My custom ffmpeg command:

```
['-compression_level', '12', '-lpc_type', 'cholesky', '-lpc_passes', '3', '-
↪exact_rice_parameters', '1']
```

1.2.5 Muxing

class `vardautomation.tooling.mux.`**`MatroskaFile`**(*output*, *tracks=None*, */*, **global_opts*)

Bases: `_AbstractMatroskaFile`

Matroska file interface

Register a new matroska file to be merged/splitted/appended

Parameters

- **output** (`PathLike[str] | str`) – Output path
- **tracks** (`Union[PathLike[str], str, Track, Iterable[PathLike[str] | str | Track], None]`) – A path or an iterable of path/Track
- **global_opts** (`str`) – Global options

global_opts: Tuple[str, ...]

Global options and other options that affect the whole process

property command: List[str]

Get the mkvmerge command

Return type
List[str]

classmethod autotrack(*file*, *lang=None*)

Automatically get the tracks from a FileInfo object and make a MatroskaFile from it

Parameters

- **file** (*FileInfo*) – FileInfo object
- **lang** (*Optional[Lang]*) –

Return type
MatroskaFile

Returns
MatroskaFile object

static automux(*file*)

Call `MatroskaFile.autotrack` and mux it.

Parameters
file (*FileInfo*) – FileInfo object

Return type
None

property track_lang: Optional[Union[List[vardautomation.language.Lang | None], Lang]]

Lang(s) of the tracks of the current MatroskaFile object

Setter
Change the Lang of the tracks

Return type
Union[List[Optional[*Lang*]], *Lang*, None]

mux(*return_workfiles: Literal[True] = True*) → CleanupSet

mux(*return_workfiles: Literal[False]*) → None

Launch a merge command

Return type
Optional[CleanupSet]

Returns
Return worksfiles if True

split(*mode*, *param*)

Split function ruled by "mode"

Parameters

- **mode** (*SplitMode*) – Split mode
- **param** (str) – Full command after the mode

Return type
None

split_size(*size*)

Split the output file after a given size

Parameters
size (str) – d[k|m|g]

Return type
None

split_duration(*duration*)

Split the output file after a given duration

Parameters
duration (str) – HH:MM:SS.nnnnnnnnn|ds

Return type
None

split_timestamps(*timestamps*)

Split the output file after specific timestamps

Parameters
timestamps (Iterable[str]) – A[,B[,C…]]

Return type
None

split_parts(*parts*)

Keep specific parts by specifying timestamp ranges while discarding others

Parameters
parts (List[Tuple[Optional[str], Optional[str]]]) – start1-end1[,[+]start2-end2[,[+]start3-end3…]]

Return type
None

split_parts_frames(*parts*)

Keep specific parts by specifying frame/field number ranges while discarding others

Parameters
parts (List[Tuple[Optional[int], Optional[int]]]) – start1-end1[,[+]start2-end2[,[+]start3-end3…]]

Return type
None

split_frames(*frames*)

Split after specific frames/fields

Parameters
frames (Union[int, Iterable[int]]) – A[,B[,C…]]

Return type
None

split_chapters(*indices*)

Split before specific chapters

Parameters
indices (Union[Literal['all'], Iterable[int]]) – "all" or A[,B[,C...]]

Return type
None

append_to(*files*, *ids=None*)

Enable append mode

Parameters

- **files** (Iterable[PathLike[str] | str]) – Files to be appended
- **ids** (Optional[Iterable[Tuple[int, int, int, int]]]) – Controls to which track another track is appended.

Return type
None

add_timestamps(*path*, *id_=0*)

Add timestamps global command

Parameters

- **path** (PathLike[str] | str) – Timecode path
- **id** – [description], defaults to 0
- **id_** (*int*) –

Return type
None

class vardautomation.tooling.mux.**SplitMode**(*value*)

MKVMerge split modes

SIZE = 'size'

Split by size

DURATION = 'duration'

Split by duration

TIMESTAMPS = 'timestamps'

Split by timestamps

PARTS = 'parts'

Keep specific parts by specifying timestamp ranges while discarding others

PARTS_FRAMES = 'parts-frames'

Keep specific parts by specifying frame/field number ranges while discarding others

FRAMES = 'frames'

Split by frames

CHAPTERS = 'chapters'

Split by chapters

class vardautomation.tooling.mux.**Track**(*path*, **opts*)

Bases: _AbstractTrack

Standard Track interface for to be passed to mkvmerge

Register a new track

Parameters

- **path** (PathLike[str] | str) – Path to the file
- **opts** (str) – Additional options

path: *VPath*

VPath to the file

opts: **Tuple[str, ...]**

Additional options for this track

class vardautomation.tooling.mux.**MediaTrack**(*path*, *name=None*, *lang=UNDEFINED*, *tid=0*, */*, **opts*)

Bases: _LanguageTrack

Interface for medias track based to be passed to mkvmerge

Register a new track

Parameters

- **path** (PathLike[str] | str) – Path to the file
- **name** (Optional[str]) – Name of the track
- **lang** (*Lang* | str) – Language of the track
- **tid** (int) – Track ID
- **opts** (str) – Additional options

lang: *Lang*

Language of the track

name: **Optional[str]**

Name of the track

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

path: *VPath*

VPath to the file

opts: **Tuple[str, ...]**

Additional options for this track

class vardautomation.tooling.mux.**VideoTrack**(*path*, *name=None*, *lang=UNDEFINED*, *tid=0*, */*, **opts*)

Bases: *MediaTrack*

Register a new track

Parameters

- **path** (PathLike[str] | str) – Path to the file
- **name** (Optional[str]) – Name of the track
- **lang** (*Lang* | str) – Language of the track
- **tid** (int) – Track ID

- **opts** (str) – Additional options

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

name: **Optional[str]**

Name of the track

lang: *Lang*

Language of the track

path: *VPath*

VPath to the file

opts: **Tuple[str, ...]**

Additional options for this track

class vardautomation.tooling.mux.**AudioTrack**(*path*, *name=None*, *lang=UNDEFINED*, *tid=0*, */*, **opts*)

Bases: *MediaTrack*

Register a new track

Parameters

- **path** (PathLike[str] | str) – Path to the file
- **name** (Optional[str]) – Name of the track
- **lang** (*Lang* | str) – Language of the track
- **tid** (int) – Track ID
- **opts** (str) – Additional options

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

name: **Optional[str]**

Name of the track

lang: *Lang*

Language of the track

path: *VPath*

VPath to the file

opts: **Tuple[str, ...]**

Additional options for this track

class vardautomation.tooling.mux.**SubtitleTrack**(*path*, *name=None*, *lang=UNDEFINED*, *tid=0*, */*, **opts*)

Bases: *MediaTrack*

Register a new track

Parameters

- **path** (PathLike[str] | str) – Path to the file
- **name** (Optional[str]) – Name of the track
- **lang** (*Lang* | str) – Language of the track
- **tid** (int) – Track ID
- **opts** (str) – Additional options

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

name: Optional[str]

Name of the track

lang: *Lang*

Language of the track

path: *VPath*

VPath to the file

opts: Tuple[str, ...]

Additional options for this track

class vardautomation.tooling.mux.**ChaptersTrack**(*path*, *lang=UNDEFINED*, *charset=None*, */*, **opts*)

Bases: _LanguageTrack

Interface for chapters track based to be passed to mkvmerge

Register a new chapters track

Parameters

- **path** (PathLike[str] | str) – Path to the file
- **lang** (*Lang* | str) – Language of the track
- **charset** (Optional[str]) – Character set that is used for the conversion to UTF-8 for simple chapter files
- **opts** (*Tuple[str, ...]*) –

charset: Optional[str]

Character set that is used for the conversion to UTF-8 for simple chapter files.

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

lang: *Lang*

Language of the track

path: *VPath*

VPath to the file

opts: **Tuple[str, ...]**

Additional options for this track

1.2.6 Utility

class vardautomation.tooling.misc.**Qpfile**(*path*, *frames=None*)

Simple namedtuple for a qpfile

Create new instance of Qpfile(path, frames)

Parameters

- **path** (VPath) –
- **frames** (*Optional[List[int]]*) –

path: *VPath*

Qpfile path

frames: **Optional[List[int]]**

List of keyframes

vardautomation.tooling.misc.**make_qpfile**(*clip*, *path=None*, */*, *overwrite=True*, *mode=SCM.WWXD | SCM.SCXVID*)

Convenience function for making a qpfile

Parameters

- **clip** (VideoNode) – Source clip
- **path** (Union[PathLike[str], str, None]) – Path where the qpfile will be written. Default to the name of the script that run this function with the ".log" extension
- **overwrite** (bool) – If True, will overwrite the file
- **mode** (Union[int, SceneChangeMode]) – Scene change mode, defaults to SCM.WWXD_SCXVID_UNION

Return type
Qpfile

Returns
A Qpfile

vardautomation.tooling.misc.**get_vs_core**(*threads=None*, *max_cache_size=None*)

Get the VapourSynth singleton core. Optionaly, set the number of threads used and the maximum cache size

Parameters

- **threads** (Optional[Iterable[int]]) – An iteratable of thread numbers, defaults to None.
- **max_cache_size** (Optional[int]) – Set the upper framebuffer cache size after which memory is aggressively freed. The value is in megabytes, defaults to None.

Return type
Core

Returns
Vapoursynth Core.

1.3 Automation

class vardautomation.automation.**SelfRunner**(*clip*, *file*, /, *config*)

Self runner interface

Parameters

- **clip** (Union[VideoNode, Sequence[VideoNode]]) – Clip to be encoded
- **file** (*FileInfo*) – FileInfo object
- **config** (*RunnerConfig*) – Confif of the runner

clip: Union[vapoursynth.VideoNode, Sequence[vapoursynth.VideoNode]]

Clip to be encoded

file: *FileInfo*

FileInfo object

config: *RunnerConfig*

Config of the runner

work_files: CleanupSet

Intermediate working files:

The SelfRunner class will add everything it can in this set-like, meaning if you want to delete the files you can just do:

```
runner = SelfRunner(...)
runner.run()
runner.work_files.clear()
```

The runner will add these attributes to be deleted:

- FileInfo.name_clip_output
- FileInfo.a_src
- FileInfo.a_src_cut
- FileInfo.a_enc_cut
- FileInfo.chapter

So if you want to keep some of these files, this is possible:

```
runner.work_files.discard(self.file.name_clip_output)
runner.work_files.discard(self.file.chapter)
```

plp_function: Union[_PLPFunction, Callable[[*VPath*], vapoursynth.VideoNode]]

Post Lossless Processing function. Set this function if you need some adjustements on the lossless clip before running the encode. If set, it will be called with the lossless path as argument and must return a VideoNode.

run(*, *show_logo=True*)

Main tooling chain

Parameters

show_logo (bool) – Print vardoto logo.

Return type
None

inject_qpfile_params(*qpfile_clip*, *qpfile_func=make_qpfile*)

Parameters

- **qpfile_clip** (VideoNode) – Clip to be used to generate the Qpfile
- **qpfile_func** (Callable[[VideoNode, PathLike[str] | str], *Qpfile*]) – Function to be used to generate the Qpfile

Return type
None

rename_final_file(*name*)

Rename the file.name_file_final

Parameters
name (PathLike[str] | str) – New filename

Return type
None

upload_ftp(*ftp_name*, *destination*, *rclone_args=None*)

Upload the name_file_final to a given FTP using rclone

Parameters

- **ftp_name** (str) – FTP name
- **destination** (PathLike[str] | str) – Path destination
- **rclone_args** (Optional[List[str]]) – Additionnal options, defaults to None

Return type
None

class vardautomation.automation.**RunnerConfig**(*v_encoder*, *v_lossless_encoder=None*, *a_extracters=None*, *a_cutters=None*, *a_encoders=None*, *mkv=None*, *order=Order.VIDEO*, *clear_outputs=True*)

Config for the SelfRunner

Parameters

- **v_encoder** (VideoEncoder) –
- **v_lossless_encoder** (*Optional[LosslessEncoder]*) –
- **a_extracters** (*Optional[Union[AudioExtracter, Sequence[AudioExtracter]]]*) –
- **a_cutters** (*Optional[Union[AudioCutter, Sequence[AudioCutter]]]*) –
- **a_encoders** (*Optional[Union[AudioEncoder, Sequence[AudioEncoder]]]*) –
- **mkv** (vardautomation.tooling.mux.MatroskaFile | *None*) –
- **order** (Order) –
- **clear_outputs** (*bool*) –

class Order(*value*)

Simple enum for priority order

v_encoder: *VideoEncoder*

Video encoder

v_lossless_encoder: Optional[*LosslessEncoder*]

Lossless video encoder

a_extracters: Optional[Union[*AudioExtracter*, Sequence[*AudioExtracter*]]]

Audio extracter(s)

a_cutters: Optional[Union[*AudioCutter*, Sequence[*AudioCutter*]]]

Audio cutter(s)

a_encoders: Optional[Union[*AudioEncoder*, Sequence[*AudioEncoder*]]]

Audio encoder(s)

mkv: *vardautomation.tooling.mux.MatroskaFile* | None

Muxer

order: *Order*

Priority order

clear_outputs: bool

Clears all clips set for output in the current environment

class vardautomation.automation.**Patch**(*encoder*, *clip*, *file*, *ranges*, *output_filename=None*, *, *debug=False*)

Easy video patching interface

Parameters

- **encoder** (*VideoEncoder*) – VideoEncoder to be used
- **clip** (VideoNode) – Clip where the patch will pick the fixed ranges
- **file** (*FileInfo*) – FileInfo object. The file that will be fixed is the file defined in *vardautomation.config.FileInfo.name_file_final*
- **ranges** (Union[Range, List[Range]]) – Ranges of frames that need to be fixed
- **output_filename** (Optional[str]) – Optional filename. If not specified a suffix _new wil be added, defaults to None
- **debug** (bool) – Debug argument, defaults to False

encoder: *VideoEncoder*

VideoEncoder to be used

clip: vapoursynth.VideoNode

Clip where the patch will pick the fixed ranges

file: *FileInfo*

FileInfo object

The file that will be fixed is the file defined in *vardautomation.config.FileInfo.name_file_final*

ranges: List[Tuple[int, int]]

Normalised ranges

workdir: *VPath*

Work directory path

output_filename: *VPath*

Output filename path

debug: **bool**

Debug boolean

run()

Launch patch

Return type
None

do_cleanup()

Delete working directory folder

Return type
None

1.4 Chapters stuff

class vardautomation.chapterisation.**Chapter**(*name*, *start_frame*, *end_frame*, *lang=<vardautomation.language.Lang object>*)

Chapter object

Create new instance of Chapter(name, start_frame, end_frame, lang)

Parameters

- **name** (*str*) –
- **start_frame** (*int*) –
- **end_frame** (*Optional[int]*) –
- **lang** (Lang) –

name: **str**

Name of the chapter

start_frame: **int**

Start frame

end_frame: **Optional[int]**

Optional end frame

lang: *Lang*

Language of the chapter

class vardautomation.chapterisation.**Chapters**(*chapter_file*)

Bases: ABC

Abtract Chapters interface

Register a new Chapters object

Parameters
chapter_file (PathLike[str] | str) – Chapters file path

chapter_file: *VPath*

Chapters file path

abstract create(*chapters*, *fps*)

Create the current Chapters object by passing a list of Chapter

Parameters

- **chapters** (List[*Chapter*]) – List of Chapter
- **fps** (Fraction) – Framerate Per Second

Return type
Optional[NoReturn]

abstract set_names(*names*)

Change/set names of the current Chapters object

Parameters
names (Sequence[Optional[str]]) – List of optional names. A None value won't change the name of the current chapter list

Return type
Optional[NoReturn]

abstract shift_times(*frames*, *fps*)

Shift timestamps by given number of frames.

Parameters

- **frames** (int) – Corresponding number of frames to be shifted
- **fps** (Fraction) – Framerate Per Second

Return type
Optional[NoReturn]

abstract to_chapters(*fps*, *lang*)

Convert the Chapters object to a list of chapter

Parameters

- **fps** (Fraction) – Framerate Per Second
- **lang** (Optional[*Lang*]) – Language of the chapter. If specified it will override the current language of the Chapters object

Return type
List[*Chapter*]

copy(*destination*)

Copy source chapter to destination and change target of *Chapters.chapter_file* to the destination one.

Parameters
destination (PathLike[str] | str) – Destination path

Return type
None

create_qpfile(*qpfile*, *fps*)

Create a qp file from the current Chapters object

Parameters

- **qpfile** (PathLike[str] | str) – Qpfile path
- **fps** (Fraction) – Framerate Per Second

Return type
None

class vardautomation.chapterisation.**OGMChapters**(*chapter_file*, *extension='.txt'*)

Bases: *Chapters*

OGMChapters object.

An OGM based Chapters is a TXT file

Register a new OGMChapters object

Parameters

- **chapter_file** (PathLike[str] | str) – Chapters file path
- **extension** (*str*) –

chapter_file: *VPath*

Chapters file path

create(*chapters*, *fps*)

Create the current Chapters object by passing a list of Chapter

Parameters

- **chapters** (List[*Chapter*]) – List of Chapter
- **fps** (Fraction) – Framerate Per Second

Return type
None

set_names(*names*)

Change/set names of the current Chapters object

Parameters
names (Sequence[Optional[str]]) – List of optional names. A None value won't change the name of the current chapter list

Return type
None

shift_times(*frames*, *fps*)

Shift timestamps by given number of frames.

Parameters

- **frames** (int) – Corresponding number of frames to be shifted
- **fps** (Fraction) – Framerate Per Second

Return type
None

to_chapters(*fps*, *lang*)

Convert the Chapters object to a list of chapter

Parameters

- **fps** (Fraction) – Framerate Per Second

- **lang** (Optional[*Lang*]) – Language of the chapter. If specified it will override the current language of the Chapters object

Return type
List[*Chapter*]

copy(*destination*)

Copy source chapter to destination and change target of *Chapters.chapter_file* to the destination one.

Parameters
destination (PathLike[str] | str) – Destination path

Return type
None

create_qpfile(*qpfile*, *fps*)

Create a qp file from the current Chapters object

Parameters

- **qpfile** (PathLike[str] | str) – Qpfile path
- **fps** (Fraction) – Framerate Per Second

Return type
None

class vardautomation.chapterisation.**MatroskaXMLChapters**(*chapter_file*, *extension='.xml'*)

Bases: *Chapters*

MatroskaXMLChapters object

An MatroskaXML based Chapters is a .xml file

Register a new MatroskaXMLChapters object

Parameters

- **chapter_file** (PathLike[str] | str) – Chapters file path
- **extension** (*str*) –

chapter_file: *VPath*

Chapters file path

create(*chapters*, *fps*)

Create the current Chapters object by passing a list of Chapter

Parameters

- **chapters** (List[*Chapter*]) – List of Chapter
- **fps** (Fraction) – Framerate Per Second

Return type
None

set_names(*names*)

Change/set names of the current Chapters object

Parameters
names (Sequence[Optional[str]]) – List of optional names. A None value won't change the name of the current chapter list

Return type
None

shift_times(*frames*, *fps*)

Shift timestamps by given number of frames.

Parameters

- **frames** (int) – Corresponding number of frames to be shifted
- **fps** (Fraction) – Framerate Per Second

Return type
None

to_chapters(*fps*, *lang=None*)

Convert the Chapters object to a list of chapter

Parameters

- **fps** (Fraction) – Framerate Per Second
- **lang** (Optional[*Lang*]) – Language of the chapter. If specified it will override the current language of the Chapters object

Return type
List[*Chapter*]

copy(*destination*)

Copy source chapter to destination and change target of *Chapters.chapter_file* to the destination one.

Parameters
destination (PathLike[str] | str) – Destination path

Return type
None

create_qpfile(*qpfile*, *fps*)

Create a qp file from the current Chapters object

Parameters

- **qpfile** (PathLike[str] | str) – Qpfile path
- **fps** (Fraction) – Framerate Per Second

Return type
None

class vardautomation.chapterisation.**MplsChapters**(*chapter_file*)

Bases: *Chapters*

MplsChapters object

Register a new Chapters object

Parameters
chapter_file (PathLike[str] | str) – Chapters file path

m2ts: *VPath*

Associated m2ts file name

chapters: List[*Chapter*]

Current list of chapters of this MplsChapters

fps: Fraction

Framerate Per Second

create(*chapter_file*) = <function MplsChapters>

Parameters
chapter_file (*os.PathLike[str] | str*) –

Return type
None

set_names(*chapter_file*) = <function MplsChapters>

Parameters
chapter_file (*os.PathLike[str] | str*) –

Return type
None

shift_times(*chapter_file*) = <function MplsChapters>

Parameters
chapter_file (*os.PathLike[str] | str*) –

Return type
None

to_chapters(*fps=None*, *lang=None*)

Convert the Chapters object to a list of chapter

Parameters

- **fps** (Optional[Fraction]) – Framerate Per Second
- **lang** (Optional[*Lang*]) – Language of the chapter. If specified it will override the current language of the Chapters object

Return type
List[*Chapter*]

copy(*destination*)

Copy source chapter to destination and change target of *Chapters.chapter_file* to the destination one.

Parameters
destination (PathLike[str] | str) – Destination path

Return type
None

create_qpfile(*qpfile*, *fps*)

Create a qp file from the current Chapters object

Parameters

- **qpfile** (PathLike[str] | str) – Qpfile path
- **fps** (Fraction) – Framerate Per Second

Return type
None

chapter_file: *VPath*

Chapters file path

class vardautomation.chapterisation.**IfoChapters**(*chapter_file*)

Bases: *Chapters*

IfoChapters object

Register a new Chapters object

Parameters
chapter_file (PathLike[str] | str) – Chapters file path

chapters: List[*Chapter*]

Current list of chapters of this IfoChapters

fps: Fraction

Framerate Per Second

create(*chapter_file*) = **<function IfoChapters>**

Parameters
chapter_file (*os.PathLike[str] | str*) –

Return type
None

set_names(*chapter_file*) = **<function IfoChapters>**

Parameters
chapter_file (*os.PathLike[str] | str*) –

Return type
None

shift_times(*chapter_file*) = **<function IfoChapters>**

Parameters
chapter_file (*os.PathLike[str] | str*) –

Return type
None

to_chapters(*fps=None*, *lang=None*)

Convert the Chapters object to a list of chapter

Parameters

- **fps** (Optional[Fraction]) – Framerate Per Second
- **lang** (Optional[*Lang*]) – Language of the chapter. If specified it will override the current language of the Chapters object

Return type
List[*Chapter*]

copy(*destination*)

Copy source chapter to destination and change target of *Chapters.chapter_file* to the destination one.

Parameters
destination (PathLike[str] | str) – Destination path

Return type
None

create_qpfile(*qpfile*, *fps*)

Create a qp file from the current Chapters object

Parameters

- **qpfile** (PathLike[str] | str) – Qpfile path
- **fps** (Fraction) – Framerate Per Second

Return type
None

chapter_file: *VPath*

Chapters file path

class vardautomation.chapterisation.**MplsReader**(*bd_folder*, *lang=UNDEFINED*, *default_chap_name='Chapter'*)

MPLS reader

Initialise a MplsReader.

Parameters

- **bd_folder** (PathLike[str] | str) – A valid bluray folder path should contain a BDMV and CERTIFICATE folders
- **lang** (*Lang*) – Language to be set, defaults to UNDEFINED
- **default_chap_name** (str) – Prefix used as default name for the generated chapters, defaults to 'Chapter'

class MplsFile(*mpls_file*, *mpls_chapters*)

Class for MPLS file

Create new instance of MplsFile(mpls_file, mpls_chapters)

Parameters

- **mpls_file** (VPath) –
- **mpls_chapters** (*List[MplsChapters]*) –

mpls_file: *VPath*

MPLS file path

mpls_chapters: List[*MplsChapters*]

MPLS Chapters list

bd_folder: *VPath*

Bluray Disc folder path (usually PLAYLIST)

mpls_folder: *VPath*

MPLS folder path

m2ts_folder: *VPath*

M2TS folder path (usually STREAMS)

lang: *Lang*

Language

default_chap_name: str

Prefix used as default name for the generated chapters

get_playlist()

Returns a list of all the mpls files contained in the folder specified in the constructor.

Return type
List[*MplsFile*]

write_playlist(*output_folder=None*, *, *chapters_obj=MatroskaXMLChapters*)

Extract and write the playlist folder

Parameters

- **output_folder** (Union[PathLike[str], str, None]) – Output path where the chapters will be written. If not specified will write in the MPLS folder, defaults to None
- **chapters_obj** (Type[*Chapters*]) – Type of wanted chapters, defaults to MatroskaXMLChapters

Return type
None

parse_mpls(*mpls_file*)

Parse a mpls file and return a list of chapters that were in the MPLS file.

Parameters
mpls_file (PathLike[str] | str) – MPL file path

Return type
List[*MplsChapters*]

class vardautomation.chapterisation.**IfoReader**(*dvd_folder*, *lang=UNDEFINED*, *default_chap_name='Chapter'*)

IFO reader

Initialise a IfoReader

Parameters

- **dvd_folder** (PathLike[str] | str) – A valid dvd folder path should contain at least a VIDEO_TS folder.
- **lang** (*Lang*) – Language to be set, defaults to UNDEFINED
- **default_chap_name** (str) – Prefix used as default name for the generated chapters, defaults to 'Chapter'

dvd_folder: *VPath*

DVD Folder path

ifo_folder: *VPath*

Current IFO folder path

lang: *Lang*

Language

default_chap_name: str

Prefix used as default name for the generated chapters

write_programs(*output_folder=None*, *, *chapters_obj=MatroskaXMLChapters*, *ifo_file='VTS_01_0.IFO'*)

Extract and write the programs from the IFO file to chapters files

Parameters

- **output_folder** (Union[PathLike[str], str, None]) – Output path where the chapters will be written. If not specified will write in the IFO folder, defaults to None
- **chapters_obj** (Type[*Chapters*]) – Type of wanted chapters, defaults to MatroskaXMLChapters
- **ifo_file** (str) – Name of the ifo file, defaults to 'VTS_01_0.IFO'

Return type
None

parse_ifo(*ifo_file*)

Parse a mpls file and return a list of chapters that were in the ifo file.

Parameters
ifo_file (PathLike[str] | str) – IFO file path

Return type
List[*IfoChapters*]

1.5 Comparison

class vardautomation.comp.**Writer**(*value*)

Writer to be used to extract frames

FFMPEG = 1

FFmpeg encoder

IMWRI = 2

vapoursynth.core.imwri Vapoursynth plugin

OPENCV = 3

opencv library

PILLOW = 4

Pillow library

PYQT = 5

PyQt library

PYTHON = 6

Pure python implementation

class vardautomation.comp.**PictureType**(*value*)

A simple enum for picture types.

I = b'I'

I frames

P = b'P'

P frames

B = b'B'

B frames

class vardautomation.comp.**SlowPicsConf**(*collection_name='runpy'*, *public=True*, *optimise=True*, *nsfw=False*, *remove_after=None*)

Slow.pics configuration

Create new instance of SlowPicsConf(collection_name, public, optimise, nsfw, remove_after)

Parameters

- **collection_name** (*str*) –
- **public** (*bool*) –
- **optimise** (*bool*) –
- **nsfw** (*bool*) –
- **remove_after** (*Optional[int]*) –

collection_name: str

Slowpics's collection name.

Default is the name of the current script

public: bool

Make the comparison public

optimise: bool

If 'true", images will be losslessly optimised

nsfw: bool

If images not suitable for minors (nudity, gore, etc.)

remove_after: Optional[int]

Remove after N days

class vardautomation.comp.**Comparison**(*clips*, *path='comps'*, *num=15*, *frames=None*, *picture_type=None*)

Extract frames, make diff between two clips and upload to slow.pics

Parameters

- **clips** (Dict[str, VideoNode]) – Named clips.
- **path** (PathLike[str] | str) – Path to your comparison folder, defaults to 'comps'
- **num** (int) – Number of frames to extract, defaults to 15
- **frames** (Optional[Iterable[int]]) – Additionnal frame numbers that will be added to the total of num, defaults to None
- **picture_type** (Union[*PictureType*, List[*PictureType*], None]) – Select picture types to pick, default to None

extract(*writer=Writer.PYTHON*, *compression=-1*, *force_bt709=False*)

Extract images from the specified clips in the constructor

Parameters

- **writer** (*Writer*) – Writer method to be used, defaults to Writer.PYTHON
- **compression** (int) – Compression level. It depends of the writer used, defaults to -1 which means automatic selection
- **force_bt709** (bool) – Force BT709 matrix before conversion to RGB24, defaults to False

Return type
None

magick_compare()

Make an image of differences between the first and second clip using ImageMagick. Will raise an exception if more than 2 clips are passed to the constructor.

Return type
None

upload_to_slowpics(*config*)

Upload to slow.pics with given configuration

Parameters
config (*SlowPicsConf*) – NamedTuple which contains the uploading configuration

Return type
None

vardautomation.comp.**make_comps**(*clips*, *path='comps'*, *num=15*, *frames=None*, ***, *picture_types=None*, *force_bt709=False*, *writer=Writer.PYTHON*, *compression=-1*, *magick_compare=False*, *slowpics_conf=None*)

Convenience function for *Comparison*.

Parameters

- **clips** (Dict[str, VideoNode]) – Named clips.
- **path** (PathLike[str] | str) – Path to your comparison folder, defaults to 'comps'
- **num** (int) – Number of frames to extract, defaults to 15
- **frames** (Optional[Iterable[int]]) – Additionnal frame numbers that will be added to the total of num, defaults to None
- **picture_types** (Union[*PictureType*, List[*PictureType*], None]) – Select picture types to pick, default to None
- **force_bt709** (bool) – Force BT709 matrix before conversion to RGB24, defaults to False
- **writer** (*Writer*) – Writer method to be used, defaults to Writer.PYTHON
- **compression** (int) – Compression level. It depends of the writer used, defaults to -1 which means automatic selection
- **magick_compare** (bool) – Make diffs between the first and second clip. Will raise an exception if more than 2 clips are passed to clips, defaults to False
- **slowpics_conf** (Optional[*SlowPicsConf*]) – slow.pics configuration. If specified, images will be uploaded following this configuration

Return type
None

1.6 Binary Path

class vardautomation.binary_path.**BinaryPath**

Class storing the path of the variable binaries used in vardautomation.

Just edit one of these attributes if the binary is not in your environment path

Return type
NoReturn

eac3to: *VPath* = **VPath('eac3to')**

https://www.videohelp.com/software/eac3to

https://en.wikibooks.org/wiki/Eac3to/How_to_Use

fdkaac: *VPath* = **VPath('fdkaac')**

https://github.com/nu774/fdkaac

https://en.wikipedia.org/wiki/Fraunhofer_FDK_AAC

Also available in ffmpeg with `--enable-libfdk-aac`

ffmpeg: *VPath* = **VPath('ffmpeg')**

https://www.ffmpeg.org/

ffmsindex: *VPath* = **VPath('ffmsindex')**

https://github.com/FFMS/ffms2

flac: *VPath* = **VPath('flac')**

https://xiph.org/flac/index.html

mkvextract: *VPath* = **VPath('mkvextract')**

https://mkvtoolnix.download/

https://mkvtoolnix.download/doc/mkvextract.html

mkvmerge: *VPath* = **VPath('mkvmerge')**

https://mkvtoolnix.download/

https://mkvtoolnix.download/doc/mkvextract.html

nvencc: *VPath* = **VPath('nvencc')**

https://github.com/rigaya/NVEnc

opusenc: *VPath* = **VPath('opusenc')**

https://github.com/xiph/opus-tools

Also available in ffmpeg

qaac: *VPath* = **VPath('qaac')**

https://sites.google.com/site/qaacpage/

rclone: *VPath* = **VPath('rclone')**

https://rclone.org/

sox: *VPath* = **VPath('sox')**

http://sox.sourceforge.net/

x264: *VPath* = **VPath('x264')**

https://www.videolan.org/developers/x264.html

x265: *VPath* = **VPath('x265')**

http://msystem.waw.pl/x265/

https://bitbucket.org/multicoreware/x265_git/wiki/Home

1.7 Language

class vardautomation.language.**Lang**(*language*, *, *iso639_variant='B'*)

Basic language class

Parameters

- **language** (Language) – Language class of the package langcodes
- **iso639_variant** (str) – Optional variant to get the 'bibliographic' code instead, defaults to 'B'

name: **str**

Name of the language

ietf: **str**

IETF BCP 47 language code

iso639: **str**

ISO-639 language code

classmethod make(*ietf*)

Make a new Lang based on IETF

Parameters
ietf (Optional[str]) – IETF BCP 47 language code

Return type
Lang

Returns
A new Lang object

vardautomation.language.**FRENCH = <vardautomation.language.Lang object>**

French Lang object

vardautomation.language.**ENGLISH = <vardautomation.language.Lang object>**

English Lang object

vardautomation.language.**JAPANESE = <vardautomation.language.Lang object>**

Japanese Lang object

vardautomation.language.**UNDEFINED = <vardautomation.language.Lang object>**

Undefined Lang object

1.8 VPath

class vardautomation.vpathlib.**VPath**(**args*, ***kwargs*)

Modified version of pathlib.Path

Construct a PurePath from one or several strings and or existing PurePath objects. The strings and path objects are combined so as to yield a canonicalized path, which is incorporated into the new PurePath object.

format(**args*, ***kwargs*)

Return type
VPath

Returns
Formatted version of *vpath*, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}')

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

set_track(*track_number*, /)

Set the track number by replacing the substitution "{track_number}" by the track_number specified

Parameters
track_number (int) – Track number

Return type
VPath

Returns
Formatted VPath

to_str()

Return type
str

Returns
String representation of the path, suitable for passing to system calls.

append_stem(*stem*)

Append stem at the end of the VPath stem

Parameters
stem (str) – Stem to add

Return type
VPath

Returns
New VPath with the stem appended

append_suffix(*suffix*)

Append stem at the end of the VPath suffix. Stolen from pathlib3x

Parameters
suffix (str) – Suffix to add. It has to start with '.'

Return type
VPath

Returns
New VPath with the file suffix appended

copy(*target*, **, follow_symlinks=True*)

Wraps shutil.copy. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.copy

Parameters

- **target** (PathLike[str] | str) – See Python official documentation
- **follow_symlinks** (bool) – See Python official documentation

Return type
None

copy2(*target*, *follow_symlinks=True*)

Wraps shutil.copy2. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.copy2

Parameters

- **target** (PathLike[str] | str) – See Python official documentation
- **follow_symlinks** (bool) – See Python official documentation

Return type
None

copyfile(*target*, *follow_symlinks=True*)

Wraps shutil.copyfile. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.copyfile

Parameters

- **target** (*VPath*) – See Python official documentation
- **follow_symlinks** (bool) – See Python official documentation

Return type
None

copymode(*target*, *follow_symlinks=True*)

Wraps shutil.copymode. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.copymode

Parameters

- **target** (PathLike[str] | str) – See Python official documentation
- **follow_symlinks** (bool) – See Python official documentation

Return type
None

copystat(*target*, *follow_symlinks=True*)

Wraps shutil.copystat. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.copystat

Parameters

- **target** (PathLike[str] | str) – See Python official documentation
- **follow_symlinks** (bool) – See Python official documentation

Return type
None

copytree(*target*, *symlinks=False*, *ignore=None*, *copy_function=<function copy2>*, *ignore_dangling_symlinks=True*, *dirs_exist_ok=False*)

Wraps shutil.copytree. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.copytree

Parameters

- **target** (PathLike[str] | str) – See Python official documentation
- **symlinks** (bool) – See Python official documentation
- **ignore** (Optional[Callable[[PathLike[str] | str, List[str]], Iterable[str]]]) – See Python official documentation
- **copy_function** (Callable[[PathLike[str] | str, PathLike[str] | str], Any]) – See Python official documentation
- **ignore_dangling_symlinks** (bool) – See Python official documentation
- **dirs_exist_ok** (bool) – See Python official documentation

Return type
None

rmtree(*ignore_errors=False*, *onerror=None*)

Wraps shutil.rmtree. Stolen from pathlib3x.

https://docs.python.org/3/library/shutil.html#shutil.rmtree

Parameters

- **ignore_errors** (bool) – See Python official documentation
- **onerror** (Optional[Callable[[Callable[..., Any], str, Union[Tuple[Type[BaseException], BaseException, traceback], Tuple[None, None, None]]], Any]]) – See Python official documentation

Return type
None

rm(*ignore_errors=False*)

Wraps os.remove.

Parameters
ignore_errors (bool) – Ignore errors emitted by os.remove

Return type
None

1.9 Types

`vardautomation.types.AnyPath = os.PathLike[str] | str`

Represent a PEP 604 union type

E.g. for int | str

`vardautomation.types.UpdateFunc`

alias of `Callable[[int, int], None]`

`vardautomation.types.VPSIdx`

alias of `Callable[[str], VideoNode]`

1.10 Internal functions

class `vardautomation.utils.Properties`

Collection of methods to get some properties from the parameters and/or the clip

classmethod `get_colour_range`(*cls*, *params*, *clip*)

Get the luma colour range specified in the params. Fallback to the clip properties.

Parameters

- **params** (`List[str]`) – Settings of the encoder.
- **clip** (`VideoNode`) – Source

Return type
`Tuple[int, int]`

Returns
A tuple of min_luma and max_luma value

static `get_depth`(*clip*, /)

Returns the bit depth of a VideoNode as an integer.

Parameters
clip (`VideoNode`) – Source clip

Return type
`int`

Returns
Bitdepth

static `get_csp`(*clip*)

Get the colourspace a the given clip based on its format

Parameters
clip (`VideoNode`) – Source clip

Return type
`str`

Returns
Colourspace suitable for x264

static get_encoder_name(*path*)

Get the encoder name from the file's tags

Parameters
path (PathLike[str] | str) – File path

Return type
str

Returns
Encoder name

static get_matrix_name(*frame*, *key*)

Gets FrameProp prop from frame frame with expected type t and then returns a corresponding string. This is necessary because x264 does not accept integers for the matrix/primaries/transfer.

For a full list of accepted matrices, please check
http://www.chaneru.com/Roku/HLS/X264_Settings.htm#colormatrix

Parameters

- **frame** (VideoFrame) – Frame containing props
- **key** (str) – Prop to get. Must be _Matrix, _Primaries, or _Transfer!
- **t** – Type of prop

Return type
str

Returns
string signalling the clip's matrix

static get_prop(*frame*, *key*, *t*)

Gets FrameProp prop from frame frame with expected type t to satisfy the type checker. Function borrowed from lvsfunc.

Parameters

- **frame** (VideoFrame) – Frame containing props
- **key** (str) – Prop to get
- **t** (Type[TypeVar(T)]) – Type of prop

Return type
TypeVar(T)

Returns
frame.prop[key]

vardautomation.render.**clip_async_render**(*clip: vs.VideoNode*, *outfile: Optional[BinaryIO] = None*, *timecodes: None = None*, *progress: Optional[str] = 'Rendering clip...'*, *callback: RenderCallback | List[RenderCallback] | None = None*) → None

vardautomation.render.**clip_async_render**(*clip: vs.VideoNode*, *outfile: Optional[BinaryIO] = None*, *timecodes: TextIO = None*, *progress: Optional[str] = 'Rendering clip...'*, *callback: RenderCallback | List[RenderCallback] | None = None*) → List[float]

Render a clip by requesting frames asynchronously using clip.frames, providing for callback with frame number and frame object.

This is mostly a re-implementation of VideoNode.output, but a little bit slower since it's pure python. You only really need this when you want to render a clip while operating on each frame in order or you want timecodes without using vspipe.

Original function borrowed from lvsfunc.render.clip_async_render.

Parameters

- **clip** (`VideoNode`) – Clip to render.
- **outfile** (`Optional[BinaryIO]`) – Y4MPEG render output BinaryIO handle. If None, no Y4M output is performed. Use `sys.stdout.buffer` for stdout. (Default: None)
- **timecodes** (`Optional[TextIO]`) – Timecode v2 file TextIO handle. If None, timecodes will not be written.
- **progress** (`Optional[str]`) – String to use for render progress display. If empty or `None`, no progress display.
- **callback** (`Union[Callable[[int, VideoFrame], None], List[Callable[[int, VideoFrame], None]], None]`) – Single or list of callbacks to be preformed. The callbacks are called when each sequential frame is output, not when each frame is done.

Return type
`Optional[List[float]]`

Returns
List of timecodes from rendered clip.

`vardautomation.render.`**`audio_async_render`**(*audio*, *outfile*, *header=WaveHeader.AUTO*, *progress='Rendering audio...'*)

Render an audio by requesting frames asynchronously using audio.frames.

Implementation-like of VideoNode.output for an AudioNode that isn't in the Cython side yet.

Parameters

- **audio** (`AudioNode`) – Audio to render.
- **outfile** (`BinaryIO`) – Render output BinaryIO handle.
- **header** (*`WaveHeader`*) – Kind of Wave header. WaveHeader.AUTO adds a Wave64 header if the audio
  - Has more than 2 channels
  - Has a bitdepth > 16
  - Has more than 44100 samples
- **progress** (`Optional[str]`) – String to use for render progress display. If empty or `None`, no progress display.

Return type
`None`

class `vardautomation.render.`**`WaveFormat`**(*value*)

WAVE form wFormatTag IDs Complete list is in mmreg.h in Windows 10 SDK.

class `vardautomation.render.`**`WaveHeader`**(*value*)

Wave headers

PYTHON MODULE INDEX

V

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z