**Milo Documentation**

Date: 30/08/2019

**UI (.py):**

1. add\_link.py
2. help\_menu.py
3. preset.py
4. converter.py
5. overlay.py
6. splice.py
7. milo.py
8. myfilebrowser.py
9. logo\_large\_rc.py
10. logo\_small\_rc.py

**UI (.ui):**

1. add\_link.ui
2. help\_menu.ui
3. preset.ui
4. converter.ui
5. overlay.ui
6. splice.ui
7. milo.ui
8. myfilebrowser.ui
9. logo\_large.qrc
10. logo\_small.qrc

**UI Modules:**

1. converter\_main.py
2. converter\_thread.py
3. overlay\_main.py
4. overlay\_thread.py
5. splice\_main.py
6. splice\_thread.py
7. milo\_main.py
8. three\_dialogs.py
9. myfilebrowser\_main.py

**Function Module:**

* [generators.py](#generators)
* [[](#ffmpeg)*[FFMPEGGenerator](#ffmpeg)*[]](#ffmpeg)
* [[*CMDGenerator*]](#cmd)
* [[*OverlayCMDGenerator*]](#overlay)
* [[*SpliceCMDGenerator*]](#splice)
* [[Sample command and explanation]](#sample)

**Note:**

UI packages are done by ‘Qt Designer’, and using Pyuic to generate corresponding py files. Files end with qrc are converted to py files using Pyrrc.

‘milo\_main.exe’ only works with correct ‘milo.config’ file. Content of ‘milo.config’ is in JSON format. It has 3 main keys [‘ffmpeg\_folder\_path’], [‘preset\_book’] and [‘link\_book’]

**generators.py**

**def write\_config(content)**

* This global function takes in a dictionary, and dumps the content into ‘milo.config’.

**def read\_config()**

* This global function returns a dictionary, it is the content from ‘milo.config’.

***class FFMPEGGenerator(object):***

1. *get\_codecs(self) :*

This function returns a list [ ] ,and contains all the codecs supported by FFMPEG

How to use:

a = ***FFMPEGGenerator() a.get\_codecs()***

CMD command: ffmpeg.exe -codecs

1. *split\_v\_a\_codecs(self,codec\_ls):*

This function takes in a list as a variable, and returns 2 lists [video codecs] and [audio codecs]

How to use:

a = ***FFMPEGGenerator()*** codec\_list = ***a.get\_codecs()***

v\_codecs,a\_codecs = a. split\_v\_a\_codecs(codec\_list)

1. *get\_muxing\_supported\_fmt(self):*

This function returns a list [] that contains all the supporting muxing formats

How to use:

a = ***FFMPEGGenerator()***

format\_list = a. get\_muxing\_supported\_fmt()

1. *output\_fmt(self):*

This function returns all the formats that are supported by FFMPEG

How to use:

a = ***FFMPEGGenerator()***

format\_list = a.output\_fmt()

CMD command: ffmpeg.exe -formats

1. *get\_resolution(self,file\_path)*

This function takes in a string variable, file\_path is either a video file or image file.

It returns a string for example ‘1920x1080’

How to use:

a = ***FFMPEGGenerator()***

resolution = a.get\_resolution(file\_path)

CMD command: ffprobe.exe -v error -select\_streams v:0 -show\_entries stream=width,height -of csv=s=x:p=0 input.mp4

1. *get\_resized\_resolution(self,ratio,file\_path)*

ratio is a number ranging from 0.2 – 1 , file\_path is a string variable(either a video file or image file.)

It returns a string for example ‘1920x1080’.

How to use:

a = ***FFMPEGGenerator()***

r = 0.5

resized\_resolution = a.get\_resized\_resolution(r,file\_path)

1. *get\_total\_frame(self,file\_path)*

This function takes in a string variable, and input file can ONLY be a video file.

It returns total frame of the video as a string

How to use:

a = ***FFMPEGGenerator()***

total\_frame = a.get\_total\_frame(file\_path)

CMD Command: ffprobe.exe -select\_streams v -show\_streams input.mp4

1. *get\_video\_audio\_status(self,input\_video)*

This function takes in a string variable. It checks whether the video has a audio stream.

Return value is a string, ‘0’ or ‘1’

How to use:

a = ***FFMPEGGenerator()***

audio\_status = a.get\_video\_audio\_status(input\_video)

***class CMDGenerator(object)***

1. *get\_parameter\_cmd(self,frame\_rate,crf,audio\_bitrate)*

This function takes in 3 string variables, and generate a cmd command for FFMPEG.

Audio\_bitrate has a unit (UPPER ‘M’ , lower ‘k’ eg. 128k,1M)

How to use :

a = CMDGenerator()

parameter\_cmd = a.get\_parameter\_cmd(‘25’,’20’,’128k’)

sample CMD: ‘ -r 25 -crf 20 -ab 128k ’

1. *get\_codec\_cmd(self, audio\_codecs\_list, video\_codecs\_list, selected\_acodec, selected\_vcodec)*

This function takes in 4 variables, ***‘audio\_codecs\_list’ and ‘video\_codecs\_list’,*** are two lists, ***‘selected\_acodec’ and ‘selected\_vcodec’*** are two string variables.

sample CMD: ‘ -vcodec h264 -acodec aac ‘

1. *get\_preset\_cmd(self, preset\_parameter\_list)*

Takes in a list variable. It has following elements :

frame\_rate = preset\_parameter\_list[0].strip()

crf = preset\_parameter\_list[1].strip()

audio\_bitrate = preset\_parameter\_list[2].strip()

selected\_acodec = preset\_parameter\_list[3].strip()

selected\_vcodec = preset\_parameter\_list[4].strip()

sample CMD: ‘-vcodec h264 -acodec aac -r 25 -crf 20 -ab 128k ‘

1. *create\_cmd(self,input\_file,output\_file,codec\_cmd,parameter\_cmd,extra\_cmd,image\_sequence,output\_resolution=None)*

This function returns a cmd that used to execute video conversion.

Variable ‘image\_sequence’ is a bool value.

Keyword argument ‘output\_resolution’ should be string (eg.’1920x1080’)

Format:

ffmpeg -i input\_file + resize\_cmd + codec\_cmd + parameter\_cmd + extra\_cmd + output\_file

Note: ‘ -pix\_fmt yuv420p ’ is commonly used color system, ‘ -y ’ will overwrite output file if it exists

sample CMD: ‘ ffmpeg.exe -i input.mp4 -vf scale=1920x1080 -vcodec h264 -acodec aac -r 25 -crf 20 -ab 128k -pix\_fmt yuv420p -y output.mov’

***Class OverlayCMDGenerator(CMDGenerator)***

Inherit two methods from class CMDGenerator, rewrite create\_cmd method.

1. *get\_text\_CMD(self, file\_name, project\_name, author\_name, task\_name, version, frame\_rate, original\_resolution, resized\_resolution ,total\_frame)*

All input variables are string, it creates a CMD for overlay text. Text size and position are based on the resolution of input video or image. It returns a CMD as string.

Note: each text label needs a ‘drawtext’ option.

Sample CMD: drawtext="fontfile=./resources/arial.ttf:text=s’Circle Line’:fontcolor=white:fontsize=20:x=w\*0.01:y=h\*0.02

1. *get\_logo\_padding\_cmd(self,resolution)*

This function takes in one string variable, it calculates the thickness of top and bottom black bars based on input resolution.

1. *create\_cmd(self,input\_file,logo\_file,output\_file,codec\_cmd,parameter\_cmd,text\_cmd,logo\_padding\_cmd,image\_sequence\_flag)*

This function returns a cmd that used to execute video conversion.

Variable ‘image\_sequence\_flag’ is a bool value.

Sample CMD:

ffmpeg.exe -i input.mp4 -i Omens\_logo.png -filter\_complex "[0:v]scale=4096x2808[resized];[resized]pad=4096:3650:0:421[padded];[1:v]scale=337:337

[img]; [padded][img]overlay=main\_w-overlay\_w-10:10",

drawtext="fontfile=./resources/arial.ttf:text='Barry':fontcolor=white:fontsize=81:x=w\*0.01:y=

h\*0.02",

drawtext="fontfile=./resources/arial.ttf:text='TRN0320\_SH0080\_painted\_converted':

fontcolor=white:fontsize=81:x=(w-tw)/2:y=h\*0.06",

drawtext="fontfile=./resources/arial.ttf:text='Circle Line':fontcolor=white:fontsize=122:x=

(w-tw)/2:y=h\*0.01",

drawtext="fontfile=./resources/arial.ttf:text='4096x2808':fontcolor=white:fontsize=81:x=

(w-tw)/2:y=h\*0.9",

drawtext="fontfile=./resources/arial.ttf:text='Task\:test':fontcolor=white:fontsize=81:x=w\*0.01:y=h\*0.9",

drawtext="fontfile=./resources/arial.ttf:text='Version\:001':fontcolor=white:fontsize=81:x=

w\*0.01:y=h\*0.93",

drawtext="fontfile=./resources/arial.ttf:text='(50/25FPS)':fontcolor=white:fontsize=81:x=

w-tw-10:y=h\*0.93",

drawtext="fontfile=./resources/arial.ttf:text='%{frame\_num}':start\_number=1:fontcolor=white:fontsize=81:x=w-tw-10:y=h\*0.9"

-vcodec h264 -acodec aac -pix\_fmt yuv420p -r 25 -ab 128k -crf 23 -y output.mov

**Explanation of above CMD: [Overlay methodology]**

Format:

FFMPEG + input.mp4 + logo.png filter\_complex + codec + parameter + output.mov

Filter\_complex parameter:

"[0:v]scale=4096x2808[resized];[resized]pad=4096:3650:0:421[padded];[1:v]scale=337:337[img]; [padded][img]overlay=main\_w-overlay\_w-10:10"

1. Resize

[0:v] -> takes video stream of file at index[0] which is the video stream of input video, resize it to target size (based on % of compression) and label a output layer [resized]

1. Add black bars

Take [resized] layer as input, add padding to it. Output is layer [padded]

Padding format = original\_width : padding\_height : 0 : padding\_thickness

Padding\_height is the video height after adding black bars

Padding\_thickness is the thickness of each black bar (top and bottom)

1. Add logo picture

Take video stream of file at index[1] which is the image file as input, resize it and output a layer[img].

Overlay [img] on [padded]

drawtext="fontfile=./resources/arial.ttf:text='Barry':fontcolor=white:fontsize=81:x=w\*0.01:y=h\*0.02",

1. Add overlay text using ‘drawtext’ option

Assign font file, text, font size and position of text.

Each text has individual ‘drawtext’ option, connected using comma ‘,’

-vcodec h264 -acodec aac -pix\_fmt yuv420p -r 25 -ab 128k -crf 23 -y output.mov

5. Add codec and parameters

***class SpliceCMDGenerator(CMDGenerator)***

1. *get\_input\_file\_cmd (self,file\_path,identity,duration=None)*

Input variables are all string, keyword ‘duration’ is defined only if ‘identity’ is ‘still image’.

Function returns input file cmd

Sample CMD:

Video: ‘ -i input.mp4 ’

Still image: ‘ -loop 1 -t 10 -i input.png ’ (duration is 10 seconds)

1. *get\_image\_sequence\_inp\_cmd(self,folder\_path)*

This is function takes in a string variable, input path must be a folder path

It returns input cmd for image sequence

Sample CMD:

‘ -thread\_queue\_size 200 -f image2 -start\_number 0001 TVN\_001\_04d%.exr’

[[Full explanation]](#imgseqDetail)

1. *get\_filter\_inp\_cmd (self,resolution,scaling,index\_num,canvas\_index)*

All input variables are string.

‘resolution’ eg.

‘scaling’, either ‘increase’ or ‘decrease’

Function returns two strings, (resized\_cmd, canvas\_cmd)

Sample CMD:

resized\_cmd: ‘ [1:v]scale=1920x1080:force\_original\_aspect\_ratio=decrease[v1] ’

#[1:v] -> [1] is index of input file, take the video stream of file [1] as input, ‘1920x1080’ is target output resolution, ‘force\_original\_aspect\_ratio=decrease’ ensures unchanged original aspect ratio, scaling is most likely to be ‘decrease’.

canvas\_cmd: ‘ [2][v1] overlay=x='(W-w)/2':y='(H-h)/2':shortest=1[v1] ’

#[2] is index of canvas input, [v1] is resized video. Put video on canvas.

[[Full explanation]](#spliceDetail)

1. *get\_filter\_output\_cmd(self,index\_num,audio\_status,dummy\_audio\_index)*

All input variables are string.

Sample CMD:

‘ [v1][1:a ] ’

[[Full explanation]](#spliceDetail)

1. *create\_combine\_cmd(self,temp\_folder,preset\_cmd,sequence\_list,output\_file)*

This function create a cmd to combine file in temp folder.

Sample CMD:

ffmpeg.exe -i \Temp\temp\_1.mp4 -i \Temp\temp\_2.mp4 -i \Temp\temp\_3.mp4

-filter\_complex "[0:v][0:a][1:v][1:a][2:v][2:a] concat=n=3:v=1:a=1[v][a]" -map "[v]" -map "[a]"

-vcodec h264 -acodec aac -r 25 -ab 128k -crf 23 -y test.mp4

[[Full explanation]](#spliceDetail)

1. *create\_temp\_folder(self,output\_file)*

Create a temp folder, and return two string values (temp\_folder,extension)

1. *create\_cmd(self,output\_file,preset\_parameter\_list,sequence\_list,file\_list\_dd,target\_resolution)*

‘preset\_parameter\_list’ and ‘sequence\_list’ are list [ ]

‘file\_list\_dd’ is a dictionary { }

Return value is a list [ ]

Create splicing cmd list.

* input files are in sequence
* resize each input file and add to a canvas, add dummy audio if needed, output resized file to a temp folder.
* Resized file name is 'temp\_' + original file name
* Assume output file is 'out.mp4', the temp folder will be 'out\_Temp'

1. *get\_overlay\_cmd(self,preset\_cmd,preset\_parameter\_list,overlay\_info\_list,output\_file,target\_resolution)*

This function returns a cmd to add overlay to the output file

**Sample command and explanation**

* Converting a video:

‘ ffmpeg.exe -i input.mp4 -vf scale=1920x1080 -vcodec h264 -acodec aac -r 25 -crf 20 -ab 128k -pix\_fmt yuv420p -y output.mov’

* ‘-vf scale=1920x1080’ assign target resolution
* ‘-pix\_fmt yuv420p’ color format
* ‘-y’ overwrite existing file
* ‘-r’ frame rate
* ‘-ab’ audio bitrate
* Converting an image sequence

ffmpeg.exe -thread\_queue\_size 200 -f image2 -start\_number 0005 -i /img\_sequence\_folder/ img\_%05d.exr" -vcodec h264 -acodec aac -r 25 -ab 128k

-crf 23 -y out.mp4

* ‘-thread\_queue\_size ’ set to 200, allow reasonable amount of images
* ‘-start\_number’ specify the starting image, in above example, it starts from img5 to the last img.
* ‘img\_%05d.exr’ specify the length of index, in above case, 0005,0006,0007 … it has 5 digits -> ‘05%d’, all images have extension ‘.exr’
* Overlay a video

ffmpeg.exe -i input.mp4 -i Omens\_logo.png -filter\_complex "[0:v]scale=4096x2808[resized];[resized]pad=4096:3650:0:421[padded];[1:v]scale=337:337

[img]; [padded][img]overlay=main\_w-overlay\_w-10:10",

drawtext="fontfile=./resources/arial.ttf:text='Barry':fontcolor=white:fontsize=81:x=w\*0.01:y=

h\*0.02",

drawtext="fontfile=./resources/arial.ttf:text='TRN0320\_SH0080\_painted\_converted':

fontcolor=white:fontsize=81:x=(w-tw)/2:y=h\*0.06",

drawtext="fontfile=./resources/arial.ttf:text='Circle Line':fontcolor=white:fontsize=122:x=

(w-tw)/2:y=h\*0.01",

drawtext="fontfile=./resources/arial.ttf:text='4096x2808':fontcolor=white:fontsize=81:x=

(w-tw)/2:y=h\*0.9",

drawtext="fontfile=./resources/arial.ttf:text='Task\:test':fontcolor=white:fontsize=81:x=w\*0.01:y=h\*0.9",

drawtext="fontfile=./resources/arial.ttf:text='Version\:001':fontcolor=white:fontsize=81:x=

w\*0.01:y=h\*0.93",

drawtext="fontfile=./resources/arial.ttf:text='(50/25FPS)':fontcolor=white:fontsize=81:x=

w-tw-10:y=h\*0.93",

drawtext="fontfile=./resources/arial.ttf:text='%{frame\_num}':start\_number=1:fontcolor=white:fontsize=81:x=w-tw-10:y=h\*0.9"

-vcodec h264 -acodec aac -pix\_fmt yuv420p -r 25 -ab 128k -crf 23 -y output.mov

* Overlay an image sequence

ffmpeg.exe -thread\_queue\_size 200 -f image2 -start\_number 0005 -i /img\_sequence\_folder/ img\_%05d.exr" -i Omens\_logo.png -filter\_complex "[0:v]scale=4096x2808[resized];[resized]pad=4096:3650:0:421[padded];[1:v]scale=337:337

[img]; [padded][img]overlay=main\_w-overlay\_w-10:10",

drawtext="fontfile=./resources/arial.ttf:text='Barry':fontcolor=white:fontsize=81:x=w\*0.01:y=

h\*0.02",

drawtext="fontfile=./resources/arial.ttf:text='TRN0320\_SH0080\_painted\_converted':

fontcolor=white:fontsize=81:x=(w-tw)/2:y=h\*0.06",

drawtext="fontfile=./resources/arial.ttf:text='Circle Line':fontcolor=white:fontsize=122:x=

(w-tw)/2:y=h\*0.01",

drawtext="fontfile=./resources/arial.ttf:text='4096x2808':fontcolor=white:fontsize=81:x=

(w-tw)/2:y=h\*0.9",

drawtext="fontfile=./resources/arial.ttf:text='Task\:test':fontcolor=white:fontsize=81:x=w\*0.01:y=h\*0.9",

drawtext="fontfile=./resources/arial.ttf:text='Version\:001':fontcolor=white:fontsize=81:x=

w\*0.01:y=h\*0.93",

drawtext="fontfile=./resources/arial.ttf:text='(50/25FPS)':fontcolor=white:fontsize=81:x=

w-tw-10:y=h\*0.93",

drawtext="fontfile=./resources/arial.ttf:text='%{frame\_num}':start\_number=1:fontcolor=white:fontsize=81:x=w-tw-10:y=h\*0.9"

-vcodec h264 -acodec aac -pix\_fmt yuv420p -r 25 -ab 128k -crf 23 -y output.mov

[[Overlay Explanation]](#overlayMethodology)

* Splicing

NOTE:

* 1. input files are in sequence
  2. resize input file and add to a canvas, add dummy audio if needed, output resized file to a temp folder. Repeat this process for each input file.

Resized file name is 'temp\_' + original file name

Assume output file is 'out.mp4', the temp folder will be 'out\_Temp'

* 1. combine all the files in folder 'out\_Temp' and output 'out.mp4'
  2. delete 'out\_Temp' foder

Resize a video, add a dummy audio, and put on a canvas

ffmpeg.exe -i 1.mp4 -f lavfi -i color=s=1920x1080 -f lavfi -t 1 -i anullsrc -filter\_complex "[0:v]scale=1920x1080:force\_original\_aspect\_ratio=decrease[v0];[1][v0]overlay=x='(W-w)/2':y='(H-h)/2':shortest=1[v0];[v0][0:a] concat=n=1:v=1:a=1[v][a]" -map "[v]" -map "[a]" -vcodec h264 -acodec aac -r 25 -ab 128k -crf 23 -y temp\_1.mp4

* ‘-f lavfi -i color=s=1920x1080’ this option set up a canvas with resolution 1920x1080 (which is target output resolution), it is considered as a file, and index of canvas is [1]
* ‘-f lavfi -t 1 -i anullsrc’ this option set up a dummy audio, ‘-t 1’ gives a duration of one second, however the ‘-filter\_complex’ will auto map the duration according to the duration of input video. It is considered as a file, and index of dummy audio is [2].
* ‘"[0:v]scale=1920x1080:force\_original\_aspect\_ratio=decrease[v0]’ -> [0:v] takes video stream of input file(index [0]), resize it to target size and keep original aspect ratio unchanged, output a layer[v0]
* ‘[1][v0]overlay=x='(W-w)/2':y='(H-h)/2':shortest=1[v0]’ -> put resized video [v0] onto the canvas [1], output a layer[v0]

(duplicated layer name does not matter, it will overwrite)

* ‘[v0][0:a] concat=n=1:v=1:a=1[v][a]’, in above case the input video ‘1.mp4’ does have an audio stream, use its own audio.
* Take video steam[v0] and original audio steam [0:a]

(If using dummy audio, the audio stream will be [2:a] instead of [0:a])

* ‘concat=n=1:v=1:a=1’ -> n is number of input file (canvas and dummy audio are not considered as input files here), v and a are number of video steam and audio stream, always 1.
* ‘-map "[v]" -map "[a]"’ map video and audio to one file

NOTE : Repeat this step for each file

Combine and output a single file

ffmpeg.exe -i \Temp\temp\_1.mp4 -i \Temp\temp\_2.mp4 -i \Temp\temp\_3.mp4

-filter\_complex "[0:v][0:a][1:v][1:a][2:v][2:a] concat=n=3:v=1:a=1[v][a]" -map "[v]" -map "[a]"

-vcodec h264 -acodec aac -r 25 -ab 128k -crf 23 -y test.mp4

‘concat=n=3:v=1:a=1[v][a]"’ -> There are 3 input files -> n=3, video and audio stream are always 1.