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
  - [*FFMPEGGenerator*]
  - [*CMDGenerator*]
  - [*OverlayCMDGenerator*]
  - [*SpliceCMDGenerator*]
  - [Sample command and explanation]

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`

2. *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)

3. *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()

4. *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`

5. *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`

6. *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)

7. *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`

8. *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 '

2. *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 '

3. *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 '

4. *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
```

2. *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.

3. *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]

2. 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)

3. 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",
```

4. 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)
```

2. *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]

3. *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]

4. *get_filter_output_cmd(self,index_num,audio_status,dummy_audio_index)*
   All input variables are string.

```
Sample CMD:
‘ [v1][1:a ] ’
```

[Full explanation]

5. *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]

6. *create_temp_folder(self,output_file)*
   Create a temp folder, and return two string values (temp_folder,extension)

7. *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'

8. *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=w
hite: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]

- 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'
  3. combine all the files in folder 'out_Temp' and output 'out.mp4'
  4. 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.