This document is a prose description of the pre-ingest video processing used by The History Makers.

**Contents**

[Running the script](#_dqs4ff42ipq5)

[Python Terminology](#_5qeoemz5nt3c)

[Function definitions](#_ih1ece7k0gmp)

[main()](#_42zaujcou627)

[dependencies()](#_nrhdnf4k87fi)

[cd()](#_ee7pu6szk6vf)

[startup()](#_fcyvianfz3om)

[sizeloop()](#_gszchglp5345)

[walk()](#_850zhztaehrf)

[compare()](#_ifb7o1uer72o)

[makefflist()](#_p63mpkxrbbbi)

[ffprocess()](#_vo0vq2zu8ob)

[movevids()](#_rhdzqebmjxpv)

[updateFM()](#_5i771lgvjkx)

[verifyFM()](#_pq1pdsivd7n6)

[log()](#_ea0xmz92rcxz)

[External Scripts](#_5do82uthfehk)

[makevideos-sc.py](#_qxt97fvgxtx2)

[send-email.py](#_nhsnogrhxnm5)

[fm-stuff.py](#_slfhzeflprcm)

[hashmove.py](#_4kbsiiytnw6u)

[ffmpeg strings](#_ii73k4fq8ugv)

[concatenate](#_6nm8p01y79e7)

[route channel 2 audio out and re-wrap](#_iyqa7w26c8kc)

[flv](#_6gaewbg7qpy8)

[mpeg](#_w1k4d194fl31)

[mp4](#_4zjyshxzdhg8)

[ffmpeg stuff](#_wvq2gbkz47n7)

[-map](#_5abuydy1zrm3)

[-c:v, -c:a](#_8x8x66lmobat)

[-filter\_complex](#_jn2dq33r29lj)

[timecode burn-in](#_5136nmtcgsb1)

[watermark](#_flpop43nq2nw)

## Running the script

To run the script:

1. open a Terminal window
2. change directories, “cd”, into the directory where the script is located
   1. cd “/Volumes/G-SPEED Q/Titan-HD/HM/thm”
3. type “python makevideos.py” (without the quotes) and hit enter

## Terminology

Here’s a brief intro to some important terminology:

1. Function - a small set of steps used to complete a task. That task can be running the whole script or running a loop that measures a filesize every 1 second. Functions can have arguments, arguments are encased in parenthesis and separated by commas, e.g. fubar(foo, bar): fubar() is the function and foo, bar are the arguments. Functions can also return values to their parent/ calling functions, but they don’t have to.
2. Method - like a function but defined differently, defined in it’s library/ class. So ConfigParser is a library, config.get() is a method in the ConfigParser library
3. Class - a pretty abstract concept, classes are data structures and functions (methods) that we assign to objects
4. Variable - a placeholder for a value that can change or is assigned in multiple places. In python, variables don’t have a sigil (in javascript they have a $ preceding them, for example) and you don’t have to declare a type (integer, float, array, etc.). You just say foo = “bar” and that’s it
5. List - a set of values, separated by commas. A very common programming tactic in python is to make a list of stuff and then iterate over that list. Lists looks like foo=[“item1”,”item2”,variable,”item4”] and you can access individual items in the list by index, e.g. foo[0] (which = “item1”). Lists, confusingly, can also contain other lists and not just individual values
6. Dictionary - composed of {key:value} pairs, other languages would probably call it an associative array. So, you can have like {“Name”:”Jane Doe”,”Age”:”42”} and the “Name” and “Age” bits are the keys and the “Jane Doe” and “42” bits are the values. You can have multiple values for an individual key and those values would be in the form of a list

## Function definitions

### main()

**Arguments:** none

**Returns:** none

**Calling function:** none

**Description:**

The script is basically controlled by a single function called main(). main() initializes the variables used throughout the script and calls the major sub-functions such as ffprocess() and movevids(). main() isn’t very long, about 50 lines or so, and most of that is taken up by initializing variables, comments, and communicating with the logs/ emailing THM staff.

main() initializes variables using the python ConfigParser library. ConfigParser takes a txt file with key value pairs, e.g. “timecodefont: /Library/Fonts/Arial.ttf” where “timecodefont” is the key and “/Library/Fonts/Arial.ttf” is the value, and reads through it to find all of our file destinations. This configuration file is saved in the same directory as the script itself as video-post-process-config.txt

After initializing variables, main() enters what’s called a “try.” The python statement try: basically says “attempt to do this and if it doesn’t work, throw out an exception and either deal with the exception (a clause called “except:”) or just move on.” Our try: here basically runs the whole process and if the running fails, and it would only fail this try: because of a python error (like a misnamed variable), it throws out an Exception,e which logs the error and sends staff an email (you can see that at the bottom of the main() function).

The try: calls 4 functions and 1 external script (to send an email). The 4 functions it calls are:

1. startup() - which validates our inputs files and folders
2. makefflist() - which generates a python data structure called a dictionary which is composed of {key: value} pairs of accession directory full paths and their corresponding raw files (themselves in a list). A typical entry in this dictionary looks like {“/Volumes/G-SPEED Q/Titan-HD/HM/IncomingQT/A2016\_002\_001\_004/”:[”0187601.MOV”,”0187602.MOV”,”0187603.MOV”]} where the full path is the key and the list of raw files are the value. Makefflist makes 1 dictionary of these pairs and every file/ folder within IncomingQT is noted there
3. ffprocess() - which actually transcodes all of the files with ffmpeg
4. movevids() - which moves the completed videos to their various destinations and also generates their SHA1 hashes, and sends and verifies the SHA1 hashes with THM’s FIleMaker db.

The 1 external script that this try: calls is:

1. Send-email.py - which takes arguments for the txt of the message body and a file to attach, in this case the log. If the script has reached this step, it means that a single accession has been processed correctly (each function has it’s own watchers for fatal errors, so if those haven’t been triggered we get to this step) and here all it says is that the script completed successfully.

The part of the script that says ”def main():” actually only lays out the definition for what main() does. To invoke main(), to make it run, we have to call it, which we do on the very last line of makevideos.py.

### dependencies()

**Arguments:** none

**Returns:** none

**Calling function:** none

**Description:**

Before main(), we call a function named dependencies() which is defined near the top of the script. dependencies() loops through a list of software executables [“ffmpeg”,”ffprobe”], tries to run them, and if it can’t it throws out an error and stops the script. We need both of these pieces of software later on.

### cd()

**Arguments:** none

**Returns:** none

**Calling function:** none

**Description:**

This is actually a class but we use it a lot so I thought we’d include a discussion here with the functions. cd() comes from a pretty important and old command also called cd which stands for “change directory.” Changing directories is how you navigate a filesystem without a graphical user interface. Python doesn’t have a great way to change directories on its own so we use this instead.

Changing directories makes our life easier because we don’t have to use full paths in our filenames all the time. We can cd into a directory, call “ffprobe concat.mov” and if there’s a concat.mov file in that directory it’ll just work

In python, indents matter, they change the context of the code you’re running. That’s why this function is called a context manager, because it uses indents to define the inside and outside of a directory. Anything running inside of an indented block after we cd() is running in the directory we cd’d into. Once the indent is broken, we exit the directory.

### startup()

**Arguments:**

1. logfile - the log for this instance of the script
2. rawCaptures - full path to directory that we define in the config file that contains subdirectories named by accession. rawCaptures also known as IncomingQT
3. watermark - the white watermark file full path
4. fontfile - the font used to burn in the timecode full path
5. sunnas - the mount point for SUNNAS server
6. sunnascopyto - the /copyto subfolder on SUNNAS
7. xendata - the mount point for XENDATA server
8. xendatacopyto - the /coyto subfolder on XENDATA

**Returns:** none

**Calling function:** main()

**Description:**

Validates our inputs by checking that:

1. The watermark and font files used by ffmpeg are where we defined them in the config file
2. That sunnas and xendata are mounted where we defined them in the config file
3. That there are files to process in the rawCaptures directory we defined in the config file
4. That nothing is currently copying into the rawCaptures directory

startup() verifies that nothing is copying in by:

1. making a list of every file in rawCaptures
2. measuring the filesize of every file in rawCaptures then measuring it again 1-second later. We do this to see if this file is currently being copied-into. keeps measuring until the filesize is the same.
3. Then, once every file is stable in rawCaptures, we wait a couple minutes, and we measure everything again to make sure that a new file hasn’t been generated since the first time we made the list (called fs). If we didn’t do this step, we could have a new file being initialized and copied into that neither of our lists had noticed because it didn’t exist when the lists were first made.

### sizeloop()

**Arguments:**

1. thing - a file full-path

**Returns:** none

**Calling function:** walk()

**Description:**

sizeloop() takes the full path to a file, measures the filesize (with os.stat()), waits 1 second, measures it again. Then: if the measurements are the same, it means the file is stable and not being copied into, so it returns to startup(); if the measurements are not the same it calls itself again and will continue to do so until the size of the file stabilizes.

### walk()

**Arguments:**

1. pth - the path to our rawCapture directory

**Returns:** thefiles - a list of every file in the rawCapture directory

**Calling function:** startup()

**Description:**

walk() implements the os.walk() method to find every file in the capture directory. It’s actually called twice by startup, with a 6-minute delay. See startup() description for more info on why

### compare()

**Arguments:**

1. fs - a list of files in the rawCapture directory
2. fsagain - a list of files in the rawCapture directly, generated 6minutes after fs

**Returns:** True or False

**Calling function:** startup()

**Description:**

compare takes these two lists and just asks if everything that is in fsagain is also in fs. if these two lists are the same, we know that a new file has not been copied into our rawCapture directory, that everything is done copying into it.

### makefflist()

**Arguments:**

1. rawCaptures - the rawCapture directory defined in our config file
2. logfile - the log file for this instance of the script

**Returns:** fflist - a dictionary of {fullpath/to/accession/directory:[“list”,”of”,”rawfiles”]}

**Calling function:** main()

**Description:**

makefflist takes the argument for our rawCaptures directory, makes a list of every file and it’s parent directory and combines them into a dictionary where the keys are the full paths to individual accessions and the values are a list of the individual raw camera captures of that accession. It then sorts the dictionary by key name to process in alphabetical order.

### ffprocess()

**Arguments:**

1. acc - an individual accession folder full path
2. fflist - the complete dictionary of files to process
3. watermark - the full path to the white watermark file
4. scriptRepo - the path where our script is located
5. logfile - the log for this instance of our script

**Returns:** none

**Calling function:** main()

**Description:**

ffprocess takes the accession path provided by main(), accesses the raw files list in fflist[acc], cd’s into the accession directory, and makes the preservation mov and the derivatives.

The process for making the preservation mov is as follows:

1. make a txt file containing the list of raw captures in the accession directory
2. use the ffmpeg concat muxer to concatenate the raw captures together
3. take the rawconcat.mov andoutput just the second audio stream. we have to do this because for some reason, ffmepg tags this audio stream as a duplciate of the first and it hides this stream (the interviewer) from decoding applications like Premiere
4. takes the rawconcat and the second audio stream movs and re-wraps them together
5. deletes the rawconcat and second audio stream movs

the derivatives are all made by taking the concat.mov preservation file as an input, combining it with the various watermark and/ or timecode filters, and outputting the filetypes necessary. See “ffmpeg stuff” section for more info

### movevids()

**Arguments:**

1. acc - the full path to the current accession we’re working on
2. sunnascopyto - the /copyto subfolder on sunnas
3. sunnas - the mount point of the sunnas server
4. xendata - the mount point of the xendata server
5. xendatacopyto - the /copyto subfolder on xendata
6. xcluster - the parent folder to the machine we’re running on “HM”
7. scriptRepo - the path where the scripts are located
8. logfile - the log file for this instance of the script

**Returns:** none

**Calling function:** main()

**Description:**

movevids() coordinates the moving and hashing of the videos. First, in the current accession directory, copies the preservation mov to the folder for LC. Then it copies each of the four filetypes to their various destinations in the following way

1. flushes the stdout. stdout is where the result of a bash command goes, if we flush it we get rid of everything currently in there, we need to read this pipe in order to validate the file hases so we don’t want any noise in there
2. calls hashmove.py which:
   1. copies the file to its destination
   2. hashes the source and destination instances of the file
   3. if they match, it deletes the source file
   4. returns the hash value and filename.ext
3. get the hashes and filename.ext from hashmove.py output
4. logs them
5. adds them to a dictionary of {“accessionName.ext”:”hashvalue”}

then, it sends that dictionary to updateFM() where the hashes are sent to FileMaker. Then it sends that list to verifyFM where the hashes in FileMaker are verified against the computed hashes. verifyFM then tells this script whether or not the file can be moved from /copyto to it’s parent.

the parent folder of /copyto, the root directory of the mount point of SUNNAS and XENDATA, is a special place. Mark Strecker’s script checks this location every 5 minutes or so for files. If a file is found, it is moved to its final location on the server, and the hash is verified again with FileMaker.

We copy things into /copyto first because of the way that files are managed by a filesystem. The first thing that happens is that a file is allocated a space in the filesystem table, so there’s like a record that a file exists at a certain path. So, if we copied directly to the server mount point, from the nanosecond that we started that, the filesystem would register that the file exists, and when Mark’s script looks for a file there, it’ll grab whatever is there.

This is obviously bad for us because the file might not be done copying when Mark’ script picks it up.

So, instead, we copy to /copyto then we move it from /copyto to the root mount point afterwards. “moving” a file on a UNIX-like system, like mac, if the file is on the same filesystem, the move doesn’t actually touch the file data at all. Instead, it just renames the filepath in the filesystem’s file table. So, that process happens instantly and we don’t have to worry about Mark’s script moving a partial file.

### updateFM()

**Arguments:**

1. hashlist - the dictionary of hashes and filenames made by movevids()
2. scriptRepo - the directory where these scripts are run from
3. logfile - the log file for this instance of the script

**Returns:** none

**Calling function:** movevids()

**Description:**

updateFM() takes the hashlist, formats the filename and extensions in a way that FileMaker can deal with, then calls external script fm-stuff to pop the hash values into their appropriate fields in FileMaker

### verifyFM()

**Arguments:**

1. hashlist - the dictionary of hashes and filenames made by movevids()
2. scriptRepo - the directory where these scripts are run from
3. logfile - the log file for this instance of the script

**Returns:** moveyn - a boolean variable for whether or not a file should be moved from /copyto to it’s parent folder.

**Calling function:** movevids()

**Description:**

verifyFM() takes the hashlist, formats the filename and extensions in a way that FileMaker can deal with, then calls external script fm-stuff to select the stored hashes in filemaker and verify that they’re in there correctly. we do this because there’s not a great way to track errors caused by fm-stuff.py/ pyodbc library.

### log()

**Arguments:**

1. logfile - the log file for this instance of our script
2. msg - the text that we want to log

**Returns:** none

**Calling function:** all of them

**Description:**

this is just a shorthand function that generates 1 log message and writes it to our log file.

## 

## 

## External Scripts

### makevideos-sc.py

This script is used for processing Special Collections files. It’s the same as makevideos.py regular except that the concatenation is made optional, as Special Collections materials don’t always need it

### send-email.py

This utility script takes arguments for the text of the email body and a file to attach. Then, using the list provided in video-post-process-config.txt, it sends an email to staff with the supplied message text and attachment.

### fm-stuff.py

fm-stuff.py mediates the flow in information between FileMaker and the script/ files. It uses the pyodbc library to implement the ODBC protocol for talking to databases. It also relies on a third-party ODBC administrator application to manage the drivers in this process, which you can find out more about [here](http://www.actualtech.com/readme.php).

fm-stuff takes arguments for the various functions it performs:

1. -qSha - stands for Query SHA, asks filemaker to select and return the SHA1 value stored for a particular object of type “-fdigi” - formatDigital in FM
2. -uSha - stands for Update SHA, asks filemaker to update the SHA1 value for a particular object of type “-fdigi” - formatDigital in FM
3. -fdigi - stands for Format Digital, the field name in filemaker that corresponds to the 3 or 4 digit file extention
4. -id - this is the accession number

### hashmove.py

hashmove is something I wrote a long time ago because I didn’t have access to rsync on my Windows computer and I needed to verify the transfer of a lot of files over a radio network connection (with a tree growing in the way).

hashmove copies a file or folder from source to destination, hashes the source and destination copies, if the hashes match, the source is deleted. hashmove then prints out the source and destination hashes to Terminal, where we grip them with subprocess module, compare them, and report on if the file transfer was successful.

## ffmpeg strings

if we were to write out the bare ffmpeg strings used to make all of the files it would look like this:

### concatenate

ffmpeg -f concat -i concat.txt -map 0:v -map 0:a -c:v copy -c:a copy -timecode 00:00:00:00 rawconcat.mov

### route channel 2 audio out and re-wrap

So, through some bug, when two mono audio streams are copied via the concat protocol, the second one (corresponding to the interviewer mic) is labelled as an alternate in the same audio group as the interviewee stream. What this means is that the second channel isn’t available in Premiere, the NLE used by THM, and it’s inaudible to the trasncribers. To get around this we use the following ffmpeg:

ffmpeg -i rawconcat.mov -map 0:a:1 -map -0:d -map -0:v -c:a copy rawconcat-as2.mov

ffmpeg -i rawconcat.mov -i rawconcat-as2.mov -map 0:v -map 0:a:0 -map 1:a:0 -map 0:d -c copy concat.mov

### flv

ffmpeg -i concat.mov -i [/path/to/watermark.png] -filter\_complex "scale=320:180,overlay=0:0;[0:a:0][0:a:1]amerge=inputs=2[a]" -c:v libx264 -preset fast -b:v 700k -r 29.97 -pix\_fmt yuv420p -c:a aac -ac 2 -map 0:v -map "[a]" -timecode 00:00:00:00 -threads 0 [accessionNumber.flv]

### mpeg

ffmpeg -i concat.mov -target ntsc-dvd -filter\_complex "[0:a:0][0:a:1]amerge=inputs=2[a]" -b:v 5000k -vtag xvid -vf "drawtext=fontfile='[/path/to/fontfile.ttf]': timecode='00\:00\:00\:00': r=29.97: x=(w-tw)/2: y=h-(2\*lh): fontcolor=white: fontsize=72: box=1: boxcolor=0x00000099,scale=720:480" -map 0:v -map "[a]" -ac 2 -threads 0 [accessionNumber.mpeg]

### mp4

ffmpeg -i concat.mov -c:v mpeg4 -b:v 372k -pix\_fmt yuv420p -r 29.97 -vf "drawtext=fontfile='[/path/to/fontfile.ttf]': timecode='00\:00\:00\:00': r=29.97: x=(w-tw)/2: y=h-(2\*lh): fontcolor=white: fontsize=72: box=1: boxcolor=0x00000099,scale=420:270" -filter\_complex "[0:a:0][0:a:1]amerge=inputs=2[a]" -c:a aac -ar 44100 -ac 2 -map 0:v -map "[a]" -threads 0 [accessionNumber.mp4]

## ffmpeg stuff

### -map

You’ll see a lot of -map in these calls, -map helps us select streams and route them from input to output.

“-map 0” maps every input stream to the output file

“-map 0:a” “-map 0:v” maps every audio and video stream (respectively) from input to output

“-map 0:a:0” maps the first audio stream of the first input, “-map 1:a:0” maps the first audio stream of the second input

“-map -0:d” disables mapping data streams, so you can combine this with “-map 0” and map every stream except for the data (timecode) streams

### -c:v, -c:a

Stands for “codec: video” and “codec: audio” respectively. These set the codec for whichever stream

### -filter\_complex

This is how we downmix the multi-stream audio in the preservation mov into a stereo stream for the derivatives: -filter\_complex "[0:a:0][0:a:1]amerge=inputs=2[a]" -map “[a]”

So, we apply ffmpeg’s amerge (for audio merge) filter, we assign it a variable [a], and that variable is equal to the merge inputs of stream 0:a:0 and 0:a:1. the 2 preceding the [a] indicates that the channel configuration for the output file is stereo

### timecode burn-in

-vf "drawtext=fontfile='[/path/to/fontfile.ttf]': timecode='00\:00\:00\:00': r=29.97: x=(w-tw)/2: y=h-(2\*lh): fontcolor=white: fontsize=72: box=1: boxcolor=0x00000099,scale=420:270"

Here, we use ffmpeg’s video filter (-vf) drawtext function, set the font that we want to use, set the timecode format (we have to escape the : with a \ because it’s a special character), set the frame rate, set the x-y position of the burn-in, set the color and size, set weather or not we want a box surrounding the timecode, it’s size and color and opacity.

The scale at the end resizes the video and timecode together

### watermark

-i [/path/to/watermark.png] -filter\_complex "scale=320:180,overlay=0:0;[0:a:0][0:a:1]amerge=inputs=2[a]"

Here, we use the overlay filter component of -filter\_complex and just have to set the position, 0:0 is in the horizontal and vertical middle