mkpreview Documentation

Release 0.1.0

Colin Bitterfield

Jul 12, 2023

Contents:

CHAPTER 1

mkpreview

mkpreview builds a grid of images from a movie file. - Support for all video file types in FFMPEG. - It will create a SQLite3 database with all of the video meta dataparameters and create an MD5 hash of the file - It provides very basic support for creating video part numbers

1.1 Get Started!

Here's how to set up *mkpreview* for local environment.

1- Clone the *mkpreview* locally:

```
$ git clone git@github.com:/mkpreview.git
```

2- Install your local copy into a *virtualenv*. Assuming you have *virtualenvwrapper* installed, this is how you set up the package for local development:

```
$ sudo make boostrap
$ mkvirtualenv mkpreview
$ pip install -r requirements/dev.txt
```

3- How to enable/disable virtualenv

```
$ workon mkpreview
$ ...
$ deactivate
```

1.2 Credits

This package was generated using Yeoman and Cookiecutter projects.

CHAPTER 2

Installation

2.1 Stable release

To install mkpreview, run this command in your terminal:

```
$ pip install mkpreview
```

This is the preferred method to install mkpreview, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for mkpreview can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git@github.com:Studio-51/mkpreview.git
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Or you can use the follow command:

```
$ sudo make install    # to install
$ sudo make uninstall  # to uninstall
```

CHAPTER 3

Dev Notes

3.1 How to use python virtualenv

```
$ workon <virtualenv_name>
$ [<virtualenv_name>] ...
$ deactivate
```

3.2 How to run unit test

```
$ make utest
```

3.3 How to run sytem test

```
$ sudo make install
$ make stest
$ sudo make uninstall
```

3.4 How to run unit test coverage

```
$ make coverage
```

3.5 How to run flake8 and pylint

```
$ make flake
$ make pylint
```

3.6 How to build a docker image

```
$ make docker-image
```

CHAPTER 4

CLI Usage

4.1 Command Line Usage

usage: mkpreview.py [-h] [–version] [-v] [-dr] [-d] [-q] [-w TILE_WIDTH] [-r TILE_ROWS] [-c TILE_COLS] [-b TILE_BK_COLOR] [-p TILE_FG_COLOR] [-i IN_FILE] [-o OUT_DIR] [-m] [-s DBFILE] [-create-new-db] [-override OVERRIDE] [-colors] [-studio-id STUDIO_ID] [-part-id PART_ID] [-hwaccel {cuda,videotoolbox}]

This program will create a video preview file of a given video

optional arguments:

-h, --help show this help message and exit

--version show program's version number and exit

-v, --verbose Turn on verbose output

-dr, --dryrun Dryrun enabled no changes will occur

-d, --debug Turn on Debugging Mode

-q, --quiet No output is produced

-w TILE_WIDTH, --tile-width TILE_WIDTH Tile width - Tiles are sqaure

-r TILE_ROWS, --tile-rows TILE_ROWS Number of rows for a preview

-c TILE_COLS, --tile-cols TILE_COLS Number of columns for a preview

-b TILE_BK_COLOR, --tile-backgrond TILE_BK_COLOR Tile Background Color

-p TILE_FG_COLOR, --tile-foreground TILE_FG_COLOR Tile Pen Color

-fs FONT_SIZE, –font-size FONT_SIZE Font size for text default is 24pt

-i IN_FILE, --input IN_FILE Video input, can by a file or directory. If directory, it will not be recursive

-o OUT_DIR, --output-dir OUT_DIR Where to put the finished files

-m, --md5 Add the MD5 of the file to the filename

-s DBFILE, --store-db-file DBFILE Store the video information in SQLI3te file

-create-new-db Create a new database file

-override OVERRIDE save image with this filename override all other possible choices

-colors Display List of Available Colors

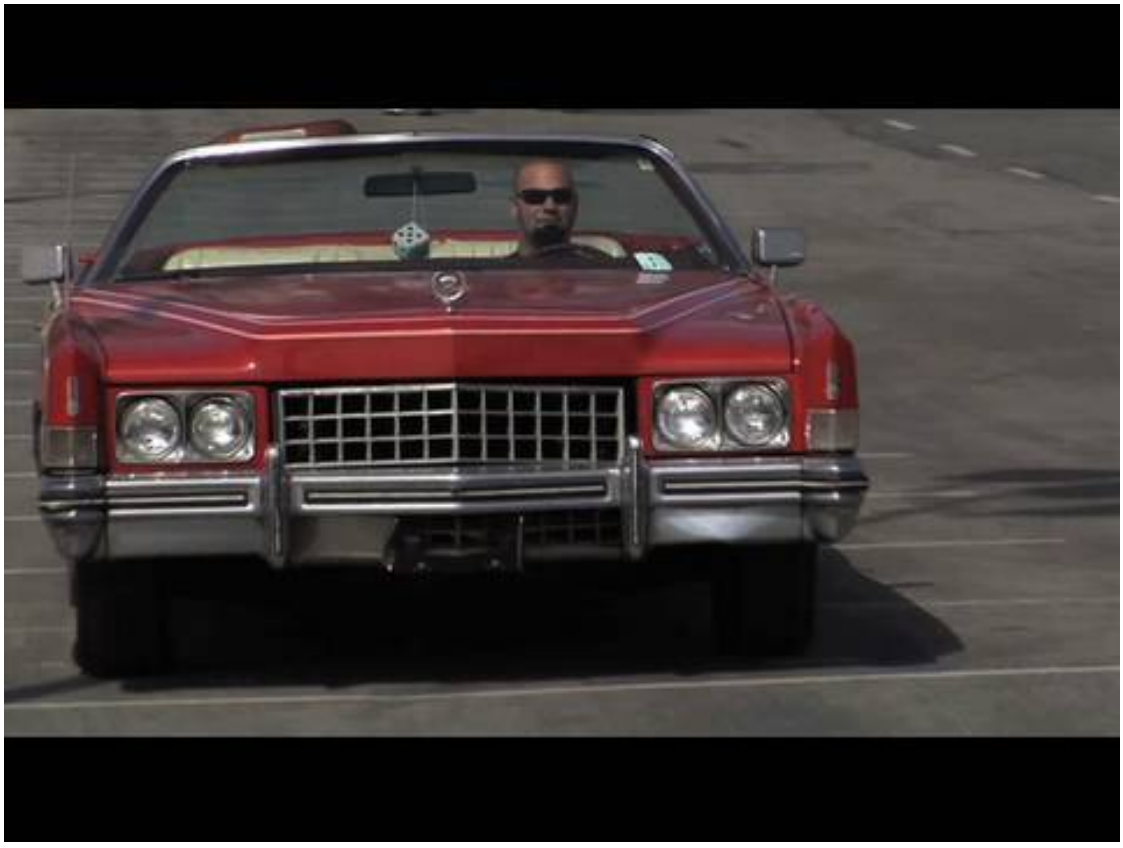
-studio-id STUDIO_ID Replace the this id, first part of filename for a part-id use in conjunction with -part-id

-part-id PART_ID Change first alpha part of filename for the part-id -hwaccel {cuda,videotoolbox}

Enable Hardware acceleration for videotoolbox or cuda

The filename of the output will be the part_id-md5-originalBaseName.png. If the part_id and md5 are unset the filename will be the original base name.png

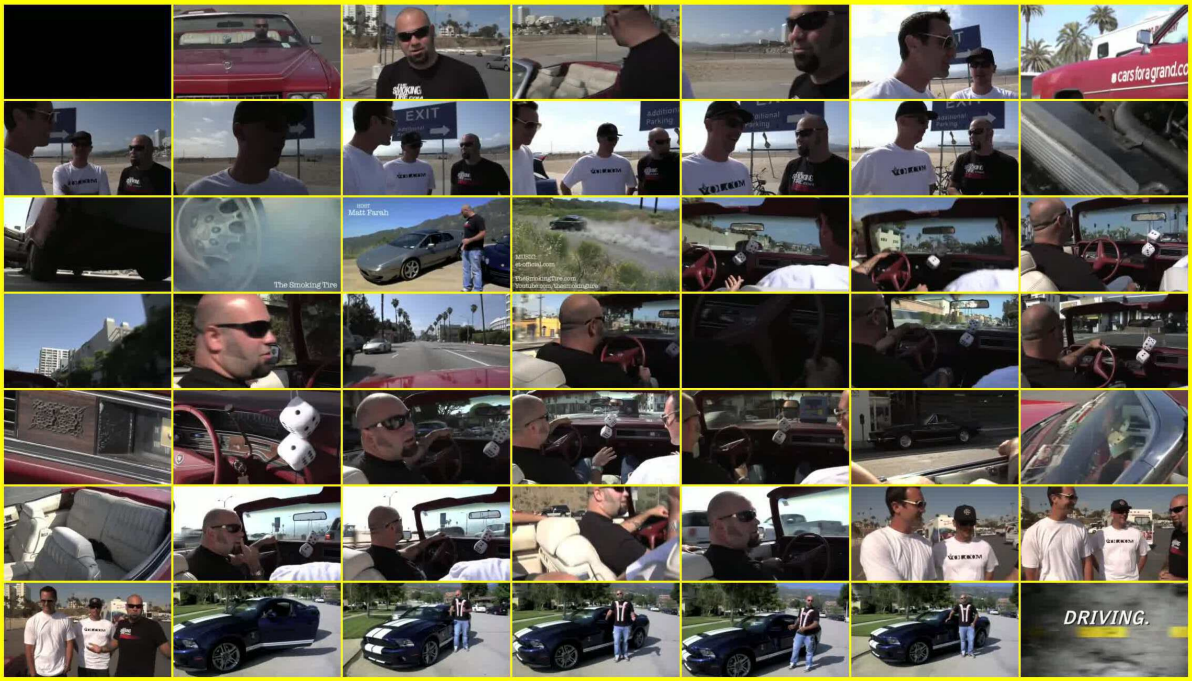
4.2 Example 1: sample



```
./mkpreview.py -i /Users/colinbitterfield/Downloads/WhatCarCanYouGetForAGrand.mp4 \
    --output-dir /tmp/ --md5 --tile-width 320 --tile-rows 7 \
    --tile-cols 7 --tile-background yellow \
    --tile-foreground blue --font-size 60 \
    --store-db-file /tmp/myDatabase.db \
    -create-new-db -override previewcard
```

Results:

WhatCarCanYouGetForAGrand.mp4
480 x 270 format 16:9
codec H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10
size 43.466005 Mibyte
runtime 0:09:27.378333 framerate 29



CHAPTER 5

mkpreview

5.1 mkpreview package

5.1.1 Submodules

5.1.2 mkpreview.config module

Created on Jan 9, 2020

@author: colin bitterfield

A list of common definitions

5.1.3 mkpreview.database module

A database class to help with managing SQLite databases

class mkpreview.database.**Database**(*database*, ***kwargs*)

Bases: object

Database class provides a higher level connector to SQLite3 It provides methods for:

Usage: myDB = Database('/tmp/filename.db')

Methods: create() : Creates a Database or tests existence of database

: Allows the database to be backed up if it exists.

name() : Returns the current database filename. Flag is true if the database exist connect() : Connects this class to the initialized database close() : Commits and Closes the current connection createTable() : Creates a table from a dictionary and adds primary indexes

: Allows overwrite (drop the data in the table)

insertORupdate(): Allows for an insert if not there and an update if there. : Uses a dictionary. Checks dictionary against columns to avoid errors

isTable() : Returns True if the tables exists backupTable() : Makes a copy of the table and renames it with a current timestamp dropTable() : Remove all of the data from the table sqlExecute() : Executes an arbitrary SQL Script against the open database. deleteData() : removes database based on table and where.

Variables: global DEBUG : Used for output global QUIET : User for output local message : Return Message for all methods local data(dict): A dictionary for method use. local flag : True / False for results of method self.database : database name for all methods self.method : calling method name self.class : name of the class being called self.db_connect : name of the connection

Returns: All methods return a [True | False] and a message If bad values are passed, it will fast fail and return a message. No further evaluations are made.

backup_table(***kwargs*)

close(***kwargs*)
Close the database connection

commit(***kwargs*)
Close the database connection

connect(***kwargs*)
connect class to database

create(***kwargs*)
overwrite = [True | False] backup = [True | False]

If database exists, with overwrite FALSE and backup FALSE, method exists with False If database does not exist it is created If database exists and overwrite without backup, it is deleted If database exists and backup is set, database will be backed up; regardless of overwrite.

FLAG = True if a database is created, False if a database is not created. MESSAGE = database name + actions taken

create_table(***kwargs*)
create a table, optional drop if exists (default) If overwrite set to false, table will error if it exists

data = {}

delete_data(***kwargs*)

flag = False

insert_update(***kwargs*)
Insert a row or update the row with the field names in a dictionary parameters: ———— table = Table Name key_field = Field for initial update and where value key_value = Where Value data = Data to update

istable(***kwargs*)
Test if table exists; if so return true

message = ''

name(***kwargs*)
None

FLAG = True if the file exists / False if not MESSAGE = database name

sql_exec(***kwargs*)
SQL : SQL Statement (no expansion)

flag : True / False results : SQL Results

truncate_table(**kwargs)
truncates the table using delete table='Table Name'

5.1.4 mkpreview.mkpreview module

5.1.5 mkpreview.version module

version string

5.1.6 Module contents

__init__ for mkpreview.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at git@github.com:Studio-51/mkpreview.git/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

6.1.4 Write Documentation

Python Boilerplate could always use more documentation, whether as part of the official Python Boilerplate docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at github repo.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *python_boilerplate* for local development.

1. Fork the *python_boilerplate* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/python_boilerplate.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv python_boilerplate
$ cd python_boilerplate/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 python_boilerplate tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/fgriberi/python_boilerplate/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_python_boilerplate
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- Colin Bitterfield <colin@bitterfield.com>

7.2 Contributors

None yet. Why not be the first?

Changelog

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.0.0/), and this project adheres to [Semantic Versioning](https://semver.org/spec/v2.0.0.html).

[Unrelease]

Added

- added configuration information
- fixed production requirements

[Unrelease] TBD-link

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index