steve Documentation

Release 0.5.dev

Will Kahn-Greene

August 05, 2016

Contents

steve is a command line interface and Python library to make it easier and funner to add collections of videos to a richard instance than using the Django admin.

Wait, wut?

It's a bunch of command line subcommands that automate building a video collection with the basic data and then a library of Python functions that make it easy to write a script or two to batch-process a large number of video metadata files.

CHAPTER 1

User/Contributor Guide

1.1 About steve

richard is a video index site. It has a very basic admin interface for adding videos by hand one-by-one. This gets very tedious when adding all the videos for a conference.

steve is a command line utility for downloading information for a conference, downloading all the metadata for the videos, and making it easier to transform the data, fix it and make it better. Then steve grew into a library of functions making it easier to manipulate video metadata and push it into richard.

richard and steve go together like peanut butter and jelly. You could use one without the other, but it's daft. steve uses richard's API for pulling/pushing video data.

It solves this use case:

> MAL sends Will a request to add the EuroPython 2011 videos to pyvideo.org. The EuroPython 2011 videos are on YouTube. Will uses steve to download all the data for the conference on YouTube, then uses steve to apply some transforms on the data, then uses steve to edit each video individually and finally uses steve to push all the data (the new conference, new videos, speakers, tags) to pyvideo.org.

1.1.1 History

I've been working on richard since March 2012. I knew I needed steve and had some thoughts on how it should work, but there were a bunch of things I wanted to do with richard, so I pushed work on steve off.

Then on May 29th, 2012, I finished up the initial bits of steve and thus steve was born.

I worked on it on and off while simultaneously working on adding the EuroPython 2011 conference videos to pyvideo.org and helping Carl with some of the conferences he was working on.

Then I continued working on it until I thought it was at the point where someone else could use it to generate conference metadata. That point happened on January 13th, 2013 when I released version 0.1.

1.2 ChangeLog

1.2.1 version 0.5 – in development

Changes

1.2.2 version 0.4 – August 5th, 2014

Changes

- **added steve.richardapi.get_video**
- **Tweaks to webedit to make it a little easier to use**
- **added steve.utils.fetch_videos_from_url**

1.2.3 version 0.3 – March 17th, 2014

API Changes

- **steve.richardapi functions changed signatures**

  The new functions match the new v2 richard api. These involve sending just an auth token—no more sending an auth token and a username.

- **steve.richardapi.MissingRequiredData changed**

  If you were missing data, you could look at the `errors` attribute of the `MissingDataRequired` exception for details. That was dumb and difficult. Now it's just part of the message. So you can do this:

```
try:
    # do something wrogn
except MissingRequiredData as exc:
    print exc
```

- **nixed steve.richardapi.create_category_if_missing**

  I think this was causing too many problems. You now have to create the category on the server.

Other Changes

- **bug fixes**

1.2.4 version 0.2 – March 17th, 2013

API Changes

- **changed steve.util.verify_json to steve.util.verify_video_data**

  The previous function name was a misnomer—it takes a Python dict as an argument and has nothing to do with JSON.

Other Changes

- **added steve.restapi**

  Basic REST client to talk to richard instances.

- **added steve.richardapi**

  Added the richardapi module which has some functions that make it easier to do richard API things

- **added steve.util.html_to_markdown**

  Since richard and pyvideo now use Markdown for summaries and descriptions, it helps to have a converter.

- **added webedit command of awesome**

  steve now has a webedit command that lets you edit JSON files in a web-based editor.

1.2.5 version 0.1 – January 13th, 2013

- **First version, so everything is new.**

  See documentation for features and how to use it!

1.3 License

1.4 Installation

There are a few ways to install steve. This document covers them.

1.4.1 Installing a released version

You can install steve using pip:

```
pip install steve
```

1.4.2 Installing a bleeding edge version

If you want a bleeding edge version of steve, you can either install with pip from a git url or clone the project and install that.

pip and git urls:

Install like this:

1. `pip install git+https://github.com/pyvideo/steve.git`

Update like this:

1. `pip install -U git+https://github.com/pyvideo/steve.git`

git clone and installing from that:

Install like this:

1. `git clone git://github.com/pyvideo/steve.git`
2. `cd steve`
3. `python setup.py develop`

Update like this:

1. `cd steve`
2. `git checkout master`
3. `git pull --rebase`

1.4.3 Installing a Bleeding edge for hacking purposes

If you want to install steve in a way that makes it easy to hack on, do this:

1. `git clone git://github.com/pyvideo/steve.git`
2. `cd steve`
3. `virtualenv ./venv/`
4. `./venv/bin/python setup.py develop`

When you want to use steve from your virtual environment, make sure to activate the virtual environment first. e.g.:

1. `. ./venv/bin/activate`
2. `steve-cmd --help`

1.5 Using steve - commandline

The `steve-cmd` utility is designed to automate the basic tasks that result in you having a directory of JSON files each containing the metadata for a single video.

After doinge some `steve-cmd` work, you can then tar/zip this directory up and send it to someone who has API access to the richard instance you're collecting videos for. This person can then look at the data and push it.

In this way, anyone can generate the metadata for a richard instance from the comfort of their own command line.

1.5.1 Usage

For list of subcommands, arguments and other help, do this:

```
steve-cmd --help
```

The basic commands are these:

createproject

Creates the directory structure and configuration files for a new steve project. Each project is a collection of videos from a single source.

fetch

Fetches the metadata for the videos from the url where all the videos are hosted and puts it in JSON files in the `json/` directory of your steve project.

status

Tells you the editing status of all the JSON files.

verify

Goes through the JSON files and verifies correctness of the keys and values. Are the required data elements present? Are the values of the correct type? Are there any "bad" values?

webedit

Provides a (really super duper) basic web server app that lets you go through the JSON files in your web browser.

Also, there are some other subcommands:

push

Pushes a bunch of JSON files to a richard instance.

pull

Pulls a bunch of data from a richard instance and puts it in JSON files.

scrapevideo

This is a convenience subcommand for scraping a single video at a url and showing the metadata.

1.5.2 Example use

Note: This is a quick tutorial—you don't have to use steve like this. Use it in a way that makes your work easier!

1. Install steve.
2. Run: `steve-cmd createproject europython2011`

   This creates a `europython2011` directory for project files.

   I usually call this the project directory.

   In that directory is:

   (a) a `steve.ini` project config file

   (b) a `json/` directory which hold the video metadata json files

   I usually have all my helper scripts in the project directory since it has the `steve.ini` file.
3. `cd europython2011`
4. Edit `steve.ini`:

```
[project]

# The name of this group of videos. For example, if this was a
# conference called EuroPython 2011, then you'd put:
# category = EuroPython 2011
category = EuroPython 2011

# The url for where all the videos are listed.
# e.g. url = http://www.youtube.com/user/PythonItalia/videos
url = http://www.youtube.com/user/PythonItalia/videos

# If the url is a YouTube-based url, you can either have 'object'
# based embed code or 'iframe' based embed code. Specify that
# here.
youtube_embed = object

# The url for the richard instance api.
# e.g. url = http://example.com/api/v1/
# api_url =

# Your username and api key.
#
# Alternatively, you can pass this on the command line or put it in a
# separate API_KEY file which you can keep out of version control.
# e.g. username = willkg
#      api_key = OU812
# username =
# api_key =
```

If you're not pushing the JSON files to a richard instance, you can ignore the `api_url`, `username` and `api_key` keys.

5. Run: `steve-cmd fetch`

   This fetches the video metadata from that YouTube user and generates a series of JSON files—one for each video—and puts them in the `json/` directory.

   The format for each file matches the format expected by the richard API.

6. See the status of your video metadata.

   Run: `steve-cmd status`

   Lists filenames for all videos that have a non-empty whiteboard field. Because you've just downloaded the metadata, all of the videos have a whiteboard field stating they haven't been edited, yet.

   Run: `steve-cmd ls`

   Lists titles and some other data for each video in the set.

7. Now you go through and edit the json metadata. There are a few ways to do this. **Don't** just pick one way—mix and match them to reduce the work required.

   Use the *whiteboard* field to keep track of which videos still have problems and/or things that need to be done with them and/or just haven't been edited, yet.

   (a) **Edit with your favorite editor.**

   You can use the `status` command to make this easier.

   For example, if you use vim:

```
steve-cmd status --list | xargs vim
```

and edit them by hand one-by-one.

(b) **Write a script to batch-process the files.**

You can also write a script which uses functions in `steve.util` to automate fixing the metadata.

For example, here's a script that takes the summary data, converts it from reStructuredText to HTML and puts it in the description field:

```
from docutils.core import publish_parts

from steve.util import (get_project_config, load_json_files,
    save_json_files)


cfg = get_project_config()
data = load_json_files(cfg)


def parse(text):
    settings = {
        'initial_header_level': 2,
        'transform_doctitle': 1
        }
    parts = publish_parts(
        text, writer_name='html', settings_overrides=settings)
    return parts['body']


for fn, contents in data:
    print fn

    summary = contents['summary'].strip()
    summary_parsed = parse(summary)
    if 'ERROR' in summary_parsed or 'WARNING' in summary_parsed:
        print 'problem with %s' % fn
        raise ValueError()

    if not contents['description']:
        contents['description'] = parse(summary)


save_json_files(cfg, data)
```

Conference data varies pretty widely, so writing scripts to batch-process it to handle issues like this is super helpful. Automate anything you can.

See the API documentation in *Using steve - library*.

(c) **Use the web editor.**

steve comes with a bare-bones web-based editor for the json files. To launch it from the project directory, do:

```
steve-cmd webedit
```

then point your browser at the url in the output.

This is helpful when you have a few things to fix and don't feel like writing json.

If there are other tools you want to use—go for it. Anything to get the job done.

8. Run: `steve-cmd verify`

   This goes through all the json files and verifies correctness.

   Is the data of the correct type and shape?

   Are required fields present?

   Are values that should be in HTML in HTML?

9. Now it's time to submit your changes!

   If you do not have an API key that gives you write access to the server, then tar the `json/` directory up and send it to someone who does.

   If you do have an API key that gives you write access to the server, then you can do:

```
steve-cmd push
```

   That will create the videos on the server and update the JSON files with the new ids.

That's it!

Note: Use version control for your steve project and commit changes to it. Make sure you back it up, too! Don't lose everything you've done because you wrote a bad batch-processing script!

1.6 Using steve - library

By day, steve is a cli of world renoun. By night, steve is a Python library capable of great cunning. This chapter covers the utility functions.

1.6.1 Writing steve scripts

steve can be used for batch processing a bunch of JSON files.

Most batch processing works this way:

1. get the config file (*`steve.util.get_project_config()`*)
2. get all the json files (*`steve.util.load_json_files()`*)
3. iterate through the json files transforming the data (Python for loop)
4. save the json files (*`steve.util.save_json_files()`*)

1.6.2 steve.util

steve.util.**with_config**(*fun*)

Decorator that passes config as first argument

Raises **ConfigNotFound** – if the config file can't be found

This calls *get_project_config()*. If that returns a configuration object, then this passes that as the first argument to the decorated function. If *get_project_config()* doesn't return a config object, then this raises ConfigNotFound.

Example:

```
>>> @with_config
... def config_printer(cfg):
...     print 'Config!: {0!r}'.format(cfg)
...
>>> config_printer()  # if it found a config
Config! ...
>>> config_printer()  # if it didn't find a config
Traceback
    ...
steve.util.ConfigNotFound: steve.ini could not be found.
```

steve.util.**get_project_config**()

Finds and opens the config file in the current directory

Raises **ConfigNotFound** – if the config file can't be found

Returns config file

steve.util.**html_to_markdown**(*text*)

Converts an HTML string to equivalent Markdown

Parameters **text** – the HTML string to convert

Returns Markdown string

Example:

```
>>> html_to_markdown('<p>this is <b>html</b>!</p>')
u'this is **html**!'
```

steve.util.**load_json_files**(*config*)

Parses and returns all video files for a project

Parameters **config** – the configuration object

Returns list of (filename, data) tuples where filename is the string for the json file and data is a Python dict of metadata.

steve.util.**save_json_files**(*config*, *data*, ***kw*)

Saves a bunch of files to json format

Parameters

- **config** – the configuration object
- **data** – list of (filename, data) tuples where filename is the string for the json file and data is a Python dict of metadata

Note: This is the *save* side of `load_json_files()`. The output of that function is the *data* argument for this one.

steve.util.**save_json_file**(*config*, *filename*, *contents*, ***kw*)
Saves a single json file

Parameters

- **config** – configuration object
- **filename** – filename
- **contents** – python dict to save
- **kw** – any keyword arguments accepted by *json.dump*

steve.util.**scrapevideo**(*video_url*)
Scrapes the url and fixes the data

Parameters **video_url** – Url of video to scrape.

Returns Python dict of metadata

Example:

```
>>> scrapevideo('http://www.youtube.com/watch?v=ywToByBkOTc')
{'url': 'http://www.youtube.com/watch?v=ywToByBkOTc', ...}
```

steve.util.**verify_video_data**(*data*)
Verify the data in a single json file for a video.

Parameters

- **data** – The parsed contents of a JSON file. This should be a Python dict.
- **category** – The category as specified in the steve.ini file.

  If the steve.ini has a category, then every data file either has to have the same category or no category at all.

  This is None if no category is specified in which case every data file has to have a category.

Returns list of error strings.

steve.util.**verify_json_files**(*json_files*)
Verifies the data in a bunch of json files.

Prints the output

Parameters

- **json_files** – list of (filename, parsed json data) tuples to call `verify_video_data()` on
- **category** – The category as specified in the steve.ini file.

  If the steve.ini has a category, then every data file either has to have the same category or no category at all.

  This is None if no category is specified in which case every data file has to have a category.

Returns dict mapping filenames to list of error strings

1.6.3 Recipes

Here's some sample code for doing batch transforms. Each script should be located in the project directory root next to the `steve.ini` file. Make sure the steve package is installed and then run the script with the python interpreter:

```
python name_of_my_script.py
```

Or however you want to structure and/or run it.

Update language

This fixes the *language* property in each json file. It sets it to "Italian" if the word "Italiana" appears in the summary. Otherwise it sets it to "English".

```
import steve.util

cfg = steve.util.get_project_config()
data = steve.util.load_json_files(cfg)

for fn, contents in data:
    print fn

    # If 'Italiana' shows up in the summary, set the language
    # to Italian.
    if 'Italiana' in contents['summary']:
        contents['language'] = u'Italian'
    else:
        contents['language'] = u'English'

steve.util.save_json_files(cfg, data)
```

Move speaker from summary to speakers

This removes the first line of the summary and puts it in the speakers field.

```
import steve.util

cfg = steve.util.get_project_config()
data = steve.util.load_json_files(cfg)

for fn, contents in data:
    print fn

    # If the data already has speakers, then we assume we've already
    # operated on it and don't operate on it again.
    if contents['speakers']:
        continue

    summary = contents['summary']
    summary = summary.split('\n')

    # The speakers field is a list of strings. So we remove the first
    # line of the summary, strip the whitespace from it, and put that
    # in the speakers field.
    # (NB: This bombs out if the summary field is empty.)
    contents['speakers'].append(summary.pop(0).strip())
```

```
    # Put the rest of the summary back.
    contents['summary'] = '\n'.join(summary)

steve.util.save_json_files(cfg, data)
```

Convert summary and description to Markdown

This converts summary and description to Markdown.

```
import steve.util

cfg = steve.util.get_project_config()
data = steve.util.load_json_files(cfg)

for fn, contents in data:
    print fn

    contents['summary'] = steve.util.html_to_markdown(
        contents.get('summary', ''))

    contents['description'] = steve.util.html_to_markdown(
        contents.get('description', ''))

steve.util.save_json_files(cfg, data)
```

1.7 Using steve - restapi

"steve comes with its own REST client API."

"Seriously? There are dozens out there? Why roll your own?"

"Because the one I was using had issues and wasn't being updated. I decided it was easier to just roll my own for my limited needs. Plus it was kind of fun to write."

"Dude. steve is turning into a Frankenstein monster monstrosity. You need to see the doctor to cure you of your NIH syndrome."

"Shh... I'm busy."

1.7.1 Using the REST client API by itself

It's similar to slumber except a little less feature(bug)-full. The gist of it is this:

1. Import some stuff:

```
    from steve.restapi import API, RestAPIException, get_content
```

2. Build an *API* object:

```
api = API('http://localhost/v1/api/')
```

3. Use the *API* object to fiddle with resources:

```
# Get all Foos
all_foos = api.foo.get()

# Get foo with id 1
foo_1 = get_content(api.foo(1).get())

# Change the data, then put it
foo_1['somekey'] = 'newvalue'
api.foo(1).put(data=foo_1)

# Create a new foo. This does a POST and if there's
# a 201, it'll return the results of that.
newfoo = get_content(api.foo.post(data={'somekey': 'newvalue'}))
```

That's pretty much it!

Why *get_content*? That way you're guaranteed that you have the requests *Response* object so you can see what's going on. That makes this REST client API a bit easier to debug—it's just a thin layer on top of requests.

1.7.2 steve.restapi

This is a REST client API since steve does a bunch of REST things with richard's API. It's a slim layer on top of requests.

steve.restapi.**get_content**(*resp*)
Returns the JSON content from a response.

Note: Mostly this just deals with the fact that requests changed *.json* from a property to a method. Once that settles out and we can use requests >= 1.0, then we can ditch this.

---

class steve.restapi.**API**(*base_url*)
Convenience wrapper around requests.

Example:

```
from steve.restapi import API

# Creates an api endpoint
api = API('http://pyvideo.org/v1/api/')

# Does a get for all videos
all_videos = api.video.get()

# Does a get for video with a specific id
video_1 = api.video(1).get()

# Update the data and then put it
video_1['somekey'] = 'newvalue'
api.video(1).put(data=video_1)

# Create a new video. This does a POST and if there's a
# redirect, will pick that up.
newvideo = api.video.post(data={'somekey': 'newvalue'})
```

class steve.restapi.**Resource**(***kwargs*)
Convenience wrapper for requests.request.

HTTP methods return requests Response objects or throw exceptions in cases where things are weird.

class steve.restapi.**RestAPIException**(**args*, ***kwargs*)

class steve.restapi.**Http4xxException**(**args*, ***kwargs*)
Exception for 4xx errors.

These usually mean you did something wrong.

Property response The full requests *Response* object.

Example:

```
from steve.restapi import Http4xxException

try:
    # do something here
except Http4xxException as exc:
    # oh noes! i did something wrogn!

    # This tells you the actual HTTP status code
    print exc.response.status_code

    # This tells you the content of the response---sometimes
    # the server will tell you an error message and it's
    # probably in here.
    print exc.response.content
```

class steve.restapi.**Http5xxException**(**args*, ***kwargs*)
Exception for 5xx errors.

These usually mean the server did something wrong. Let me know.

Property response The full requests *Response* object.

Example:

```
from steve.restapi import Http5xxException

try:
    # do something here
except Http5xxException as exc:
    # oh noes! i hit dumb willkg code and server is br0ken!

    # This tells you the actual HTTP status code
    print exc.response.status_code

    # This tells you the content of the response---sometimes
    # the server will tell you an error message and it's
    # probably in here.
    print exc.response.content
```

1.8 Using steve - richardapi

This module holds a series of functions that use the richard API to move data back and forth.

- *steve.richardapi*

1.8.1 steve.richardapi

New in version 0.2.

Note: Carl, Ryan: These functions are for you!

They use an auth token. If you need an auth token, let one of the pyvideo admin know.

steve.richardapi.**get_all_categories**(*api_url*)
Given an api_url, retrieves all categories

Parameters **api_url** – URL for the api.

Returns list of dicts each belonging to a category

Raises ***steve.restapi.Http5xxException*** – if there's a server error

Example:

```
from steve.util import get_all_categories

cats = get_all_categories('http://pyvideo.org/api/v1/')
print [cat['title'] for cat in cats]

# Prints something like:
# [u'PyCon 2012', u'PyCon 2011', etc.]
```

steve.richardapi.**get_category**(*api_url*, *title*)
Gets information for specified category

Parameters

- **api_url** – URL for the api
- **title** – title of category to retrieve

Returns category data

Raises **steve.richardapi.DoesNotExist** – if the category doesn't exist

steve.richardapi.**get_video**(*api_url*, *auth_token*, *video_id*)
Gets information for specified video

Parameters

- **api_url** – URL for the api
- **auth_token** – auth token
- **video_id** – The id for the video

Returns video data

Raises **steve.richardapi.DoesNotExist** – if the video doesn't exist

steve.richardapi.**create_video**(*api_url*, *auth_token*, *video_data*)
Creates a video on the site

This creates a video on the site using HTTP POST. It returns the video data it posted which also contains the id.

Note: This doesn't yet check to see if the video already exists.

Parameters

- **api_url** – URL for the api
- **auth_token** – auth token
- **video_data** – Python dict holding the values to create this video

Returns the video data

Raises

- ***steve.restapi.Http5xxException*** – if there's a server error
- **steve.richardapi.MissingRequiredData** – if the video_data is missing keys that are required

Example:

```
import datetime

from steve.util import STATE_LIVE, create_video, MissingRequiredData

try:
    video = create_video(
        'http://pyvideo.org/api/v1/',
        auth_token='ou812authkey',
        video_data={
            'category': 'Test Category',
            'state': STATE_LIVE,
            'title': 'Test video title',
            'speakers': ['Jimmy Discotheque'],
            'language': 'English',
            'added': datetime.datetime.now().isoformat()
        })

    # Prints the video data.
    print video

except MissingRequiredData as exc:
    # Prints the errors
    print exc
```

Note: Check the richard project in the video app at `models.py` for up-to-date list of fields and their types.

https://github.com/pyvideo/richard/blob/master/richard/videos/models.py

steve.richardapi.**update_video**(*api_url*, *auth_token*, *video_id*, *video_data*)

Updates an existing video on the site

This updates an existing video on the site using HTTP PUT. It returns the final video data.

Warning: This stomps on the data that's currently there. If you have the video_id wrong, then this will overwrite the current data.
Be very careful about updating existing video data. Best to get it, make sure the id is correct (check the title? the slug?), and then update it.

Parameters

- **api_url** – URL for the api
- **auth_token** – auth token
- **video_id** – The id for the video
- **video_data** – Python dict holding all the data for this video

Returns the updated video data

Raises

- ***steve.restapi.Http4xxException*** – if the video doesn't exist on the server
- ***steve.restapi.Http5xxException*** – if there's a server error
- **steve.richardapi.MissingRequiredData** – if the video_data is missing keys that are required

Example:

```
import datetime

from steve.util import STATE_LIVE, update_video, MissingRequiredData

try:
    video = update_video(
        'http://pyvideo.org/api/v1/',
        auth_token='ou812authkey',
        video_id=1101,
        video_data={
            'id': 1101,
            'category': 'Test Category',
            'state': STATE_LIVE,
            'title': 'Test video title',
            'speakers': ['Jimmy Discotheque'],
            'language': 'English',
            'added': datetime.datetime.now().isoformat()
        })

    # Prints the video data.
    print video

except MissingRequiredData as exc:
    # Prints the errors
    print exc
```

Note: Check the richard project in the video app at `models.py` for up-to-date list of fields and their types.

https://github.com/pyvideo/richard/blob/master/richard/videos/models.py

1.9 Hacking on steve

It's likely steve will never "be done". Thus, it's likely it will be in a perpetual need for people to tweak steve to do the things they need it to do. These people are you!

This chapter covers contributing to steve.

1.9.1 Contributing

We use Github to host the code. After you've forked the project, make changes like this:

1. create a branch based on master to hold your changes
2. make your changes in that branch and commit them
3. create a pull request between pyvideo/master and your branch with all the details you think I'll need to know to understand what you did, why, and what problem you were trying to solve

This is somewhat high level and sort of assumes you know git, Github, and contributing to projects like this one. If you need more help because these assumptions don't match you, please ask me on IRC.

1.9.2 Code conventions

PEP-8 and pyflakes is your friend.

1.9.3 Documenting

steve documentation is in two places:

1. in the code in docstrings
2. in the `docs/` directory in reStructuredText files as a Sphinx docs project

Everything is in reStructuredText.

Generally speaking:

1. Good docs are good.
2. Bad docs are lousy.
3. Lack of docs are suboptimal.

1.9.4 Running and writing tests

steve comes with unit tests. Unit tests are executed using nose. If you don't already have nose installed, then install it with:

```
pip install nose
```

I like to use nose-progressive, too, because it's awesome. To install that:

```
pip install nose-progressive
```

To run the unit tests from a git clone or the source tarball, do this from the project directory:

```
nosetests
```

With nose-progressive and fail-fast:

```
nosetests -x --with-progressive
```

1.10 Resources I found helpful

1.10.1 richard api docs

- richard api docs

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

S

A

C

G

H

L

R

S

U

V

W