Savify Documentation

Release 2.3.4

Laurence Rawlings

Jan 29, 2021

CONTENTS:

CHAPTER

ONE

SAVIFY

1.1 Savify

Savify is a python library that downloads songs from a selected provider (by default YouTube), and then scrapes the meta information from Spotify. Given a query, Savify will find and download songs to mp3 format with quality as high as **320 kb/s**! The application will also scrape and write **id3v2 tags** to all your songs. Tags include **title, artists, year, album and even cover-art!**

Savify supports all Spotify track, album, and playlist links. Additionally, there is an **integrated search function** so even if you do not have the Spotify link you can simply enter song name and Savify will download it!

As well as mp3, Savify can also download and convert to other file types. Inside the application, you can specify which format and quality you would like to download the song in for maximum compatibility across all your devices. Available formats: mp3, aac, flac, m4a, opus, vorbis, and wav. **Tags and cover art will only be applied to songs downloaded in mp3 format.**

Please note this library does not go against Spotify TOS in any way, songs are not ripped directly from Spotify, but are instead downloaded from other sources such as YouTube and Soundcloud using the youtube-dl python library. Spotify is only used to gather accurate meta information to be embedded into the downloaded song files.

Any questions or feedback join the Discord Server

1.1.1 FFmpeg

Savify relies on the open source FFmpeg library to convert and write metadata to the songs it downloads. Please make sure FFmpeg is installed on your computer and added to the System PATH. Follow the tutorial here.

1.2 Playlists

If you want to use Savify to download personal Spotify playlists, ensure their visibility is set to 'Public'. This is so Savify can use the Spotify API to retrieve the song details from your playlist.

1.3 Installation

If you are on Windows you can download the latest pre-packed executable package (which I recommend as you will not have to provide a Savify API key), or you can download the python library and run the module directly using the CLI.

1.3.1 Download the latest release

Go here to download the latest Savify.exe then make sure you have:

- FFmpeg downloaded and it added to your Path
- Spotify API credentials added to your environment variables

That is it, you should be good to go! See some usage examples below.

1.3.2 Using the Python module

```
$ pip install -U savify
```

1.4 Usage

Currently Savify only supports Spotify URLs and search queries, however support for Spotify URIs will be added in the future.

1.4.1 CLI

If you have downloaded the latest Savify.exe from the releases page open your terminal and navigate to the same directory as the binary, then you can run:

```
$ Savify.exe
```

If you are using the Python package and savify is installed to your site-packages and your pip folder is in your PATH (which it should be by default), from anywhere you can simply run:

```
$ savify
```

For help run:

```
$ savify --help
```

General usage

Using the default above:

```
$ savify "https://open.spotify.com/track/4Dju9g4NCz0LDxwcjonSvI"
```

Specifying your own options:

```
$ savify "https://open.spotify.com/track/4Dju9g4NCz0LDxwcjonSvI" -q best -f
mp3 -o "/path/to/downloads" -g "%artist%/%album%"
```

With a search query:

```
$ savify "You & I - Bru-C" -t track -q best -f mp3 -o "/path/to/downloads" -g
"%artist%/%album%"
```

Grouping

Available variables: `%artist%, %album%, %playlist%`

For example:

```
$ savify "You & I - Bru-C" -o /path/to/downloads -g "%artist%/%album%"
```

Would download in the following directory structure:

```
/path/to/downloads
     |
     |- /Bru-C
          |
          |- /Original Sounds
               |
               |- Bru-C - You & I.mp3
```

1.4.2 Download Defaults

Query Type track

Quality best

Format mp3

Path Windows: HOME/AppData/Roaming/Savify/downloads

Linux: HOME/.local/share/Savify/downloads

MacOS: HOME/Library/Application Support/Savify/downloads

Grouping no grouping

For more usage examples read the docs.

1.4.3 Spotify Application

To use the Savify Python module you will need your own Spotify developer application to access their API. To do this sign up here. When you have made a new application take note of your client id and secret. You can pass the id and secret to Savify in two ways:

Environment variables (recommended)

Now you need to add 2 environment variables to your system:

`SPOTIPY_CLIENT_ID`

`SPOTIPY_CLIENT_SECRET`

To find out how to do this find a tutorial online for your specific operating system. Once you have done this make sure to restart your shell.

During object instantiation

You can pass in your id and secret using a tuple when creating your Savify object:

```
s = Savify(api_credentials=("CLIENT_ID","CLIENT_SECRET"))
```

1.4.4 Use in your Python project

Install the package to your environment:

```
$ pip install savify
```

Import and use Savify:

```
from savify import Savify
from savify.types import Type, Format, Quality

s = Savify()
# Spotify URL
s.download("SPOTIFY URL")

# Search Query
# Types: TRACK, ALBUM, PLAYLIST
s.download("QUERY", query_type=Type.TRACK)
```

Savify optional constructor arguments (see above for defaults):

```
import logging

 from savify import Savify
 from savify.types import Type, Format, Quality
 from savify.utils import PathHolder

 # Quality Options: WORST, Q32K, Q96K, Q128K, Q192K, Q256K, Q320K, BEST
 # Format Options: MP3, AAC, FLAC, M4A, OPUS, VORBIS, WAV
 Savify(api_credentials=None, quality=Quality.BEST, download_format=Format.MP3, path_
↪holder=PathHolder(downloads_path='path/for/downloads'), group='%artist%/%album%',␣
↪quiet=False, skip_cover_art=False, log_level=logging.INFO)
```

Manually customising youtube-dl options:

```
from savify import Savify

options = {
    'cookiefile': 'cookies.txt'
}

Savify(ydl_options=options)
```

Passing in your own logger:

```
from savify import Savify
from savify.logger import Logger

logger = Logger(log_location='path/for/logs', log_level=None) # Silent output

Savify(logger=logger)
```

The group argument is used to sort you downloaded songs inside the output path. Possible variables for the path string are: %artist%, %album%, and %playlist%. The variables are replaced with the songs metadata. For example, a song downloaded with the above Savify object would save to a path like this: *path/for/downloads/Example Artist/Example Album/Example Song.mp3*

1.5 For Developers

If you want to try your hand at adding to Savify use the instructions here. From there you can make any additions you think would make Savify better.

1.5.1 Tip

If you are developing Savify, install the pip package locally so you can make and test your changes. From the root directory run:

```
$ pip install -e .
```

You can then run the Python module:

```
$ savify
```

1.6 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install Savify, run this command in your terminal:

```
$ pip install savify
```

This is the preferred method to install Savify, 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 Savify can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/LaurenceRawlings/savify
```

Or download the tarball:

```
$ curl -OJL https://github.com/LaurenceRawlings/savify/tarball/master
```

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

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

CHAPTER

THREE

USAGE

To use Savify in a project:

```
import savify
```

CHAPTER

FOUR

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/LaurenceRawlings/savify/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/LaurenceRawlings/savify/issues.

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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/savify.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 savify
$ cd savify/
$ 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 savify tests
$ python setup.py test or pytest
$ 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.

4.3 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 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/LaurenceRawlings/savify/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ pytest tests.test_savify
```

4.5 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:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- Laurence Rawlings <contact@laurencerawlings.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER

SIX

HISTORY

6.1 0.1.0 (2020-11-04)

- First release on PyPI.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search