Introduction

Metadata is a useful way to describe specific details about a particular file or series of files. Having good metadata is important to easily find what you're looking for in your system. However, adding, removing and even editing a file's metadata is not always an easy task and in some cases, it's not possible. MyTag is an easy-to-use application-oriented programming language that allows the user to explore their file's metadata and edit in any way they desire. Whether you need to divide a book by chapters, add cover artwork to a music album, or embed subtitles to a movie, we've got you covered. MyTag is intended to be used by the average user that wants to describe their personal files to fit their unique preferences. Users will be able to enhance their file managing experience through the application of elegant and straightforward syntax.

Language Tutorial

Requirements

It is required that the user has Python 3.6.4 or above. PLY parsing tool is also required.
The rest of the required libraries are inside the requirement.txt file in the MyTag repository.

Getting Started

First of all, clone the MyTag repository to your computer. Now, to begin the setup, make sure python is already installed. The external libraries need to be installed using PIP (if you have Python 3 installed, then this is already installed).

Then, from the terminal, navigate to the clone repository and run the following command:

pip install –r requirements.txt

Now that you have everything installed, you can begin to edit your metadata using MyTag!

To start using MyTag, run:

python3 myTag.py

And you are now writing in the MyTag language! To see what you can do with this language, read the example and reference manual below.

Example of a Program

Music albums are a great example of why having good metadata is so useful. With MyTag you can easily organize your files by putting all songs from the same album in a separate folder and add the shared album metadata to each song with just a few lines of code.

To get started, a variable must be defined for the specific file type to handle. In this example, we are dealing with an audio file, therefore *aud* is used. (For more information on file types, see *Language Reference Manual.*) A file type variable is always a path to the file or series of files.

>>> aud testAlbum = /Users/your_name/Music/More Life

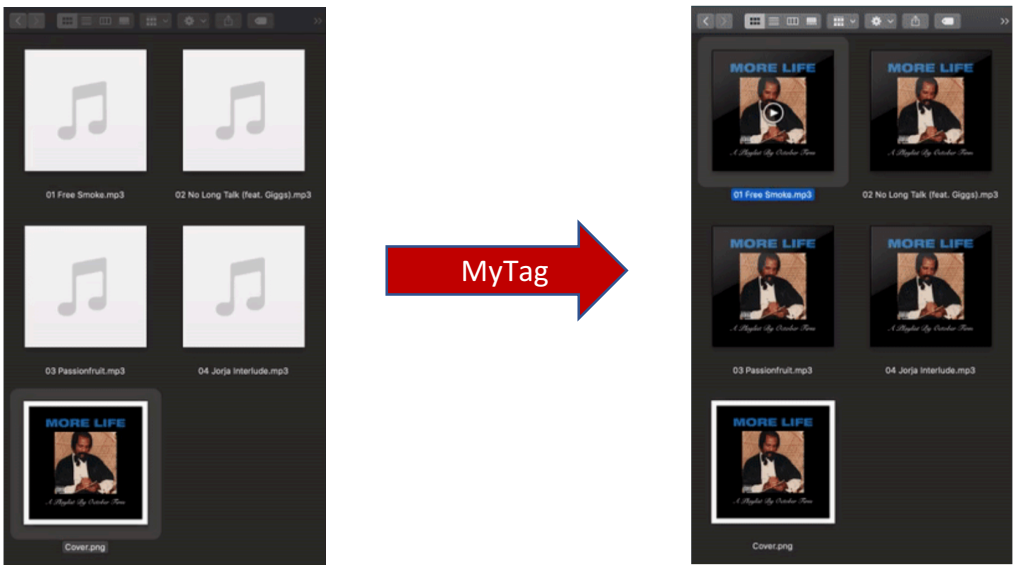
Now we can proceed to edit the file's metadata. For simplicity, we will only edit the artist, album, date and artwork of the album.

>>> set testAlbum<artist> = "Drake"
>>> set testAlbum<album> = "More Life"
>>> set testAlbum<date> = "2017"
>>> set testAlbum<artwork> = /Users/your_name/Music/More Life/Cover.png

After updating the desired metadata, to finish we save the changes.

>>> save testAlbum

It's that simple, in just six lines of straightforward code we've obtained these results:



Language Reference Manual

MyTag currently allows the user to handle multiple file types*:

File Type	MyTag Type	Format
Audio	aud	.mp3
Video	vid	.mp4
Document	doc	.pdf

*More formats could work with the current version of MyTag with some minor source code modifications, but only these were tested for the purpose of this project.

Initialization

type name = value

- *type*: the type of file to handle.
  - Example: aud, vid, doc
- *name*: the name of the file object.
- *value:* the path to the file to be handled. Can be written directly as a path or as a previously defined path variable.
  - Example: *type name = /path/to/file*
    or
    type name = var, where *var* is a previously defined path.

name = value

- *name*: the name of the variable
- *value*: the value of the variable. Can be a path or a string.
  - Example: name = /path/to/file
    or
    name = “String goes here”

These variables are of type None. They do not point to a file object; therefore, methods cannot be applied to them and are only used to facilitate your code writing whenever a path or string is used multiple times.

Get

get name<tag>

- *get*: calls the get function of the object and tag given.
- *name*: the name of the file object.
- *tag*: the metadata tag to be accessed. The current available tags are the following:
  - aud: *title, artist, album, albumartist, composer, genre, date, tracknumber, discnumber, bpm.*
  - vid: *title, artist, album, track, genre, date, description, comment, grouping.*
  - doc: *format, title, author, subject, keywords, creator, producer, creationDate, modDate, encryption.*
- *return*: prints tag in console.

Set
set name<tag> = value

- *set*: calls the set function of the object, tag and value given.
- *name*: the name of the file object.
- *tag*: the metadata tag to be updated. The current available tags are the following:
    - aud: *title, artist, album, albumartist, composer, genre, date, tracknumber, discnumber, bpm, artwork.*
    - vid: *title, artist, album, track, genre, date, description, comment, grouping, artwork.*
    - doc: *format, title, author, subject, keywords, creator, producer, creationDate, modDate, encryption, watermark, toc.*
- *value*: the string (or path in some cases) to be assigned to the given tag. Can be written directly as a string (or path) or as a previously defined variable.
    - Example: *set name<tag> = "new tag value"*
      or
      set name <tag> = var, where *var* is previously defined.

Both *artwork* and *watermark* require you to give a path to the new item. Table of Contents (*toc*) is written as follows:

set name<toc> = "hierarchy-level title page-number"

Add
add name<tag> = value

- *add*: call the add function of the object, tag and value given.
- *name*: the name of the file object.
- *tag*: the metadata tag to be added. The current available tags are the following:
    - vid: *chapter, sub.*
- *Value*: the string or path to be assigned to the given tag. Can be written directly as a string or path or as a previously defined variable.
    - Example: add name<sub> = /path/to/sub
      or
      add name<chapter> = "# of chapters"
      or
      add name<tag> = var, where *var* is previously defined.

Clear
clear name

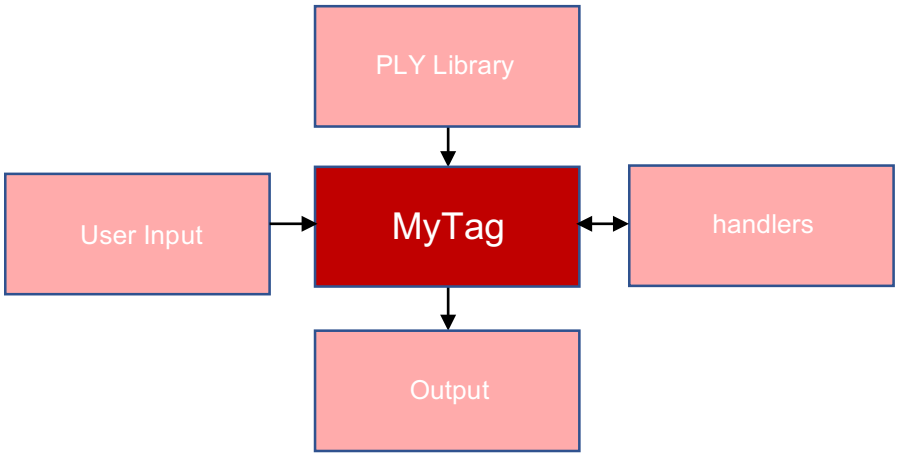
- *clear*: clears the file object's metadata.
- *name*: the name of the file object to be cleared.

Save
save name

- *save*: saves the file object's metadata.
- *name*: the name of the file object to be saved.

Language Development

Translator Architecture



Interface Between Modules

The main module is called myTag.py. During the execution of this program, the user input is passed through lexer to see if there are any syntax errors. The lexer output is then sent to the parser where it will determine the meaning of the input according to the programming language's grammar. The parser will communicate with the corresponding file handler and execute the functions that the user calls. If the function returns a value, said value will be sent to myTag where it will be outputted to the console.

Software Development Environment

- **PyCharm** – IDE used to develop MyTag programming language.
- **Python** – high-level programming language for general purpose programming.
- **PLY** – implementation of lex and yacc parsing tools for python.
- **GitHub** – web-based hosting service for version control using Git.

Test Methodology

- Tested lexer and parser to make sure complies with the defined grammar.
- Tested each handler method separately on various files and different cases comparing the expected results with the obtained results.
- User feedback.

Tester Program

Example of tester program for video.py

```
>>> import video
>>> pf = "/Users/jeanmerced/Desktop/pulp_fiction.mp4"
>>> title = "Pulp Fiction”
>>> artist = "Quentin Tarantino"
>>> genre = "Crime"
>>> date = "1994"
>>> myvid = video.Video(pf)
>>> myvid.set_tag("title", title)
>>> myvid.set_tag("artist", artist)
>>> myvid.set_tag("genre", genre)
>>> myvid.set_tag("date", date)
>>> myvid.save()
```

Conclusion

MyTag makes modifying metadata an easy task. There are some tools that also modify metadata but few of them handle multiple file types and formats. When editing video metadata, most tools take too long and in case of a crash your media will most likely be corrupted. On the other hand, MyTag is fast and reliable since a copy of the original file is made, therefore, in case of a crash the original file can still be recovered. As for PDF, there are many libraries available, but they each tackle a specific part of the document. MyTag does them all. It is possible that our audio and video implementations work with other formats as well but they were not tested for the purpose of this project. If they do work, only minor adjustments need to be made.