Tagger documentation

Jonas Wennerström

Table of contents

1. Introduction
2. Usage
3. Database
4. Python
    a. External libraries
    b. Files
        i. taggergui
        ii. taggersql
        iii. taggermodels
        iv. taggercreatedb
    c. Flowchart
5. Appendix: Scraper

Introduction

Tagger is a management program for a tag-based database aimed towards tracking media. It provides complete control over the data and an easy to use GUI for quick access to what the user is looking for. The GUI uses tabs within a main window to differentiate between the various functions of the program and to ensure the only information presented to the user is that which is relevant to the current task.

The purpose of the program is to provide a user with a tool to create their own library of media with a straightforward way of storing and finding data relevant to their current interests.

Usage

Find file

The tab presents the user with checkboxes for all tags extant in the database along with a button title “Show results”. The user is free to check as many boxes as they wish to narrow their result. Note that ticking no boxes will give no results.

By clicking “Show results” a new window is opened presenting all files which match the selection: One line for each file. From left to right it shows the file length, the title (which also functions as a link), and a collapsible list of all the files tags.

Add file

The tab presents the user with checkboxes for all tags extant in the database along with entry fields for the three pieces of data relating to the file stored: Length, Title, and Link. Title and Link can be collected from a file selection if the user so chooses. A button titled “Add file” will add the new file, matched with all selected tags, into the database. The user is then notified when the insertion is successful, as well as if requirements for insertion are not met.

Add tags

This tab presents the user with an entry field and a button labelled “Add tags”, as well as a list of all extant tags and brief instructions on how to format their input. New tags are to be written on one line with each tag separated by a comma(“,”) and no spaces. An example is given: Comedy,Drama. The list of extant tags is shown to prevent duplicates. Clicking the Commit button will insert all tags and update all lists showing extant tags.

Update file

This tab looks almost identical to “Add file”, except the Title entry autocompletes based on what exists in the database. Link can be collected from file selection. Upon entering the information, the file entry with the chosen Title is updated.

Remove file

This tab presents the user with an entry field and a button labelled “Delete file”, as well as instructions on what it does and a warning that the deletion is irreversible. Upon beginning to type in the entry field autocompletion of all extant titles in the database is offered.

Remove tag

This tab presents the user with checkboxes of all tags extant in the database along with a button labelled “Delete”. Upon clicking Delete all selected tags and their related matches will be deleted from the database and all lists of extant tags will be updated.

Database

The program uses SQLite 3 to create a local database.

Design

The database consists of three tables:

- `file` - Stores information on files
- `tag` - Stores names of tags
- `match` - Makes connections between file and tag

The three-table design was chosen to ensure many-to-many relationships between `file` and `tag` of arbitrary scope are supported.

A flaw in the design is the `file-match` relationship: `file` is effectively a parent to `match`, yet a `file` without any `match`-entries is inaccessible from the scope of the program. Deletion of `tag`s is therefore generally discouraged.

Structure

```
file
     id         INTEGER PRIMARY KEY
     length     INT
     title      VARCHAR(255) UNIQUE
     link       VARCHAR(255)

tag
     id         INTEGER PRIMARY KEY
     name       VARCHAR(255) UNIQUE

match
     id         INTEGER PRIMARY KEY
     file_id    INTEGER FOREIGN KEY references file.id
     tag_id     INTEGER FOREIGN KEY references tag.id
```

Python

External libraries

appJar - Used to create the GUI and manage user input.
SQLAlchemy - Used to maintain database connection and perform database manipulation.
SQLite3 - Used to create new databases

Files

While describing functions, “the database” is understood to refer to the database open in the current session.

Structure

The program is split into four files with distinct functionality:

- taggergui creates the GUI and manages data collection and formatting for SQL commands.
- taggersql handles all database interaction.
- taggermodels provides SQLAlchemy models for use in other SQLAlchemy functions.
- taggercreatedb implements SQLite3 and creates databases.

A flowchart is provided for overview.

taggergui

This file contains both the GUI itself and all functions used to gather information from it, in addition to creating the database connection. In accordance with appJar documentation, the GUI and its elements are created globally rather than within a function.

Note: A number of functions are listed as taking btn as indata but do not use it: This is due to built-in appJar functionality (‘btn’ is the title of the button pressed as a string).

Global variables

BUTTON_POS - Used to center buttons on the GUI in reference to Properties.
MAX_PROPS - The maximum number of GUI: Properties to create.
WINDOW_SIZE - Initial window size.
TAG_TABLE_PREFIXES - List of all GUI: Property prefixes.
AUTO_ENTRIES - List of all GUI: Auto Entries.
MAX_RESULTS - The maximum number of results shown.
MAX_PROPSIZE - The maximum number of entries in each GUI: Property.
app - The GUI instance.

GUI events

These are the buttons users can press and their function calls.

- Show results(play_window)
- Choose file(file_select_window)
- Add file(add_file)
- Add tags(add_tag)
- Get link from file(file_select_window)
- Update file(update_file)
- Delete file(del_file)
- Delete tags(del_tag)
- Make DB(create_db)

In addition there is a menu:

- New db (new_db)
- Open db (change_db)
- Clean up db (cleanup)
- Propsize (menu_press)
- Results (menu_press)
- Help (show_help)

Functions

Aux functions perform various data-gathering and GUI updates useful to several functions.

`list_files() -> titles:list` Calls `taggersql:get_all_files`, extracts the title of all files in the database and returns them.

`stringify_ticked(prefix:string) -> tagword:string` Returns a string of all 'ticked' options in appJar Properties with titles beginning with `prefix` and ending in `n` for 0≤`n`<`MAX_PROPS`, with each value separated by ','.

`open_file(title:string)` Calls `taggersql:get_link` to get link associated with `title` in database, then opens link using OS default program.

`path_leaf(path:string) -> title` Given a filepath, returns filename.

`close()` Closes session when user closes Tagger.

`make_tag_tables()` Resets, empties, and populates all appJar Properties with prefixes defined in `TAG_TABLE_PREFIXES` with all tag.names existing in the database..

Prep functions are the primary functions called on user input: They prepare data for SQL functions and handles GUI updates.

`lookup_file() -> file_info:SQL query result` Calls `stringify_ticked("Tags to filter")` to get a string of all 'ticked' options on the Find file tab of the GUI ("tags"), then `taggersql:select_file(session, tags)` and returns the result of that call.

`add_file(btn)` Calls `taggersql:insert_file` with information collected from the GUI based on btn then resets the GUI.

`add_tag(btn)` Collects information from the GUI: Entry "new_tags", calls `taggersql:insert_tag` with that information, calls `make_tag_tables`, then resets the GUI.

`update_file(btn)` Calls del_file and add_file to recreate an entry in file and entries in match.

`del_file(btn)` Calls `taggersql:insert_file` with information collected from the GUI based on btn then resets the GUI.

`del_tag(btn)` Calls `stringify_ticked("Tags to delete")`, then `taggersql:delete_tag` with that information, and calls `make_tag_tables()`.

Database management functions perform low-level database maintenance: Creation and cleanup.

`cleanup(btn)` Calls `taggersql:cleanup_files` with the current session.

`create_db(btn)` Calls `taggercreatedb:make_new_db` with argument collected from Entry “Database name:”.

Subwindow functions manage additional windows.

`play_window()` Populates and opens a new window displaying results of a user query: Calls `lookup_file`, then for each instance in the result creates a link calling `open_file` and calls `taggersql:get_all_matchs` to provide more information to the user.

`file_select_window()` Opens a file selection window. Upon user selecting a file, writes file path in entry “Link”, calls `path_leaf` and writes result in entry “Title”.

Menu functions respond to user invocations from the menu and are unrelated to the rest of the GUI.

`new_db(btn)` Opens a window allowing access to create_db.

`change_db(btn)` Opens a window prompting the user to load a db file, and then populates the GUI based on data in db.

`cleanup(btn)` Invokes cleanup.

`menu_press(btn)` Allows user to modify MAX_PROPSIZE and MAX_RESULTS for the current session.

`show_help(btn)` Opens a help window.

taggersql

This file contains all functionality for database manipulation. ‘session’ refers to an SQLAlchemy session instance: An open database connection.

Functions

`select_file(session, tagword:string) -> files:SQL Query result`
Queries the session to find all entries in file with matches to all tag.names in tagword.

`get_link(session, title:string) -> path:string` Returns file.path for file.title == title.

`get_all_file_titles(session) -> titles:SQL Query result` Queries database and returns all file.titles.

`get_all_tag_names(session) -> names:SQL Query result` Queries database and returns all tag.names.

`get_all_matchs(session, title:string) -> tag:SQL Query result`
Returns all tag.names matched with file.title == title.

`insert_file(session, [length:integer, title:string, link:string], tagword:string)` Inserts `[length, title, link]` into file and creates matches between new file and all tag.names in `tagword`. If an entry already exists with title, appends “ i”.

`insert_tag(session, tagword:string)` Inserts all tag.names in tagword into tag.

`delete_tag(session, tagword:string)` Deletes all entries in tag with tag.name existing in tagword.

`delete_file(session, title:string)` Deletes all entries in file with file.title == title.

`cleanup_files(session)` Deletes all entries in file lacking any entry in match.

`file_exists(session, title:string) -> bool` Checks if a file with file.title==title exists in session.

taggermodels

Creates one class representing each table in the database structure for use in SQLAlchemy calls.

taggercreatedb

Imports sqlite3

`make_new_db(title:string)` Uses sqlite3 functionality to create a new database file named title in accordance with the database schema.

Flowchart

GUI Controls	Prep Functions	Aux called	SQL called
Show results	play_window	open_file	get_all_matchs
		lookup_file	
Choose file		file_select_window	
Add file	add_file	list_files	insert_file
		stringify_ticked	
Add tags	add_tag	make_tag_tables	insert_tag
Get link from file		file_select_window	
Update file	update_file		file_exists
	del_file		delete_file
	add_file		insert_file
Delete file	del_file	list_files	delete_file
Delete tags	del_tag	make_tag_tables	delete_tag
		stringify_ticked	
Make DB	create_db		make_new_db

Example: User presses Add file, which invokes: add_file, which invokes insert_file; list_files; stringify_ticked. Aux functions may also invoke SQL functions, see below.

Aux Functions	Aux called	SQL called
list_files		get_all_files
make_tag_tables		get_all_tags
open_file		get_link
cleanup		cleanup_files
lookup_file	stringify_ticked	select_file
file_select_window	path_leaf	
path_leaf		
stringify_ticked		

Menu option	Menu function
New db	new_db
Open db	change_db
Clean up db	cleanup
Config	menu_press
Help	show_help

Appendix: Scraper

For testing and example purposes, a webscraper which creates a database with information from imdb.com top 1000 was made and is included. The core functionality was created by Melih Akdag at kaggle.com: Link to article. Minor modifications were made to collect information of interest to Tagger. In addition a new insertion function was created:

`insert_both(session, fileinfo [length:int, title:string, link:string], tagword:string)` Inserts fileinfo into file, each tag in tagword into tag, and entries in match for the file and each tag. Ensures there are no duplicate tag.name or file.title - if duplicate file.title is found, new entry is appended with “ i”.