MPy3Player Documentation

**Date**: Jun 23, 2022 **Version**: 1.0

MPy3Player is simple and elegant MP3 player, designed to play mp3 files, extract meta data from them, manage playlists and displaying cover art photos.

* [Getting started](#getting_started)
* [User Guide](#user_guide)
* [API reference](#api_reference)

Getting Started

Installation

1. Download [Python 3](https://www.python.org/downloads/)
2. Create folder for the project e.g. C:\Users\youruser\Desktop\MPy3Player
3. Navigate to folder from your terminal and create virtual environment with following command:

$ python -m venv myenv

1. Activate environment by navigating to myenv/Scripts and execute following command

$ activate.bat

1. Navigate back to C:\Users\youruser\Desktop\MPy3Player and copy all files of the project
2. Install requirements.txt with following command:

$ pip install -r requirements.txt

1. Set environment variables for spotify API for windows machines:

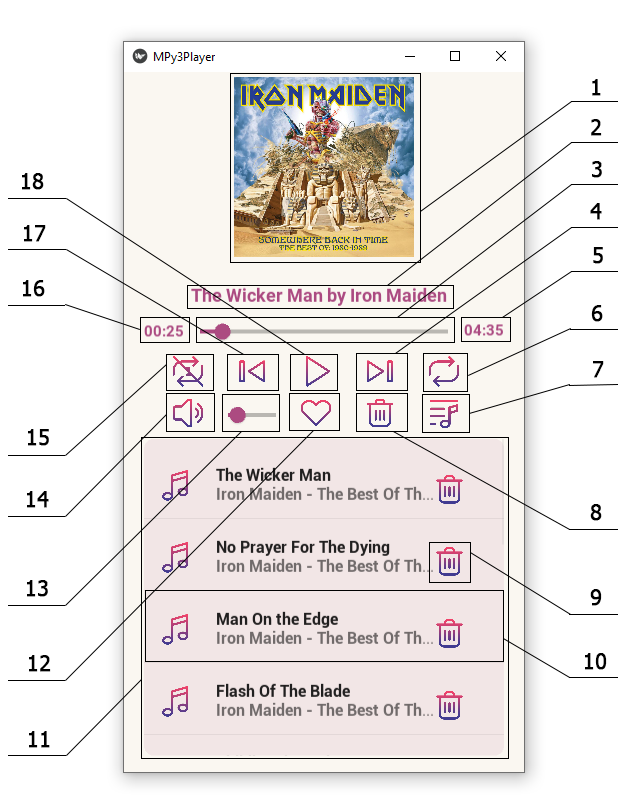
$ set SPOTIPY\_CLIENT\_ID=7b142cd3f4204416b16a772e6ee2464c

$ set SPOTIPY\_CLIENT\_SECRET=de7c0e6c6d4c4d7fbffbdd0bd105a4cf

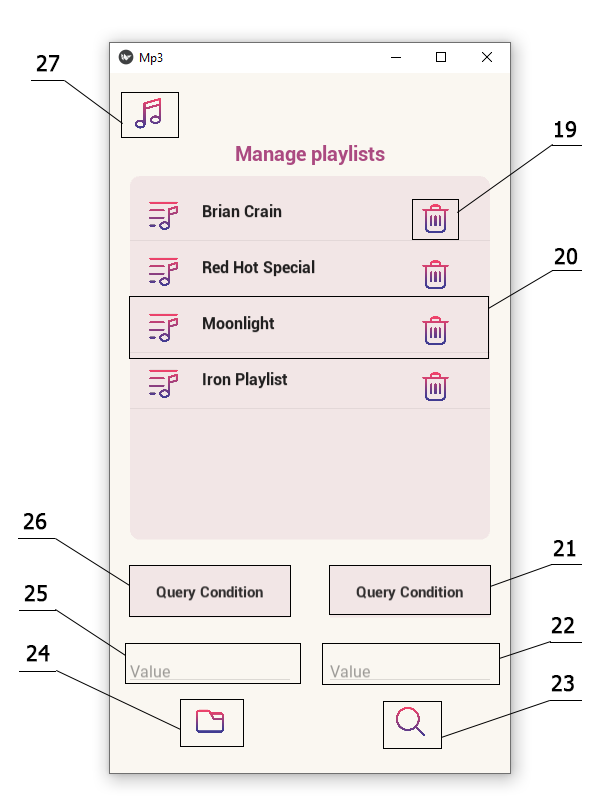
For linux instead of set use export.

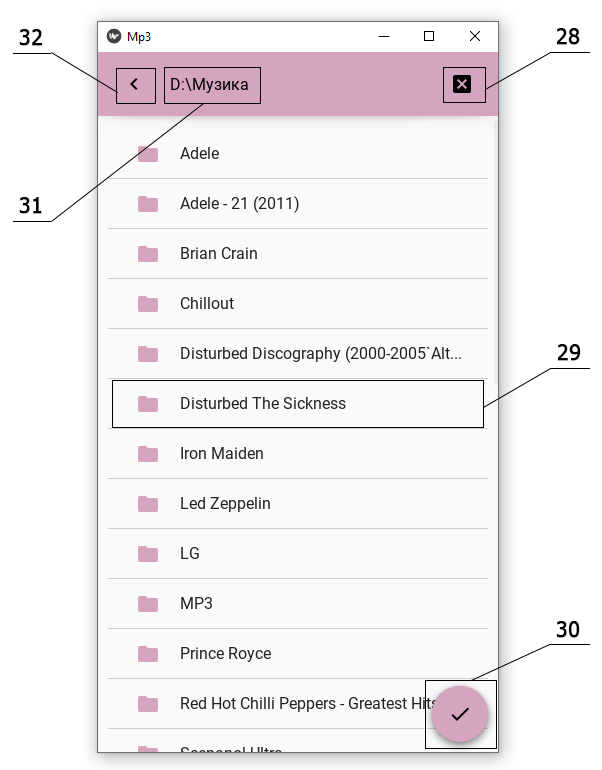
1. Run project with following command:

$ python RumenMP3.py

Intro to MPy3Player

1. Cover art display
2. Now playing label
3. Current song position slider
4. Next
5. Total song length
6. Shuffle
7. Playlists and database
8. Clear playlist
9. Remove current song from song list
10. Song widget
11. Song list
12. Create new playlist
13. Volume slider
14. Mute
15. Play one song ON/OFF
16. Current position of the song
17. Back
18. Play/Pause



1. Delete playlist from database
2. Playlist widget
3. Second query condition
4. Second query parameter
5. Search in database
6. File manager
7. First query parameter
8. First query condition
9. Back to main screen

28. Close file manager

29. Folder

30. Retrieve path

31. Current directory

32. One level up in directories tree

[Back to top](#back_to_top)

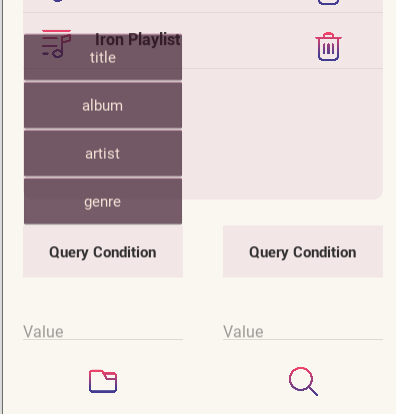
User Guide

* [How to insert song/ folder with songs in database?](#how_to_insert_song_in_database)
* [How to search in database?](#how_to_search_in_database)
* [How to create playlist?](#how_to_create_playlist)
* [How to load playlist?](#load_playlist)

How to insert song/ folder with songs in database?

1. Start player
2. Click playlist and database button in front screen
3. Click on file manager button in second screen
4. Navigate to desired song or folder with songs and press retrieve path button

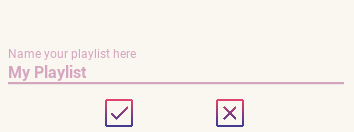
How to search in database?

1. Start player
2. Click playlist and database button in front screen
3. You can search by one or two parameters. If you want to search only by one parameter, first you have to choose an option from one of two spinner boxes noted with “query condition” text over them.
4. You can choose to search for title, album, artist or genre in database. When you choose option, you have to insert parameter in text input box below spinner box. For example, if you want to search for Iron Maiden in data base, in spinner box option you have to choose artist and in input box you have to write Iron. It will make query in database and will search in artist column where name is like iron, and will return all records where in artist column there is “iron”.
5. To search by multiple criteria, you have to use also second spinner box, and for example you can search for any combination between title, album, artist and genre.
6. If we want to search for “Fear of the dark” of “Iron Maiden” in one of spinner box we have to choose artist and in text input below we have to write at least “Iron”. In the other spinner box, we have to choose title and in corresponding input box to write “Fear of the dark”.



1. Click on search in database button

How to create playlist?

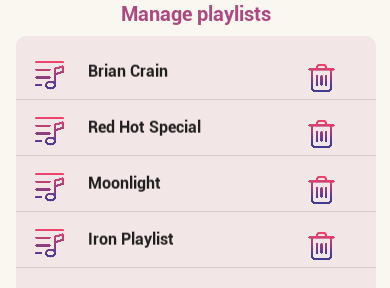
1. Start player
2. Click on Create new playlist button
3. Insert name of playlist



1. Click on Confirm button

How to load playlist?

1. Start player
2. Click playlist and database button in front screen
3. Click on playlist widget



[Back to top](#back_to_top)

API reference

* [MPy3PlayerApp](#MPy3PlayerApp)
* [Screen Manager](#ScreenManager)
* [MainScreen](#MainScreen)
* [SecondScreen](#SecondScreen)
* [SongManager](#SongManager)
* [Song](#song)
* [Playlist](#playlist)
* [StorageManager](#StorageManager)
* [APIManager](#APIManager)

MPy3PlayerApp

Base class to create the application. It`s an entry point into the Kivy run loop.

**Methods**

|  |  |
| --- | --- |
| MPy3PlayerApp.build() | Return constructed widget tree. |
| MPy3PlayerApp.run() | Runs the app. |

ScreenManager

Screen manager is a widget dedicated to managing the two screens of the application.

MainScreen

Main screen of the application.

**Atributes**

|  |  |
| --- | --- |
| MainScreen.if\_art\_not\_called | Boolean parameter showing if call to spotify api is made. |
| MainScreen.is\_playing | Boolean parameter showing if song is playing. |
| MainScreen.muted | Boolean parameter showing if volume is muted. |
| MainScreen.loop\_same\_song | Boolean parameter showing if player have to loop same song. |
| MainScreen.playlist\_index | Integer value representing index of song object in song\_list. |
| MainScreen.volume\_level | Integer value representing volume level. |
| MainScreen.converted\_song\_length | String value representing song length in format “00:00”. |
| MainScreen.raw\_song\_length | Integer value representing song length in seconds. |
| MainScreen.current\_position | Integer value representing current position of progress slider. |
| MainScreen.song\_manager | Instance of SongManager class |

**Methods**

|  |  |
| --- | --- |
| MainScreen.call\_api() | Makes call to spotify api. |
| MainScreen.populate\_playlist(p\_id, first\_spinner,  first\_text\_input, second\_spinner, second\_text\_input) | Populate song list with song widgets based on query made to database. |
| MainScreen.generate\_songlist\_items() | Creates songlist widgets. |
| MainScreen.create\_playlist\_items() | Takes info from database about present playlists and populates playlist list in SecondScreen. |
| MainScreen.remove\_from\_songlist(index) | Removes song object from SongManager`s song\_list by passed index. |
| MainScreen.change\_screen() | Changes screen to MainScreen |
| MainScreen.delete\_playlist(p\_id) | Deletes playlist from database, based on playlist id parameter. |
| MainScreen.play\_from\_list\_item(index, title) | On list item click plays clicked song based on index. |
| MainScreen.save\_playlist(name) | Creates playlist object based on song objects currently in SongManager`s song list. |
| MainScreen.shuffle() | Clears playlist, shuffles song\_list and repopulate playlist. |
| MainScreen.loop\_one() | If loop flag is true sets it to false else sets it to true. |
| MainScreen.clear\_songs\_from\_list() | Clears songlist widgets from playlist and clears song\_list of SongManager class. |
| MainScreen.prev\_next(command) | Stopes and unload current played song. Sets position of progress bar slider to 0. Checks if looping one song is active and if it is continues to play the same song. If previous button is pressed loads previous song object in song\_list, else loads next object. If index is greater than length of song\_lists plays first song, if is less than 0 plays last song. All this based on command parameter which shows which song to be loaded previous or next. |
| MainScreen.load(index) | First unload current song and loads song object contained in song\_list of SongManager instance by passed index. Extracts information about song length and its converted value in format of '00:00' from songs objects. |
| MainScreen.move\_label() | Makes song info label to move. |
| MainScreen.progress\_slider\_update() | First sets max value of progress bar slider, after which sets labels showing song length and current playing position. Position is updating every second by 1. If slider current position is equal to its max value triggers prev\_next method as one time event. |
| MainScreen.play() | If music playing pauses it. If music not playing checks current position of progress bar slider and if  this value is greater than 0 resumes playing song from this position else starts new song. |
| MainScreen.mute(volume\_button\_widget) | Mutes volume if not muted and unmute it if muted. |
| MainScreen.change\_volume(volume\_slider\_widget) | Changes volume based on slider value. Mute sound if value reaches 0. |

SecondScreen

Second screen of the application.

**Atributes**

|  |  |
| --- | --- |
| SecondScreen.manager\_open | Boolean parameter showing if storage manager is open. |
| SecondScreen.directory | String value holding path to selected directory or file. |
| SecondScreen.songs\_paths | List of all paths to selected mp3 files. |
| SecondScreen.storage | Instance of StorageManager class. |
| SecondScreen.file\_manager | Instance of FileManager class. |

**Methods**

|  |  |
| --- | --- |
| SecondScreen.file\_manager\_open() | Opens file\_manager in path set in show method. |
| SecondScreen.exit\_manager() | If there is valid path, file\_manager passes list with all paths to song\_info\_extraction method of StorageManager class, after which file\_manager closes. |
| SecondScreen.select\_path(path) | It will be called when you click on the file name of the retrieve path button. Checks is path is to directory or to file and append paths to songs\_path list. |

SongManager

Class responsible for creation of playlist and songs objects.

**Atributes**

|  |  |
| --- | --- |
| SongManager.song\_ids | List holding song ids of current played playlist. |
| SongManager.song\_list | List holding created song objects. |
| SongManager.playlist\_list | List holding created playlist objects. |
| SongManager.db\_query | List holding information retrieved by methods of StorageManager class. |
| SongManager.storage | Instance of StorageManager class. |
| SongManager.api | Instance of APIManager class. |

**Methods**

|  |  |
| --- | --- |
| SongManager.create\_song\_objects(p\_id, f\_spinner, f\_text, s\_spinner, s\_text) | Creates song objects from saved playlist or from database queries. |
| SongManager.getting\_s\_ids\_from\_song\_list() | Gets song ids from song objects in song\_list. |
| SongManager.repair\_current\_playlist(hash\_value,  s\_id) | If path passed to pygame mixer load method is not valid, takes id and hash value of the song, pops song object from SongManager song\_list with matching s\_id, makes query to check if there is another record with same hash value in database and if such exists, inserts new song object on place of old one. |
| SongManager. create\_playlist\_objects() | Creates playlist objects. |

Song

Class responsible for creation of Song objects.

**Atributes**

|  |  |
| --- | --- |
| Song.s\_id | Integer value representing song`s id. |
| Song.title | String value representing song`s title. |
| Song.album | String value representing song`s album. |
| Song.artist | String value representing song`s artist. |
| Song.genre | String value representing song`s genre. |
| Song.raw\_length | Integer value representing song`s length in seconds. |
| Song.converted\_length | String value representing song`s converted length in format “00:00”. |
| Song.path | String value representing song`s path. |
| Song.hash\_value | Integer value representing song`s hash representation. |

Playlist

Class responsible for creation of Playlist objects.

**Atributes**

|  |  |
| --- | --- |
| Playlist.p\_id | Integer value representing playlist`s id. |
| Playlist.name | String value representing playlist`s name. |

StorageManager

Class responsible for interactions with database.

**Atributes**

|  |  |
| --- | --- |
| StorageManager.records | List of songs extracted info. |

**Methods**

|  |  |
| --- | --- |
| StorageManager.writing\_songs\_to\_database() | Checking if hash value of song we would like to write in database exists and if not writes all the data (ID3 tags plus song length, path, calculated crc32 value of the song) to database. |
| StorageManager.song\_info\_extraction(audio\_path) | Extracts ID3 tags from MP3s, creates hash values of each song based on these tags and converts length of the song in format “00:00” based on audio\_path. |
| StorageManager.check\_for\_path\_changed(hash\_value, s\_id) | Checks if there is different song record of same song based on hash\_value and song id of current song. Returns record with matching hash value but different song id. |
| StorageManager.get\_songs\_from\_database(f\_spinner, f\_text, s\_spinner, s\_text) | Fetch song information from database, based on values passed from user input. |
| StorageManager.delete\_song\_from\_database(s\_id) | Deletes song CASCADE from database. |
| StorageManager.get\_playlists\_from\_database() | Fetches data for records in playlist table in database. |
| StorageManager.get\_playlist\_songs\_from\_database(p\_id) | Fetches playlist record based on playlist id. |
| StorageManager.delete\_playlist\_from\_database(p\_id) | Deletes playlist CASCADE from database. |
| StorageManager.create\_playlist(name, song\_ids) | Makes record in playlist table in database after that takes p\_id of the playlists and makes records in playlist\_songs junction table based on song ids extracted in song\_ids container of SongManager class. |
| StorageManager.create\_data\_base() | Creates schema of playlist database. |

APIManager

Class responsible for API queries.

**Atributes**

|  |  |
| --- | --- |
| APIManager.spotify | Instance of spotipy`s Spotify class. |

**Methods**

|  |  |
| --- | --- |
| APIManager.find\_art(album, artist) | Makes query to Spotify API based on album and artist names, and tries to extract url address of cover art. |

[Back to top](#back_to_top)