ytmusicapi

Release 1.10.2

sigma67

Mar 05, 2025

CONTENTS

The purpose of this library is to automate interactions with YouTube Music, such as retrieving your library content, managing playlists and uploading songs. To achieve this, it emulates web requests that would occur if you performed the same actions in your web browser.

This project is not supported nor endorsed by Google

CHAPTER

ONE

FEATURES

Browsing:

- search (including all filters) and suggestions
- get artist information and releases (songs, videos, albums, singles, related artists)
- get user information (videos, playlists)
- get albums
- get song metadata
- get watch playlists (next songs when you press play/radio/shuffle in YouTube Music)
- get song lyrics

Exploring music:

- get moods and genres playlists
- get latest charts (globally and per country)

Library management:

- get library contents: playlists, songs, artists, albums and subscriptions, podcasts, channels
- add/remove library content: rate songs, albums and playlists, subscribe/unsubscribe artists
- get and modify play history

Playlists:

- create and delete playlists
- modify playlists: edit metadata, add/move/remove tracks
- get playlist contents
- get playlist suggestions

Podcasts:

- get podcasts
- get episodes
- get channels
- get episodes playlists

Uploads:

- upload songs and remove them again
- list uploaded songs, artists and albums

Localization:

- all regions are supported (see locations FAQ
- 16 languages are supported (see languages FAQ

If you find something missing or broken, check the FAQ or feel free to create an issue.

CHAPTER

TWO

REQUIREMENTS

- Python 3.9 or higher - https://www.python.org

CHAPTER

THREE

SETUP

See the Documentation for detailed instructions

CHAPTER

FOUR

USAGE

```
from ytmusicapi import YTMusic

yt = YTMusic('oauth.json')
playlistId = yt.create_playlist('test', 'test description')
search_results = yt.search('Oasis Wonderwall')
yt.add_playlist_items(playlistId, [search_results[0]['videoId']])
```

The tests are also a great source of usage examples.

To **get started**, read the *setup instructions*.

For a **complete documentation** of available functions, see the *Reference*.

CHAPTER

FIVE

CONTENTS

5.1 Setup

To install, run:

```
pip install ytmusicapi
```

Further setup is only needed if you want to access account data using authenticated requests.

The simplest way of authentication is to use *OAuth authentication*.

However, these OAuth credentials do not work for uploads. If you need to upload music, instead follow the instructions at *Browser authentication*.

5.1.1 OAuth authentication

Attention: As of November 2024, YouTube Music requires a Client Id and Secret for the YouTube Data API to connect to the API.

Go to the YouTube Data API docs to obtain the credentials. This requires a Google Cloud Console account and project.

For your new credentials, select `OAuth client ID` and pick `TVs and Limited Input devices`.

After you have installed `ytmusicapi`, run

```
ytmusicapi oauth
```

and follow the instructions. This will create a file `oauth.json` in the current directory.

You can pass this file to `YTMusic` as explained in *Usage*.

You will also need to pass `client_id` and `client_secret` to `YTMusic`:

```
from ytmusicapi import YTMusic, OAuthCredentials

ytmusic = YTMusic('oauth.json', oauth_credentials=OAuthCredentials(client_id=client_id,␣
↪client_secret=client_secret)
```

This OAuth flow uses the Google API flow for TV devices.

5.1.2 Browser authentication

This method of authentication emulates your browser session by reusing its request headers. Follow the instructions to have your browser's YouTube Music session request headers parsed to a `ytmusicapi` configuration file.

Copy authentication headers

To run authenticated requests, set it up by first copying your request headers from an authenticated POST request in your browser. To do so, follow these steps:

- Open a new tab
- Open the developer tools (Ctrl-Shift-I) and select the "Network" tab
- Go to https://music.youtube.com and ensure you are logged in
- Find an authenticated POST request. The simplest way is to filter by `/browse` using the search bar of the developer tools. If you don't see the request, try scrolling down a bit or clicking on the library button in the top bar.
- Verify that the request looks like this: **Status** 200, **Method** POST, **Domain** music.youtube.com, **File** `browse?...`
- Copy the request headers (right click > copy > copy request headers)
- Verify that the request looks like this: **Status** 200, **Name** `browse?...`
- Click on the Name of any matching request. In the "Headers" tab, scroll to the section "Request headers" and copy everything starting from "accept: */*" to the end of the section

Using the headers in your project

To set up your project, open a console and call

```
ytmusicapi browser
```

Follow the instructions and paste the request headers to the terminal input.

If you don't want terminal interaction in your project, you can pass the request headers with the `headers_raw` parameter:

```
import ytmusicapi
ytmusicapi.setup(filepath="browser.json", headers_raw="<headers copied above>")
```

The function returns a JSON string with the credentials needed for *Usage*. Alternatively, if you passed the filepath parameter as described above, a file called `browser.json` will be created in the current directory, which you can pass to `YTMusic()` for authentication.

These credentials remain valid as long as your YTMusic browser session is valid (about 2 years unless you log out).

- MacOS terminal application can only accept 1024 characters pasted to std input. To paste in terminal, a small utility called pbpaste must be used.
- **In terminal just prefix the command used to run the script you created above with**
  `pbpaste |`
- This will pipe the contents of the clipboard into the script just as if you had pasted it from the edit menu.

Manual file creation

Alternatively, you can create your own file `browser.json` and paste the cookie:

```
{
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0) Gecko/20100101␣
↪Firefox/72.0",
    "Accept": "*/*",
    "Accept-Language": "en-US,en;q=0.5",
    "Content-Type": "application/json",
    "X-Goog-AuthUser": "0",
    "x-origin": "https://music.youtube.com",
    "Cookie" : "PASTE_COOKIE"
}
```

5.2 Usage

5.2.1 Unauthenticated

Unauthenticated requests for retrieving playlist content or searching:

```
from ytmusicapi import YTMusic

ytmusic = YTMusic()
```

If an endpoint requires authentication you will receive an error: `Please provide authentication before using this function`

5.2.2 Authenticated

For authenticated requests you need to set up your credentials first: *Setup*

After you have created the authentication JSON, you can instantiate the class:

```
from ytmusicapi import YTMusic, OAuthCredentials
ytmusic = YTMusic("browser.json") # or, alternatively
ytmusic = YTMusic("oauth.json", oauth_credentials=OAuthCredentials(client_id=client_id,␣
↪client_secret=client_secret)
```

With the `ytmusic` instance you can now perform authenticated requests:

```
playlistId = ytmusic.create_playlist("test", "test description")
search_results = ytmusic.search("Oasis Wonderwall")
ytmusic.add_playlist_items(playlistId, [search_results[0]['videoId']])
```

Brand accounts

To send requests as a brand account, there is no need to change authentication credentials. Simply provide the ID of the brand account when instantiating YTMusic. You can get the ID from https://myaccount.google.com/ after selecting your brand account (https://myaccount.google.com/b/21_digit_number).

Example:

```
from ytmusicapi import YTMusic
ytmusic = YTMusic("oauth.json", "101234161234936123473")
```

5.3 Reference

Reference for the YTMusic class.

5.3.1 YTMusic

class ytmusicapi.**YTMusic**(*auth: str | dict | None = None*, *user: str | None = None*, *requests_session: requests.sessions.Session | None = None*, *proxies: dict[str, str] | None = None*, *language: str = 'en'*, *location: str = ''*, *oauth_credentials:* ytmusicapi.auth.oauth.credentials.OAuthCredentials *| None = None*)

Allows automated interactions with YouTube Music by emulating the YouTube web client's requests. Permits both authenticated and non-authenticated requests. Authentication header data must be provided on initialization.

YTMusic.**__init__**(*auth: str | dict | None = None*, *user: str | None = None*, *requests_session: requests.sessions.Session | None = None*, *proxies: dict[str, str] | None = None*, *language: str = 'en'*, *location: str = ''*, *oauth_credentials:* ytmusicapi.auth.oauth.credentials.OAuthCredentials *| None = None*)

Create a new instance to interact with YouTube Music.

Parameters

- **auth** (Union[str, dict, None]) – Optional. Provide a string, path to file, or oauth token dict. Authentication credentials are needed to manage your library. See *setup()* for how to fill in the correct credentials. Default: A default header is used without authentication.
- **user** (Optional[str]) – Optional. Specify a user ID string to use in requests. This is needed if you want to send requests on behalf of a brand account. Otherwise the default account is used. You can retrieve the user ID by going to https://myaccount.google.com/brandaccounts and selecting your brand account. The user ID will be in the URL: https://myaccount.google.com/b/user_id/
- **requests_session** (Optional[Session]) – A Requests session object or None to create one. Default sessions have a request timeout of 30s, which produces a requests.exceptions.ReadTimeout. The timeout can be changed by passing your own Session object:

```
s = requests.Session()
s.request = functools.partial(s.request, timeout=3)
ytm = YTMusic(requests_session=s)
```

- **proxies** (Optional[dict[str, str]]) – Optional. Proxy configuration in requests format.

- **language** (str) – Optional. Can be used to change the language of returned data. English will be used by default. Available languages can be checked in the ytmusicapi/locales directory.
- **location** (str) – Optional. Can be used to change the location of the user. No location will be set by default. This means it is determined by the server. Available languages can be checked in the FAQ.
- **oauth_credentials** (Optional[*OAuthCredentials*]) – Optional. Used to specify a different oauth client to be used for authentication flow.

5.3.2 Setup

See also the *Setup* page

ytmusicapi.**setup**(*filepath: str | None = None*, *headers_raw: str | None = None*) → str

Requests browser headers from the user via command line and returns a string that can be passed to YTMusic()

Parameters

- **filepath** (Optional[str]) – Optional filepath to store headers to.
- **headers_raw** (Optional[str]) – Optional request headers copied from browser. Otherwise requested from terminal

Return type
str

Returns
configuration headers string

ytmusicapi.**setup_oauth**(*client_id: str*, *client_secret: str*, *filepath: str | None = None*, *session: requests.sessions.Session | None = None*, *proxies: dict | None = None*, *open_browser: bool = False*) → *ytmusicapi.auth.oauth.token.RefreshingToken*

Starts oauth flow from the terminal and returns a string that can be passed to YTMusic()

Parameters

- **client_id** (str) – Optional. Used to specify the client_id oauth should use for authentication flow. If provided, client_secret MUST also be passed or both will be ignored.
- **client_secret** (str) – Optional. Same as client_id but for the oauth client secret.
- **session** (Optional[Session]) – Session to use for authentication
- **proxies** (Optional[dict]) – Proxies to use for authentication
- **filepath** (Optional[str]) – Optional filepath to store headers to.
- **open_browser** (bool) – If True, open the default browser with the setup link

Return type
RefreshingToken

Returns
configuration headers string

5.3.3 Search

YTMusic.**search**(*query: str*, *filter: str | None = None*, *scope: str | None = None*, *limit: int = 20*, *ignore_spelling: bool = False*) → list[dict]

Search YouTube music Returns results within the provided category.

Parameters

- **query** (str) – Query string, i.e. 'Oasis Wonderwall'
- **filter** (Optional[str]) – Filter for item types. Allowed values: songs, videos, albums, artists, playlists, community_playlists, featured_playlists, uploads. Default: Default search, including all types of items.
- **scope** (Optional[str]) – Search scope. Allowed values: library, uploads. Default: Search the public YouTube Music catalogue. Changing scope from the default will reduce the number of settable filters. Setting a filter that is not permitted will throw an exception. For uploads, no filter can be set. For library, community_playlists and featured_playlists filter cannot be set.
- **limit** (int) – Number of search results to return Default: 20
- **ignore_spelling** (bool) – Whether to ignore YTM spelling suggestions. If True, the exact search term will be searched for, and will not be corrected. This does not have any effect when the filter is set to uploads. Default: False, will use YTM's default behavior of autocorrecting the search.

Return type
list[dict]

Returns

List of results depending on filter. resultType specifies the type of item (important for default search). albums, artists and playlists additionally contain a browseId, corresponding to albumId, channelId and playlistId (browseId=`VL`+playlistId)

Example list for default search with one result per resultType for brevity. Normally there are 3 results per resultType and an additional thumbnails key:


```
[
  {
    "category": "Top result",
    "resultType": "video",
    "videoId": "vU05Eksc_iM",
    "title": "Wonderwall",
    "artists": [
      {
        "name": "Oasis",
        "id": "UCmMUZbaYdNH0bEd1PAlAqsA"
      }
    ],
    "views": "1.4M",
    "videoType": "MUSIC_VIDEO_TYPE_OMV",
    "duration": "4:38",
    "duration_seconds": 278
  },
  {
    "category": "Songs",
```


(continues on next page)

(continued from previous page)


```
    "resultType": "song",
    "videoId": "ZrOKjDZOtkA",
    "title": "Wonderwall",
    "artists": [
      {
        "name": "Oasis",
        "id": "UCmMUZbaYdNH0bEd1PAlAqsA"
      }
    ],
    "album": {
      "name": "(What's The Story) Morning Glory? (Remastered)",
      "id": "MPREb_9nqEki4ZDpp"
    },
    "duration": "4:19",
    "duration_seconds": 259
    "isExplicit": false,
    "feedbackTokens": {
      "add": null,
      "remove": null
    }
  },
  {
    "category": "Albums",
    "resultType": "album",
    "browseId": "MPREb_IInSY5QXXrW",
    "playlistId": "OLAK5uy_kunInnOpcKECWIBQGB0Qj6ZjquxDvfckg",
    "title": "(What's The Story) Morning Glory?",
    "type": "Album",
    "artist": "Oasis",
    "year": "1995",
    "isExplicit": false
  },
  {
    "category": "Community playlists",
    "resultType": "playlist",
    "browseId": "VLPLK1PkWQlWtnNfovRdGWpKffO1Wdi2kvDx",
    "title": "Wonderwall - Oasis",
    "author": "Tate Henderson",
    "itemCount": "174"
  },
  {
    "category": "Videos",
    "resultType": "video",
    "videoId": "bx1Bh8ZvH84",
    "title": "Wonderwall",
    "artists": [
      {
        "name": "Oasis",
        "id": "UCmMUZbaYdNH0bEd1PAlAqsA"
      }
    ],
    "views": "386M",
```


(continues on next page)

(continued from previous page)


```
    "duration": "4:38",
    "duration_seconds": 278
  },
  {
    "category": "Artists",
    "resultType": "artist",
    "browseId": "UCmMUZbaYdNH0bEd1PAlAqsA",
    "artist": "Oasis",
    "shuffleId": "RDAOkjHYJjL1a3xspEyVkhHAsg",
    "radioId": "RDEMkjHYJjL1a3xspEyVkhHAsg"
  },
  {
    "category": "Profiles",
    "resultType": "profile",
    "title": "Taylor Swift Time",
    "name": "@TaylorSwiftTime",
    "browseId": "UCSCRK7XlVQ6fBdEl00kX6pQ",
    "thumbnails": ...
  }
]
```


YTMusic.**get_search_suggestions**(*query: str*, *detailed_runs=False*) → list[str] | list[dict]

Get Search Suggestions

Parameters

- **query** (str) – Query string, i.e. 'faded'
- **detailed_runs** – Whether to return detailed runs of each suggestion. If True, it returns the query that the user typed and the remaining suggestion along with the complete text (like many search services usually bold the text typed by the user). Default: False, returns the list of search suggestions in plain text.

Return type

Union[list[str], list[dict]]

Returns

A list of search suggestions. If `detailed_runs` is False, it returns plain text suggestions.
If `detailed_runs` is True, it returns a list of dictionaries with detailed information.

Example response when `query` is 'fade' and `detailed_runs` is set to `False`:


```
[
  "faded",
  "faded alan walker lyrics",
  "faded alan walker",
  "faded remix",
  "faded song",
  "faded lyrics",
  "faded instrumental"
]
```


Example response when `detailed_runs` is set to `True`:

```
[
  {
    "text": "faded",
    "runs": [
      {
        "text": "fade",
        "bold": true
      },
      {
        "text": "d"
      }
    ],
    "fromHistory": true,
    "feedbackToken": "AEEJK..."
  },
  {
    "text": "faded alan walker lyrics",
    "runs": [
      {
        "text": "fade",
        "bold": true
      },
      {
        "text": "d alan walker lyrics"
      }
    ],
    "fromHistory": false,
    "feedbackToken": None
  },
  {
    "text": "faded alan walker",
    "runs": [
      {
        "text": "fade",
        "bold": true
      },
      {
        "text": "d alan walker"
      }
    ],
    "fromHistory": false,
    "feedbackToken": None
  },
  ...
]
```

YTMusic.**remove_search_suggestions**(*suggestions: list[dict[str, Any]]*, *indices: list[int] | None = None*) → bool

Remove search suggestion from the user search history.

Parameters

- **suggestions** (list[dict[str, Any]]) – The dictionary obtained from the *get_search_suggestions()* (with detailed_runs=True)`

- **indices** (Optional[list[int]]) – Optional. The indices of the suggestions to be removed. Default: remove all suggestions.

Return type
bool

Returns

True if the operation was successful, False otherwise.

Example usage:

```
# Removing suggestion number 0
suggestions = ytmusic.get_search_suggestions(query="fade", detailed_
↪runs=True)
success = ytmusic.remove_search_suggestions(suggestions=suggestions,␣
↪indices=[0])
if success:
    print("Suggestion removed successfully")
else:
    print("Failed to remove suggestion")
```

5.3.4 Browsing

YTMusic.**get_home**(*limit=3*) → list[dict]

Get the home page. The home page is structured as titled rows, returning 3 rows of music suggestions at a time. Content varies and may contain artist, album, song or playlist suggestions, sometimes mixed within the same row

Parameters
limit – Number of rows to return

Return type
list[dict]

Returns
List of dictionaries keyed with 'title' text and 'contents' list

Example list:

```
[
    {
        "title": "Your morning music",
        "contents": [
            { //album result
                "title": "Sentiment",
                "browseId": "MPREb_QtqXtd2xZMR",
                "thumbnails": [...]
            },
            { //playlist result
                "title": "r/EDM top submissions 01/28/2022",
                "playlistId": "PLz7-xrYmULdSLRZGk-6GKUtaBZcgQNwel",
                "thumbnails": [...],
                "description": "redditEDM • 161 songs",
                "count": "161",
                "author": [
```

(continues on next page)

(continued from previous page)

```
                    {
                        "name": "redditEDM",
                        "id": "UCaTrZ9tPiIGHrkCe5bxOGwA"
                    }
                ]
            }
        ]
    },
    {
        "title": "Your favorites",
        "contents": [
            { //artist result
                "title": "Chill Satellite",
                "browseId": "UCrPLFBWdOroD57bkqPbZJog",
                "subscribers": "374",
                "thumbnails": [...]
            }
            { //album result
                "title": "Dragon",
                "year": "Two Steps From Hell",
                "browseId": "MPREb_M9aDqLRbSeg",
                "thumbnails": [...]
            }
        ]
    },
    {
        "title": "Quick picks",
        "contents": [
            { //song quick pick
                "title": "Gravity",
                "videoId": "EludZd6lfts",
                "artists": [{
                        "name": "yetep",
                        "id": "UCSW0r7dClqCoCvQeqXiZBlg"
                    }],
                "thumbnails": [...],
                "album": {
                    "name": "Gravity",
                    "id": "MPREb_D6bICFcuuRY"
                }
            },
            { //video quick pick
                "title": "Gryffin & Illenium (feat. Daya) - Feel Good (L3V3LS Remix)
↪",
                "videoId": "bR5l0hJDnX8",
                "artists": [
                    {
                        "name": "L3V3LS",
                        "id": "UCCVNihbOdkOWw_-ajIYhAbQ"
                    }
                ],
                "thumbnails": [...],
```

(continues on next page)

(continued from previous page)

```
                "views": "10M"
            }
        ]
    }
]
```

YTMusic.**get_artist**(*channelId: str*) → dict

Get information about an artist and their top releases (songs, albums, singles, videos, and related artists). The top lists contain pointers for getting the full list of releases.

Possible content types for get_artist are:

- songs
- albums
- singles
- shows
- videos
- episodes
- podcasts
- related

Each of these content keys in the response contains `results` and possibly `browseId` and `params`.

- For songs/videos, pass the browseId to *get_playlist()*.
- For albums/singles/shows, pass browseId and params to *get_artist_albums()*.

Parameters
channelId (`str`) – channel id of the artist

Return type
`dict`

Returns
Dictionary with requested information.

Warning: The returned channelId is not the same as the one passed to the function. It should be used only with *subscribe_artists()*.

Example:

```
{
    "description": "Oasis were ...",
    "views": "3,693,390,359 views",
    "name": "Oasis",
    "channelId": "UCUDVBtnOQi4c7E8jebpjc9Q",
    "shuffleId": "RDAOkjHYJjL1a3xspEyVkhHAsg",
    "radioId": "RDEMkjHYJjL1a3xspEyVkhHAsg",
    "subscribers": "3.86M",
    "subscribed": false,
```

(continues on next page)

(continued from previous page)


```
    "thumbnails": [...],
    "songs": {
        "browseId": "VLPLMpM3Z0118S42R1npOhcjoakLIv1aqnS1",
        "results": [
            {
                "videoId": "ZrOKjDZOtkA",
                "title": "Wonderwall (Remastered)",
                "thumbnails": [...],
                "artist": "Oasis",
                "album": "(What's The Story) Morning Glory? (Remastered)"
            }
        ]
    },
    "albums": {
        "results": [
            {
                "title": "Familiar To Millions",
                "thumbnails": [...],
                "year": "2018",
                "browseId": "MPREb_AYetWMZunqA"
            }
        ],
        "browseId": "UCmMUZbaYdNH0bEd1PAlAqsA",
        "params": "6gPTAUNwc0JDbndLYlFBQV..."
    },
    "singles": {
        "results": [
            {
                "title": "Stand By Me (Mustique Demo)",
                "thumbnails": [...],
                "year": "2016",
                "browseId": "MPREb_7MPKLhibN5G"
            }
        ],
        "browseId": "UCmMUZbaYdNH0bEd1PAlAqsA",
        "params": "6gPTAUNwc0JDbndLYlFBQV..."
    },
    "videos": {
        "results": [
            {
                "title": "Wonderwall",
                "thumbnails": [...],
                "views": "358M",
                "videoId": "bx1Bh8ZvH84",
                "playlistId": "PLMpM3Z0118S5xuNckw1HUcj1D021AnMEB"
            }
        ],
        "browseId": "VLPLMpM3Z0118S5xuNckw1HUcj1D021AnMEB"
    },
    "related": {
        "results": [
            {
```


(continues on next page)

(continued from previous page)

```
                "browseId": "UCt2KxZpY5D__kapeQ8cauQw",
                "subscribers": "450K",
                "title": "The Verve"
            },
            {
                "browseId": "UCwK2Grm574W1u-sBzLikldQ",
                "subscribers": "341K",
                "title": "Liam Gallagher"
            },
            ...
        ]
    }
}
```

YTMusic.**get_artist_albums**(*channelId: str*, *params: str*, *limit: int | None = 100*, *order: Literal['Recency', 'Popularity', 'Alphabetical order'] | None = None*) → list[dict]

Get the full list of an artist's albums, singles or shows

Parameters

- **channelId** (str) – browseId of the artist as returned by *get_artist()*
- **params** (str) – params obtained by *get_artist()*
- **limit** (Optional[int]) – Number of albums to return. None retrieves them all. Default: 100
- **order** (Optional[Literal['Recency', 'Popularity', 'Alphabetical order']]) – Order of albums to return. Allowed values: Recency, Popularity, *Alphabetical order*. Default: Default order.

Return type
list[dict]

Returns
List of albums in the format of *get_library_albums()*, except artists key is missing.

YTMusic.**get_album**(*browseId: str*) → dict

Get information and tracks of an album

Parameters
browseId (str) – browseId of the album, for example returned by *search()*

Return type
dict

Returns
Dictionary with album and track metadata.

The result is in the following format:

```
{
  "title": "Revival",
  "type": "Album",
  "thumbnails": [],
  "description": "Revival is the...",
  "artists": [
    {
```

(continues on next page)

(continued from previous page)


```
      "name": "Eminem",
      "id": "UCedvOgsKFzcK3hA5taf3KoQ"
    }
  ],
  "year": "2017",
  "trackCount": 19,
  "duration": "1 hour, 17 minutes",
  "audioPlaylistId": "OLAK5uy_nMr9h2VlS-2PULNz3M3XVXQj_P3C2bqaY",
  "tracks": [
    {
      "videoId": "iKLU7z_xdYQ",
      "title": "Walk On Water (feat. Beyoncé)",
      "artists": [
        {
          "name": "Eminem",
          "id": "UCedvOgsKFzcK3hA5taf3KoQ"
        }
      ],
      "album": "Revival",
      "likeStatus": "INDIFFERENT",
      "thumbnails": null,
      "isAvailable": true,
      "isExplicit": true,
      "duration": "5:03",
      "duration_seconds": 303,
      "trackNumber": 0,
      "feedbackTokens": {
        "add": "AB9zfpK...",
        "remove": "AB9zfpK..."
      }
    }
  ],
  "other_versions": [
    {
      "title": "Revival",
      "year": "Eminem",
      "browseId": "MPREb_fefKFOTEZSp",
      "thumbnails": [...],
      "isExplicit": false
    },
  ],
  "duration_seconds": 4657
}
```


YTMusic.**get_album_browse_id**(*audioPlaylistId: str*) → str | None

Get an album's browseId based on its audioPlaylistId

Parameters

audioPlaylistId (str) – id of the audio playlist (starting with *OLAK5uy_*)

Return type

Optional[str]

Returns

browseId (starting with MPREb_)

YTMusic.**get_user**(*channelId: str*) → dict

Retrieve a user's page. A user may own videos or playlists.

Use *get_user_playlists()* to retrieve all playlists:

```
result = get_user(channelId)
get_user_playlists(channelId, result["playlists"]["params"])
```

Similarly, use *get_user_videos()* to retrieve all videos:

```
get_user_videos(channelId, result["videos"]["params"])
```

Parameters
channelId (str) – channelId of the user

Return type
dict

Returns
Dictionary with information about a user.

Example:

```
{
  "name": "4Tune - No Copyright Music",
  "videos": {
    "browseId": "UC44hbeRoCZVVMVg5z0FfIww",
    "results": [
      {
        "title": "Epic Music Soundtracks 2019",
        "videoId": "bJonJjgS2mM",
        "playlistId": "RDAMVMbJonJjgS2mM",
        "thumbnails": [
          {
            "url": "https://i.ytimg.com/vi/bJon...",
            "width": 800,
            "height": 450
          }
        ],
        "views": "19K"
      }
    ]
  },
  "playlists": {
    "browseId": "UC44hbeRoCZVVMVg5z0FfIww",
    "results": [
      {
        "title": " Machinimasound | Playlist",
        "playlistId": "PLRm766YvPiO9ZqkBuEzSTt6Bk4eWIr3gB",
        "thumbnails": [
          {
            "url": "https://i.ytimg.com/vi/...",
            "width": 400,
```

(continues on next page)

(continued from previous page)

```
            "height": 225
          }
        ]
      }
    ],
    "params": "6gO3AUNvWU..."
  }
}
```

YTMusic.**get_user_playlists**(*channelId: str*, *params: str*) → list[dict]

Retrieve a list of playlists for a given user. Call this function again with the returned `params` to get the full list.

Parameters

- **channelId** (`str`) – channelId of the user.
- **params** (`str`) – params obtained by *get_user()*

Return type
`list[dict]`

Returns
List of user playlists in the format of *get_library_playlists()*

YTMusic.**get_user_videos**(*channelId: str*, *params: str*) → list[dict]

Retrieve a list of videos for a given user. Call this function again with the returned `params` to get the full list.

Parameters

- **channelId** (`str`) – channelId of the user.
- **params** (`str`) – params obtained by *get_user()*

Return type
`list[dict]`

Returns
List of user videos

YTMusic.**get_song**(*videoId: str*, *signatureTimestamp: int | None = None*) → dict

Returns metadata and streaming information about a song or video.

Parameters

- **videoId** (`str`) – Video id
- **signatureTimestamp** (`Optional[int]`) – Provide the current YouTube signatureTimestamp. If not provided a default value will be used, which might result in invalid streaming URLs

Return type
`dict`

Returns
Dictionary with song metadata.

Example:

```
{
    "playabilityStatus": {
```

(continues on next page)

(continued from previous page)


```
        "status": "OK",
        "playableInEmbed": true,
        "audioOnlyPlayability": {
            "audioOnlyPlayabilityRenderer": {
                "trackingParams": "CAEQx2kiEwiuv9X5i5H1AhWBvlUKHRoZAHk=",
                "audioOnlyAvailability": "FEATURE_AVAILABILITY_ALLOWED"
            }
        },
        "miniplayer": {
            "miniplayerRenderer": {
                "playbackMode": "PLAYBACK_MODE_ALLOW"
            }
        },
        "contextParams": "Q0FBU0FnZ0M="
    },
    "streamingData": {
        "expiresInSeconds": "21540",
        "adaptiveFormats": [
            {
                "itag": 140,
                "url": "https://rr1---sn-h0jelnez.c.youtube.com/videoplayback?
↪expire=1641080272...",
                "mimeType": "audio/mp4; codecs="mp4a.40.2"",
                "bitrate": 131007,
                "initRange": {
                    "start": "0",
                    "end": "667"
                },
                "indexRange": {
                    "start": "668",
                    "end": "999"
                },
                "lastModified": "1620321966927796",
                "contentLength": "3967382",
                "quality": "tiny",
                "projectionType": "RECTANGULAR",
                "averageBitrate": 129547,
                "highReplication": true,
                "audioQuality": "AUDIO_QUALITY_MEDIUM",
                "approxDurationMs": "245000",
                "audioSampleRate": "44100",
                "audioChannels": 2,
                "loudnessDb": -1.3000002
            }
        ]
    },
    "playbackTracking": {
        "videostatsPlaybackUrl": {
          "baseUrl": "https://s.youtube.com/api/stats/playback?cl=491307275&
↪docid=AjXQiKP5kMs&ei=Nl2HY-6MH5WE8gPjnYnoDg&fexp=1714242%2C9405963%2C23804281
↪%2C23858057%2C23880830%2C23880833%2C23882685%2C23918597%2C23934970%2C23946420
↪%2C23966208%2C23983296%2C23998056%2C24001373%2C24002022%2C24002025%2C24004644
```


(continues on next page)

(continued from previous page)


```
↪%2C24007246%2C24034168%2C24036947%2C24077241%2C24080738%2C24120820%2C24135310
↪%2C24135692%2C24140247%2C24161116%2C24162919%2C24164186%2C24169501%2C24175560
↪%2C24181174%2C24187043%2C24187377%2C24187854%2C24191629%2C24197450%2C24199724
↪%2C24200839%2C24209349%2C24211178%2C24217535%2C24219713%2C24224266%2C24241378
↪%2C24248091%2C24248956%2C24255543%2C24255545%2C24262346%2C24263796%2C24265426
↪%2C24267564%2C24268142%2C24279196%2C24280220%2C24283426%2C24283493%2C24287327
↪%2C24288045%2C24290971%2C24292955%2C24293803%2C24299747%2C24390674%2C24391018
↪%2C24391537%2C24391709%2C24392268%2C24392363%2C24392401%2C24401557%2C24402891
↪%2C24403794%2C24406605%2C24407200%2C24407665%2C24407914%2C24408220%2C24411766
↪%2C24413105%2C24413820%2C24414162%2C24415866%2C24416354%2C24420756%2C24421162
↪%2C24425861%2C24428962%2C24590921%2C39322504%2C39322574%2C39322694%2C39322707&
↪ns=yt&plid=AAXusD4TIOMjS5N4&el=detailpage&len=246&of=Jx1iRksbq-rB9N1KSijZLQ&
↪osid=MWU2NzBjYTI%3AAOeUNAagU8UyWDUJIki5raGHy29-60-yTA&uga=29&
↪vm=CAEQABgEOjJBUEV3RWxUNmYzMXNMMC1MYVpCVnRZTmZWMWw1OWVZX2ZOcUtCSkphQ245VFZwOXdTQWJbQVBta0tETEpWNX
↪",
          "headers": [
            {
              "headerType": "USER_AUTH"
            },
            {
              "headerType": "VISITOR_ID"
            },
            {
              "headerType": "PLUS_PAGE_ID"
            }
          ]
        },
        "videostatsDelayplayUrl": {(as above)},
        "videostatsWatchtimeUrl": {(as above)},
        "ptrackingUrl": {(as above)},
        "qoeUrl": {(as above)},
        "atrUrl": {(as above)},
        "videostatsScheduledFlushWalltimeSeconds": [
          10,
          20,
          30
        ],
        "videostatsDefaultFlushIntervalSeconds": 40
    },
    "videoDetails": {
        "videoId": "AjXQiKP5kMs",
        "title": "Sparks",
        "lengthSeconds": "245",
        "channelId": "UCvCk2zFqkCYzpnSgWfx0qOg",
        "isOwnerViewing": false,
        "isCrawlable": false,
        "thumbnail": {
            "thumbnails": []
        },
        "allowRatings": true,
        "viewCount": "12",
        "author": "Thomas Bergersen",
```


(continues on next page)

(continued from previous page)


```
        "isPrivate": true,
        "isUnpluggedCorpus": false,
        "musicVideoType": "MUSIC_VIDEO_TYPE_PRIVATELY_OWNED_TRACK",
        "isLiveContent": false
    },
    "microformat": {
        "microformatDataRenderer": {
            "urlCanonical": "https://music.youtube.com/watch?v=AjXQiKP5kMs",
            "title": "Sparks - YouTube Music",
            "description": "Uploaded to YouTube via YouTube Music Sparks",
            "thumbnail": {
                "thumbnails": [
                    {
                        "url": "https://i.ytimg.com/vi/AjXQiKP5kMs/hqdefault.jpg",
                        "width": 480,
                        "height": 360
                    }
                ]
            },
            "siteName": "YouTube Music",
            "appName": "YouTube Music",
            "androidPackage": "com.google.android.apps.youtube.music",
            "iosAppStoreId": "1017492454",
            "iosAppArguments": "https://music.youtube.com/watch?v=AjXQiKP5kMs",
            "ogType": "video.other",
            "urlApplinksIos": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=applinks",
            "urlApplinksAndroid": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=applinks",
            "urlTwitterIos": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=twitter-deep-link",
            "urlTwitterAndroid": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=twitter-deep-link",
            "twitterCardType": "player",
            "twitterSiteHandle": "@YouTubeMusic",
            "schemaDotOrgType": "http://schema.org/VideoObject",
            "noindex": true,
            "unlisted": true,
            "paid": false,
            "familySafe": true,
            "pageOwnerDetails": {
                "name": "Music Library Uploads",
                "externalChannelId": "UCvCk2zFqkCYzpnSgWfx0qOg",
                "youtubeProfileUrl": "http://www.youtube.com/channel/
↪UCvCk2zFqkCYzpnSgWfx0qOg"
            },
            "videoDetails": {
                "externalVideoId": "AjXQiKP5kMs",
                "durationSeconds": "246",
                "durationIso8601": "PT4M6S"
            },
            "linkAlternates": [
```


(continues on next page)

(continued from previous page)

```
                {
                    "hrefUrl": "android-app://com.google.android.youtube/http/
↪youtube.com/watch?v=AjXQiKP5kMs"
                },
                {
                    "hrefUrl": "ios-app://544007664/http/youtube.com/watch?
↪v=AjXQiKP5kMs"
                },
                {
                    "hrefUrl": "https://www.youtube.com/oembed?format=json&url=https
↪%3A%2F%2Fmusic.youtube.com%2Fwatch%3Fv%3DAjXQiKP5kMs",
                    "title": "Sparks",
                    "alternateType": "application/json+oembed"
                },
                {
                    "hrefUrl": "https://www.youtube.com/oembed?format=xml&url=https
↪%3A%2F%2Fmusic.youtube.com%2Fwatch%3Fv%3DAjXQiKP5kMs",
                    "title": "Sparks",
                    "alternateType": "text/xml+oembed"
                }
            ],
            "viewCount": "12",
            "publishDate": "1969-12-31",
            "category": "Music",
            "uploadDate": "1969-12-31"
        }
    }
}
```

YTMusic.**get_song_related**(*browseId: str*)

Gets related content for a song. Equivalent to the content shown in the "Related" tab of the watch panel.

Parameters

browseId (str) – The `related` key in the `get_watch_playlist` response.

Example:

```
[
  {
    "title": "You might also like",
    "contents": [
      {
        "title": "High And Dry",
        "videoId": "7fv84nPfTH0",
        "artists": [{
            "name": "Radiohead",
            "id": "UCr_iyUANcn9OX_yy9piYoLw"
          }],
        "thumbnails": [
          {
            "url": "https://lh3.googleusercontent.com/
↪TWWT47cHLv3yAugk4h9eOzQ46FHmXc_g-KmBVy2d4sbg_F-Gv6xrPglztRVzp8D_l-yzOnvh-
↪QToM8s=w60-h60-l90-rj",
```

(continues on next page)

(continued from previous page)


```
            "width": 60,
            "height": 60
          }
        ],
        "isExplicit": false,
        "album": {
          "name": "The Bends",
          "id": "MPREb_xsmDKhqhQrG"
        }
      }
    ]
  },
  {
    "title": "Recommended playlists",
    "contents": [
      {
        "title": "'90s Alternative Rock Hits",
        "playlistId": "RDCLAK5uy_m_h-nx7OCFaq9AlyXv78lG0AuloqW_NUA",
        "thumbnails": [...],
        "description": "Playlist • YouTube Music"
      }
    ]
  },
  {
    "title": "Similar artists",
    "contents": [
      {
        "title": "Noel Gallagher",
        "browseId": "UCu7yYcX_wIZgG9azR3PqrxA",
        "subscribers": "302K",
        "thumbnails": [...]
      }
    ]
  },
  {
    "title": "Oasis",
    "contents": [
      {
        "title": "Shakermaker",
        "year": "2014",
        "browseId": "MPREb_WNGQWp5czjD",
        "thumbnails": [...]
      }
    ]
  },
  {
    "title": "About the artist",
    "contents": "Oasis were a rock band consisting of Liam Gallagher, Paul ...␣
↪(full description shortened for documentation)"
  }
]
```

YTMusic.**get_lyrics**(*browseId: str*, *timestamps: bool | None = False*) → *ytmusicapi.models.lyrics.Lyrics* | *ytmusicapi.models.lyrics.TimedLyrics* | None

Returns lyrics of a song or video. When *timestamps* is set, lyrics are returned with timestamps, if available.

Parameters

- **browseId** (str) – Lyrics browseId obtained from *get_watch_playlist()* (startswith MPLYt...).
- **timestamps** (Optional[bool]) – Optional. Whether to return bare lyrics or lyrics with timestamps, if available. (Default: *False*)

Return type

Union[*Lyrics*, *TimedLyrics*, None]

Returns

Dictionary with song lyrics or None, if no lyrics are found. The hasTimestamps-key determines the format of the data.

Example when *timestamps*=*False*, or no timestamps are available:

```
{
    "lyrics": "Today is gonna be the day\nThat they're gonna throw it␣
↪back to you\n",
    "source": "Source: LyricFind",
    "hasTimestamps": False
}
```

Example when *timestamps* is set to *True* and timestamps are available:

```
{
    "lyrics": [
        LyricLine(
            text="I was a liar",
            start_time=9200,
            end_time=10630,
            id=1
        ),
        LyricLine(
            text="I gave in to the fire",
            start_time=10680,
            end_time=12540,
            id=2
        ),
    ],
    "source": "Source: LyricFind",
    "hasTimestamps": True
}
```

YTMusic.**get_tasteprofile**() → dict

Fetches suggested artists from taste profile (music.youtube.com/tasteprofile). Tasteprofile allows users to pick artists to update their recommendations. Only returns a list of suggested artists, not the actual list of selected entries

Return type

dict

Returns
Dictionary with artist and their selection & impression value

Example:

```
{
    "Drake": {
        "selectionValue": "tastebuilder_selection=/m/05mt_q"
        "impressionValue": "tastebuilder_impression=/m/05mt_q"
    }
}
```

YTMusic.**set_tasteprofile**(*artists: list[str]*, *taste_profile: dict | None = None*) → None

Favorites artists to see more recommendations from the artist. Use *get_tasteprofile()* to see which artists are available to be recommended

Parameters

- **artists** (list[str]) – A List with names of artists, must be contained in the tasteprofile
- **taste_profile** (Optional[dict]) – tasteprofile result from *get_tasteprofile()*. Pass this if you call *get_tasteprofile()* anyway to save an extra request.

Return type
None

Returns
None if successful

5.3.5 Explore

YTMusic.**get_mood_categories**() → dict

Fetch "Moods & Genres" categories from YouTube Music.

Return type
dict

Returns
Dictionary of sections and categories.

Example:

```
{
    'For you': [
        {
            'params': 'ggMPOg1uX1ZwN0pHT2NBT1Fk',
            'title': '1980s'
        },
        {
            'params': 'ggMPOg1uXzZQbDB5eThLRTQ3',
            'title': 'Feel Good'
        },
        ...
    ],
    'Genres': [
        {
            'params': 'ggMPOg1uXzVLbmZnaWI4STNs',
```

(continues on next page)

(continued from previous page)

```
            'title': 'Dance & Electronic'
        },
        {
            'params': 'ggMPOg1uX3NjZllsNGVEMkZo',
            'title': 'Decades'
        },
        ...
    ],
    'Moods & moments': [
        {
            'params': 'ggMPOg1uXzVuc0dnZlhpV3Ba',
            'title': 'Chill'
        },
        {
            'params': 'ggMPOg1uX2ozUHlwbWM3ajNq',
            'title': 'Commute'
        },
        ...
    ],
}
```

YTMusic.**get_mood_playlists**(*params: str*) → list[dict]

Retrieve a list of playlists for a given "Moods & Genres" category.

Parameters
params (str) – params obtained by *get_mood_categories()*

Return type
list[dict]

Returns
List of playlists in the format of *get_library_playlists()*

YTMusic.**get_charts**(*country: str = 'ZZ'*) → dict

Get latest charts data from YouTube Music: Top songs, top videos, top artists and top trending videos. Global charts have no Trending section, US charts have an extra Genres section with some Genre charts.

Parameters
country (str) – ISO 3166-1 Alpha-2 country code. Default: ZZ = Global

Return type
dict

Returns
Dictionary containing chart songs (only if authenticated with premium account), chart videos, chart artists and trending videos.

Example:

```
{
    "countries": {
        "selected": {
            "text": "United States"
        },
        "options": ["DE",
            "ZZ",
```

(continues on next page)

(continued from previous page)


```
            "ZW"]
    },
    "songs": {
        "playlist": "VLPL4fGSI1pDJn6O1LS0XSdF3RyO0Rq_LDeI",
        "items": [
            {
                "title": "Outside (Better Days)",
                "videoId": "oT79YlRtXDg",
                "artists": [
                    {
                        "name": "MO3",
                        "id": "UCdFt4Cvhr7Okaxo6hZg5K8g"
                    },
                    {
                        "name": "OG Bobby Billions",
                        "id": "UCLusb4T2tW3gOpJS1fJ-A9g"
                    }
                ],
                "thumbnails": [...],
                "isExplicit": true,
                "album": {
                    "name": "Outside (Better Days)",
                    "id": "MPREb_fX4Yv8frUNv"
                },
                "rank": "1",
                "trend": "up"
            }
        ]
    },
    "videos": {
        "playlist": "VLPL4fGSI1pDJn69On1f-8NAvX_CYlx7QyZc",
        "items": [
            {
                "title": "EVERY CHANCE I GET (Official Music Video) (feat. Lil Baby␣
↪& Lil Durk)",
                "videoId": "BTivsHlVcGU",
                "playlistId": "PL4fGSI1pDJn69On1f-8NAvX_CYlx7QyZc",
                "thumbnails": [],
                "views": "46M"
            }
        ]
    },
    "artists": {
        "playlist": null,
        "items": [
            {
                "title": "YoungBoy Never Broke Again",
                "browseId": "UCR28YDxjDE3ogQROaNdnRbQ",
                "subscribers": "9.62M",
                "thumbnails": [],
                "rank": "1",
                "trend": "neutral"
```


(continues on next page)

(continued from previous page)


```
            }
        ]
    },
    "genres": [
        {
            "title": "Top 50 Pop Music Videos United States",
            "playlistId": "PL4fGSI1pDJn77aK7sAW2AT0oOzo5inWY8",
            "thumbnails": []
        }
    ],
    "trending": {
        "playlist": "VLPLrEnWoR732-DtKgaDdnPkezM_nDidBU9H",
        "items": [
            {
                "title": "Permission to Dance",
                "videoId": "CuklIb9d3fI",
                "playlistId": "PLrEnWoR732-DtKgaDdnPkezM_nDidBU9H",
                "artists": [
                    {
                        "name": "BTS",
                        "id": "UC9vrvNSL3xcWGSkV86REBSg"
                    }
                ],
                "thumbnails": [],
                "views": "108M"
            }
        ]
    }
}
```


5.3.6 Watch

YTMusic.**get_watch_playlist**(*videoId: str | None = None*, *playlistId: str | None = None*, *limit=25*, *radio: bool = False*, *shuffle: bool = False*) → dict[str, Union[list[dict], str, NoneType]]

Get a watch list of tracks. This watch playlist appears when you press play on a track in YouTube Music.

Please note that the `INDIFFERENT` likeStatus of tracks returned by this endpoint may be either `INDIFFERENT` or `DISLIKE`, due to ambiguous data returned by YouTube Music.

Parameters

- **videoId** (`Optional[str]`) – videoId of the played video
- **playlistId** (`Optional[str]`) – playlistId of the played playlist or album
- **limit** – minimum number of watch playlist items to return
- **radio** (`bool`) – get a radio playlist (changes each time)
- **shuffle** (`bool`) – shuffle the input playlist. only works when the playlistId parameter is set at the same time. does not work if radio=True

Return type

`dict[str, Union[list[dict], str, None]]`

Returns

List of watch playlist items. The counterpart key is optional and only appears if a song has a corresponding video counterpart (UI song/video switcher).

Example:


```
{
    "tracks": [
        {
          "videoId": "9mWr4c_ig54",
          "title": "Foolish Of Me (feat. Jonathan Mendelsohn)",
          "length": "3:07",
          "thumbnail": [
            {
              "url": "https://lh3.googleusercontent.com/
↪ulK2YaLtOW0PzcN7ufltG6e4ae3WZ9Bvg8CCwhe6LOccu1lCKxJy2r5AsYrsHeMBSLrGJCNpJqXgwczk=w60-
↪h60-l90-rj",
              "width": 60,
              "height": 60
            }...
          ],
          "feedbackTokens": {
            "add": "AB9zfpIGg9XN4u2iJ...",
            "remove": "AB9zfpJdzWLcdZtC..."
          },
          "likeStatus": "INDIFFERENT",
          "videoType": "MUSIC_VIDEO_TYPE_ATV",
          "artists": [
            {
              "name": "Seven Lions",
              "id": "UCYd2yzYRx7b9FYnBSlbnknA"
            },
            {
              "name": "Jason Ross",
              "id": "UCVCD9Iwnqn2ipN9JIF6B-nA"
            },
            {
              "name": "Crystal Skies",
              "id": "UCTJZESxeZ0J_M7JXyFUVmvA"
            }
          ],
          "album": {
            "name": "Foolish Of Me",
            "id": "MPREb_C8aRK1qmsDJ"
          },
          "year": "2020",
          "counterpart": {
            "videoId": "E0S4W34zFMA",
            "title": "Foolish Of Me [ABGT404] (feat. Jonathan Mendelsohn)",
            "length": "3:07",
            "thumbnail": [...],
            "feedbackTokens": null,
            "likeStatus": "LIKE",
            "artists": [
```


(continues on next page)

(continued from previous page)


```
            {
              "name": "Jason Ross",
              "id": null
            },
            {
              "name": "Seven Lions",
              "id": null
            },
            {
              "name": "Crystal Skies",
              "id": null
            }
          ],
          "views": "6.6K"
        }
      },...
  ],
  "playlistId": "RDAMVM4y33h81phKU",
  "lyrics": "MPLYt_HNNclO0Ddoc-17"
}
```


5.3.7 Library

YTMusic.**get_library_playlists**(*limit: int | None = 25*) → list[dict]

Retrieves the playlists in the user's library.

Parameters
limit (Optional[int]) – Number of playlists to retrieve. None retrieves them all.

Return type
list[dict]

Returns
List of owned playlists.

Each item is in the following format:

```
{
    'playlistId': 'PLQwVIlKxHM6rz0fDJVv_0UlXGEWf-bFys',
    'title': 'Playlist title',
    'thumbnails: [...],
    'count': 5
}
```

YTMusic.**get_library_songs**(*limit: int = 25*, *validate_responses: bool = False*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the songs in the user's library (liked videos are not included). To get liked songs and videos, use *get_liked_songs()*

Parameters

- **limit** (int) – Number of songs to retrieve
- **validate_responses** (bool) – Flag indicating if responses from YTM should be validated and retried in case when some songs are missing. Default: False

- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of songs to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of songs. Same format as *get_playlist()*

YTMusic.**get_library_albums**(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the albums in the user's library.

Parameters

- **limit** (int) – Number of albums to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of albums to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of albums.

Each item is in the following format:

```
{
  "browseId": "MPREb_G8AiyN7RvFg",
  "playlistId": "OLAK5uy_lKgoGvlrWhX0EIPavQUXxyPed8Cj38AWc",
  "title": "Beautiful",
  "type": "Album",
  "thumbnails": [...],
  "artists": [{
    "name": "Project 46",
    "id": "UCXFv36m62USAN5rnVct9B4g"
  }],
  "year": "2015"
}
```

YTMusic.**get_library_artists**(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the artists of the songs in the user's library.

Parameters

- **limit** (int) – Number of artists to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of artists to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of artists.

Each item is in the following format:

```
{
  "browseId": "UCxEqaQWosMHaTih-tgzDqug",
  "artist": "WildVibes",
  "subscribers": "2.91K",
  "thumbnails": [...]
}
```

YTMusic.**get_library_subscriptions**(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the artists the user has subscribed to.

Parameters

- **limit** (int) – Number of artists to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of artists to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of artists. Same format as *get_library_artists()*

YTMusic.**get_library_podcasts**(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Get podcasts the user has added to the library

Parameters

- **limit** (int) – Number of podcasts to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of podcasts to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of podcasts. New Episodes playlist is the first podcast returned, but only if subscribed to relevant podcasts.

Example:

```
[
    {
        "title": "New Episodes",
        "channel":
        {
            "id": null,
            "name": "Auto playlist"
        },
        "browseId": "VLRDPN",
        "podcastId": "RDPN",
        "thumbnails": [...]
```

(continues on next page)

(continued from previous page)

```
    },
    {
        "title": "5 Minuten Harry Podcast",
        "channel":
        {
            "id": "UCDIDXF4WM1qQzerrxeEfSdA",
            "name": "coldmirror"
        },
        "browseId": "MPSPPLDvBqWb1UAGeEt9n6vFH_zdGw65Obf3sH",
        "podcastId": "PLDvBqWb1UAGeEt9n6vFH_zdGw65Obf3sH",
        "thumbnails": [...]
    }
]
```

YTMusic.**get_library_channels**(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Get channels the user has added to the library

Parameters

- **limit** (int) – Number of channels to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of channels to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of channels.

Example:

```
[
    {
        "browseId": "UCRFF8xw5dg9mL4r5ryFOtKw",
        "artist": "Jumpers Jump",
        "subscribers": "1.54M",
        "thumbnails": [...]
    },
    {
        "browseId": "UCQ3f2_sO3NJyDkuCxCNSOVA",
        "artist": "BROWN BAG",
        "subscribers": "74.2K",
        "thumbnails": [...]
    }
]
```

YTMusic.**get_liked_songs**(*limit: int = 100*) → dict

Gets playlist items for the 'Liked Songs' playlist

Parameters
limit (int) – How many items to return. Default: 100

Return type
dict

Returns
List of playlistItem dictionaries. See *get_playlist()*

YTMusic.**get_saved_episodes**(*limit: int = 100*) → dict

Gets playlist items for the 'Liked Songs' playlist

Parameters
limit (int) – How many items to return. Default: 100

Return type
dict

Returns
List of playlistItem dictionaries. See *get_playlist()*

YTMusic.**get_history**() → list[dict]

Gets your play history in reverse chronological order

Return type
list[dict]

Returns
List of playlistItems, see *get_playlist()* The additional property `played` indicates when the playlistItem was played The additional property `feedbackToken` can be used to remove items with *remove_history_items()*

YTMusic.**add_history_item**(*song*)

Add an item to the account's history using the playbackTracking URI obtained from *get_song()*. A `204` return code indicates success.

Usage:

```
song = yt_auth.get_song(videoId)
response = yt_auth.add_history_item(song)
```

Note: You need to use the same YTMusic instance as you used for *get_song()*.

Parameters
song – Dictionary as returned by *get_song()*

Returns
Full response. response.status_code is 204 if successful

YTMusic.**remove_history_items**(*feedbackTokens: list[str]*) → dict

Remove an item from the account's history. This method does currently not work with brand accounts

Parameters
feedbackTokens (list[str]) – Token to identify the item to remove, obtained from *get_history()*

Return type
dict

Returns
Full response

YTMusic.**rate_song**(*videoId: str*, *rating: str = 'INDIFFERENT'*) → dict | None

Rates a song ("thumbs up"/"thumbs down" interactions on YouTube Music)

Parameters

- **videoId** (str) – Video id
- **rating** (str) – One of LIKE, DISLIKE, INDIFFERENT

  INDIFFERENT removes the previous rating and assigns no rating

Return type
Optional[dict]

Returns
Full response

YTMusic.**edit_song_library_status**(*feedbackTokens: list[str] | None = None*) → dict

Adds or removes a song from your library depending on the token provided.

Parameters
feedbackTokens (Optional[list[str]]) – List of feedbackTokens obtained from authenticated requests to endpoints that return songs (i.e. *get_album()*)

Return type
dict

Returns
Full response

YTMusic.**rate_playlist**(*playlistId: str*, *rating: str = 'INDIFFERENT'*) → dict

Rates a playlist/album ("Add to library"/"Remove from library" interactions on YouTube Music) You can also dislike a playlist/album, which has an effect on your recommendations

Parameters

- **playlistId** (str) – Playlist id
- **rating** (str) – One of LIKE, DISLIKE, INDIFFERENT

  INDIFFERENT removes the playlist/album from the library

Return type
dict

Returns
Full response

YTMusic.**subscribe_artists**(*channelIds: list[str]*) → dict

Subscribe to artists. Adds the artists to your library

Parameters
channelIds (list[str]) – Artist channel ids

Return type
dict

Returns
Full response

YTMusic.**unsubscribe_artists**(*channelIds: list[str]*) → dict

Unsubscribe from artists. Removes the artists from your library

Parameters
channelIds (list[str]) – Artist channel ids

Return type
dict

Returns
Full response

YTMusic.**get_account_info**() → dict

Gets information about the currently authenticated user's account.

Return type
dict

Returns
Dictionary with user's account name, channel handle, and URL of their account photo.

Example:

```
{
    "accountName": "Sample User",
    "channelHandle": "@SampleUser
    "accountPhotoUrl": "https://yt3.ggpht.com/sample-user-photo"
}
```

5.3.8 Playlists

YTMusic.**get_playlist**(*playlistId: str*, *limit: int | None = 100*, *related: bool = False*, *suggestions_limit: int = 0*) → dict

Returns a list of playlist items

Parameters

- **playlistId** (str) – Playlist id
- **limit** (Optional[int]) – How many songs to return. None retrieves them all. Default: 100
- **related** (bool) – Whether to fetch 10 related playlists or not. Default: False
- **suggestions_limit** (int) – How many suggestions to return. The result is a list of suggested playlist items (videos) contained in a "suggestions" key. 7 items are retrieved in each internal request. Default: 0

Return type
dict

Returns
Dictionary with information about the playlist. The key tracks contains a List of playlistItem dictionaries

The result is in the following format:

```
{
  "id": "PLQwVIlKxHM6qv-o99iX9R85og7IzF9YS_",
  "privacy": "PUBLIC",
  "title": "New EDM This Week 03/13/2020",
  "thumbnails": [...]
  "description": "Weekly r/EDM new release roundup. Created with github.com/sigma67/
↪spotifyplaylist_to_gmusic",
  "author": "sigmatics",
  "year": "2020",
  "duration": "6+ hours",
  "duration_seconds": 52651,
  "trackCount": 237,
  "suggestions": [
      {
        "videoId": "HLCsfOykA94",
        "title": "Mambo (GATTÜSO Remix)",
        "artists": [{
            "name": "Nikki Vianna",
            "id": "UCMW5eSIO1moVlIBLQzq4PnQ"
          }],
        "album": {
          "name": "Mambo (GATTÜSO Remix)",
          "id": "MPREb_jLeQJsd7U9w"
        },
        "likeStatus": "LIKE",
        "thumbnails": [...],
        "isAvailable": true,
        "isExplicit": false,
        "duration": "3:32",
        "duration_seconds": 212,
        "setVideoId": "to_be_updated_by_client"
      }
  ],
  "related": [
      {
        "title": "Presenting MYRNE",
        "playlistId": "RDCLAK5uy_mbdO3_xdD4NtU1rWI0OmvRSRZ8NH4uJCM",
        "thumbnails": [...],
        "description": "Playlist • YouTube Music"
      }
  ],
  "tracks": [
    {
      "videoId": "bjGppZKiuFE",
      "title": "Lost",
      "artists": [
        {
          "name": "Guest Who",
          "id": "UCkgCRdnnqWnUeIH7EIc3dBg"
        },
        {
          "name": "Kate Wild",
          "id": "UCwR2l3JfJbvB6aq0RnnJfWg"
```


(continues on next page)

(continued from previous page)

```
        }
      ],
      "album": {
        "name": "Lost",
        "id": "MPREb_PxmzvDuqOnC"
      },
      "duration": "2:58",
      "duration_seconds": 178,
      "setVideoId": "748EE8..."
      "likeStatus": "INDIFFERENT",
      "thumbnails": [...],
      "isAvailable": True,
      "isExplicit": False,
      "videoType": "MUSIC_VIDEO_TYPE_OMV",
      "feedbackTokens": {
        "add": "AB9zfpJxtvrU...",
        "remove": "AB9zfpKTyZ..."
    }
  ]
}
```

The setVideoId is the unique id of this playlist item and needed for moving/removing playlist items

YTMusic.**create_playlist**(*title: str*, *description: str*, *privacy_status: str = 'PRIVATE'*, *video_ids: list | None = None*, *source_playlist: str | None = None*) → str | dict

Creates a new empty playlist and returns its id.

Parameters

- **title** (str) – Playlist title
- **description** (str) – Playlist description
- **privacy_status** (str) – Playlists can be PUBLIC, PRIVATE, or UNLISTED. Default: PRIVATE
- **video_ids** (Optional[list]) – IDs of songs to create the playlist with
- **source_playlist** (Optional[str]) – Another playlist whose songs should be added to the new playlist

Return type
Union[str, dict]

Returns
ID of the YouTube playlist or full response if there was an error

YTMusic.**edit_playlist**(*playlistId: str*, *title: str | None = None*, *description: str | None = None*, *privacyStatus: str | None = None*, *moveItem: str | tuple[str, str] | None = None*, *addPlaylistId: str | None = None*, *addToTop: bool | None = None*) → str | dict

Edit title, description or privacyStatus of a playlist. You may also move an item within a playlist or append another playlist to this playlist.

Parameters

- **playlistId** (str) – Playlist id
- **title** (Optional[str]) – Optional. New title for the playlist

- **description** (Optional[str]) – Optional. New description for the playlist
- **privacyStatus** (Optional[str]) – Optional. New privacy status for the playlist
- **moveItem** (Union[str, tuple[str, str], None]) – Optional. Move one item before another. Items are specified by setVideoId, which is the unique id of this playlist item. See *get_playlist()*
- **addPlaylistId** (Optional[str]) – Optional. Id of another playlist to add to this playlist
- **addToTop** (Optional[bool]) – Optional. Change the state of this playlist to add items to the top of the playlist (if True) or the bottom of the playlist (if False - this is also the default of a new playlist).

Return type
Union[str, dict]

Returns
Status String or full response

YTMusic.**delete_playlist**(*playlistId: str*) → str | dict

Delete a playlist.

Parameters
playlistId (str) – Playlist id

Return type
Union[str, dict]

Returns
Status String or full response

YTMusic.**add_playlist_items**(*playlistId: str*, *videoIds: list[str] | None = None*, *source_playlist: str | None = None*, *duplicates: bool = False*) → str | dict

Add songs to an existing playlist

Parameters

- **playlistId** (str) – Playlist id
- **videoIds** (Optional[list[str]]) – List of Video ids
- **source_playlist** (Optional[str]) – Playlist id of a playlist to add to the current playlist (no duplicate check)
- **duplicates** (bool) – If True, duplicates will be added. If False, an error will be returned if there are duplicates (no items are added to the playlist)

Return type
Union[str, dict]

Returns
Status String and a dict containing the new setVideoId for each videoId or full response

YTMusic.**remove_playlist_items**(*playlistId: str*, *videos: list[dict]*) → str | dict

Remove songs from an existing playlist

Parameters

- **playlistId** (str) – Playlist id
- **videos** (list[dict]) – List of PlaylistItems, see *get_playlist()*. Must contain videoId and setVideoId

Return type
Union[str, dict]

Returns
Status String or full response

5.3.9 Podcasts

YTMusic.**get_channel**(*channelId: str*) → dict

Get information about a podcast channel (episodes, podcasts). For episodes, a maximum of 10 episodes are returned, the full list of episodes can be retrieved via *get_channel_episodes()*

Parameters
channelId (str) – channel id

Return type
dict

Returns
Dict containing channel info

Example:

```
{
    "title": 'Stanford Graduate School of Business',
    "thumbnails": [...]
    "episodes":
    {
        "browseId": "UCGwuxdEeCf0TIA2RbPOj-8g",
        "results":
        [
            {
                "index": 0,
                "title": "The Brain Gain: The Impact of Immigration on American␣
↪Innovation with Rebecca Diamond",
                "description": "Immigrants' contributions to America ...",
                "duration": "24 min",
                "videoId": "TS3Ovvk3VAA",
                "browseId": "MPEDTS3Ovvk3VAA",
                "videoType": "MUSIC_VIDEO_TYPE_PODCAST_EPISODE",
                "date": "Mar 6, 2024",
                "thumbnails": [...]
            },
        ],
        "params": "6gPiAUdxWUJXcFlCQ3BN..."
    },
    "podcasts":
    {
        "browseId": null,
        "results":
        [
            {
                "title": "Stanford GSB Podcasts",
                "channel":
                {
```

(continues on next page)

(continued from previous page)

```
                    "id": "UCGwuxdEeCf0TIA2RbPOj-8g",
                    "name": "Stanford Graduate School of Business"
                },
                "browseId": "MPSPPLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9",
                "podcastId": "PLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9",
                "thumbnails": [...]
            }
        ]
    }
}
```

YTMusic.**get_channel_episodes**(*channelId: str*, *params: str*) → list[dict]

Get all channel episodes. This endpoint is currently unlimited

Parameters

- **channelId** (str) – channelId of the user
- **params** (str) – params obtained by *get_channel()*

Return type
list[dict]

Returns
List of channel episodes in the format of *get_channel()* "episodes" key

YTMusic.**get_podcast**(*playlistId: str*, *limit: int | None = 100*) → dict

Returns podcast metadata and episodes

Note: To add a podcast to your library, you need to call `rate_playlist` on it

Parameters

- **playlistId** (str) – Playlist id
- **limit** (Optional[int]) – How many songs to return. None retrieves them all. Default: 100

Return type
dict

Returns
Dict with podcast information

Example:

```
{
    "author":
    {
        "name": "Stanford Graduate School of Business",
        "id": "UCGwuxdEeCf0TIA2RbPOj-8g"
    },
    "title": "Think Fast, Talk Smart: The Podcast",
    "description": "Join Matt Abrahams, a lecturer of...",
    "saved": false,
    "episodes":
```

(continues on next page)

(continued from previous page)


```
    [
        {
            "index": 0,
            "title": "132. Lean Into Failure: How to Make Mistakes That Work |␣
↪Think Fast, Talk Smart: Communication...",
            "description": "Effective and productive teams and...",
            "duration": "25 min",
            "videoId": "xAEGaW2my7E",
            "browseId": "MPEDxAEGaW2my7E",
            "videoType": "MUSIC_VIDEO_TYPE_PODCAST_EPISODE",
            "date": "Mar 5, 2024",
            "thumbnails": [...]
        }
    ]
}
```


YTMusic.**get_episode**(*videoId: str*) → dict

Retrieve episode data for a single episode

Note: To save an episode, you need to call `add_playlist_items` to add it to the `SE` (saved episodes) playlist.

Parameters

videoId (str) – browseId (MPED..) or videoId for a single episode

Return type

dict

Returns

Dict containing information about the episode

The description elements are based on a custom dataclass, not shown in the example below The description text items also contain "n" to indicate newlines, removed below due to RST issues

Example:


```
{
    "author":
    {
        "name": "Stanford GSB Podcasts",
        "id": "MPSPPLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9"
    },
    "title": "124. Making Meetings Me...",
    "date": "Jan 16, 2024",
    "duration": "25 min",
    "saved": false,
    "playlistId": "MPSPPLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9",
    "description":
    [
        {
            "text": "Delve into why people hate meetings, ... Karin Reed ("
        },
        {
```


(continues on next page)

(continued from previous page)


```
            "text": "https://speakerdynamics.com/team/",
            "url": "https://speakerdynamics.com/team/"
        },
        {
            "text": ")Chapters:("
        },
        {
            "text": "00:00",
            "seconds": 0
        },
        {
            "text": ") Introduction Host Matt Abrahams...("
        },
        {
            "text": "01:30",
            "seconds": 90
        },
    ]
}
```


YTMusic.**get_episodes_playlist**(*playlist_id: str = 'RDPN'*) → dict

Get all episodes in an episodes playlist. Currently the only known playlist is the "New Episodes" auto-generated playlist

Parameters
playlist_id (str) – Playlist ID, defaults to "RDPN", the id of the New Episodes playlist

Return type
dict

Returns
Dictionary in the format of *get_podcast()*

5.3.10 Uploads

YTMusic.**get_library_upload_songs**(*limit: int | None = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Returns a list of uploaded songs

Parameters

- **limit** (Optional[int]) – How many songs to return. None retrieves them all. Default: 25
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of songs to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of uploaded songs.

Each item is in the following format:

```
{
  "entityId": "t_po_CICr2crg7OWpchDpjPjrBA",
  "videoId": "Uise6RPKoek",
  "artists": [{
    'name': 'Coldplay',
    'id': 'FEmusic_library_privately_owned_artist_detaila_po_
↪CICr2crg7OWpchIIY29sZHBsYXk',
  }],
  "title": "A Sky Full Of Stars",
  "album": "Ghost Stories",
  "likeStatus": "LIKE",
  "thumbnails": [...]
}
```

YTMusic.**get_library_upload_artists**(*limit: int | None = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the artists of uploaded songs in the user's library.

Parameters

- **limit** (Optional[int]) – Number of artists to return. None retrieves them all. Default: 25
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of artists to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of artists as returned by *get_library_artists()*

YTMusic.**get_library_upload_albums**(*limit: int | None = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the albums of uploaded songs in the user's library.

Parameters

- **limit** (Optional[int]) – Number of albums to return. None retrives them all. Default: 25
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of albums to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of albums as returned by *get_library_albums()*

YTMusic.**get_library_upload_artist**(*browseId: str*, *limit: int = 25*) → list[dict]

Returns a list of uploaded tracks for the artist.

Parameters

- **browseId** (str) – Browse id of the upload artist, i.e. from *get_library_upload_songs()*
- **limit** (int) – Number of songs to return (increments of 25).

Return type
list[dict]

Returns
List of uploaded songs.

Example List:

```
[
  {
    "entityId": "t_po_CICr2crg7OWpchDKwoakAQ",
    "videoId": "Dtffhy8WJgw",
    "title": "Hold Me (Original Mix)",
    "artists": [
      {
        "name": "Jakko",
        "id": "FEmusic_library_privately_owned_artist_detaila_po_
↪CICr2crg7OWpchIFamFra28"
      }
    ],
    "album": null,
    "likeStatus": "LIKE",
    "thumbnails": [...]
  }
]
```

YTMusic.**get_library_upload_album**(*browseId: str*) → dict

Get information and tracks of an album associated with uploaded tracks

Parameters
browseId (str) – Browse id of the upload album, i.e. from i.e. from *get_library_upload_songs()*

Return type
dict

Returns
Dictionary with title, description, artist and tracks.

Example album:

```
{
  "title": "18 Months",
  "type": "Album",
  "thumbnails": [...],
  "trackCount": 7,
  "duration": "24 minutes",
  "audioPlaylistId": "MLPRb_po_55chars",
  "tracks": [
    {
      "entityId": "t_po_22chars",
      "videoId": "FVo-UZoPygI",
      "title": "Feel So Close",
      "duration": "4:15",
      "duration_seconds": 255,
      "artists": None,
```

(continues on next page)

(continued from previous page)

```
    "album": {
      "name": "18 Months",
      "id": "FEmusic_library_privately_owned_release_detailb_po_55chars"
    },
    "likeStatus": "INDIFFERENT",
    "thumbnails": None
  },
```

YTMusic.**upload_song**(*filepath: str*) → *ytmusicapi.enums.ResponseStatus* | requests.models.Response

Uploads a song to YouTube Music

Parameters
filepath (str) – Path to the music file (mp3, m4a, wma, flac or ogg)

Return type
Union[*ResponseStatus*, Response]

Returns
Status String or full response

YTMusic.**delete_upload_entity**(*entityId: str*) → str | dict

Deletes a previously uploaded song or album

Parameters
entityId (str) – The entity id of the uploaded song or album, e.g. retrieved from *get_library_upload_songs()*

Return type
Union[str, dict]

Returns
Status String or error

5.3.11 ytmusicapi

ytmusicapi package

Subpackages

ytmusicapi.auth package

Subpackages

ytmusicapi.auth.oauth package

Submodules

ytmusicapi.auth.oauth.credentials module

class ytmusicapi.auth.oauth.credentials.**Credentials**(*client_id: str*, *client_secret: str*) → None

Bases: ABC

Base class representation of YouTubeMusicAPI OAuth Credentials

client_id: str

client_secret: str

abstract get_code() → collections.abc.Mapping

Method for obtaining a new user auth code. First step of token creation.

Return type
Mapping

abstract refresh_token(*refresh_token: str*) → *ytmusicapi.auth.oauth.models.BaseTokenDict*

Method for requesting a new access token for a given refresh_token. Token must have been created by the same OAuth client.

Return type
BaseTokenDict

abstract token_from_code(*device_code: str*) → *ytmusicapi.auth.oauth.models.RefreshableTokenDict*

Method for verifying user auth code and conversion into a FullTokenDict.

Return type
RefreshableTokenDict

class ytmusicapi.auth.oauth.credentials.OAuthCredentials(*client_id: str*, *client_secret: str*, *session: requests.sessions.Session | None = None*, *proxies: dict | None = None*)

Bases: *Credentials*

Class for handling OAuth credential retrieval and refreshing.

client_id: str

client_secret: str

get_code() → *ytmusicapi.auth.oauth.models.AuthCodeDict*

Method for obtaining a new user auth code. First step of token creation.

Return type
AuthCodeDict

refresh_token(*refresh_token: str*) → *ytmusicapi.auth.oauth.models.BaseTokenDict*

Method for requesting a new access token for a given `refresh_token`. Token must have been created by the same OAuth client.

Parameters
refresh_token (str) – Corresponding `refresh_token` for a matching `access_token`. Obtained via

Return type
BaseTokenDict

token_from_code(*device_code: str*) → *ytmusicapi.auth.oauth.models.RefreshableTokenDict*

Method for verifying user auth code and conversion into a FullTokenDict.

Return type
RefreshableTokenDict

ytmusicapi.auth.oauth.exceptions module

exception ytmusicapi.auth.oauth.exceptions.**BadOAuthClient**

Bases: Exception

OAuth client request failure. Ensure provided client_id and secret are correct and YouTubeData API is enabled.

exception ytmusicapi.auth.oauth.exceptions.**UnauthorizedOAuthClient**

Bases: Exception

OAuth client lacks permissions for specified token. Token can only be refreshed by OAuth credentials used for its creation.

ytmusicapi.auth.oauth.models module

models for oauth authentication

class ytmusicapi.auth.oauth.models.**AuthCodeDict**

Bases: TypedDict

Keys for the json object obtained via code response during auth flow.

device_code: str

code obtained via user confirmation and oauth consent

expires_in: int

seconds from original request timestamp

interval: int

(?) "5" (?)

user_code: str

alphanumeric code user is prompted to enter as confirmation. formatted as XXX-XXX-XXX.

verification_url: str

base url for OAuth consent screen for user signin/confirmation

class ytmusicapi.auth.oauth.models.**BaseTokenDict**

Bases: TypedDict

Limited token. Does not provide a refresh token. Commonly obtained via a token refresh.

access_token: str

str to be used in Authorization header

expires_in: int

seconds until expiration from request timestamp

scope: Union[str, Literal['https://www.googleapis.com/auth/youtube']]

should be 'https://www.googleapis.com/auth/youtube'

token_type: Union[str, Literal['Bearer']]

should be 'Bearer'

class ytmusicapi.auth.oauth.models.**RefreshableTokenDict**

Bases: dict

Entire token. Including refresh. Obtained through token setup.

access_token: `str`

expires_at: `int`

UNIX epoch timestamp in seconds

expires_in: `int`

refresh_token: `str`

str used to obtain new access token upon expiration

scope: `Union[str, Literal['https://www.googleapis.com/auth/youtube']]`

token_type: `Union[str, Literal['Bearer']]`

ytmusicapi.auth.oauth.token module

class ytmusicapi.auth.oauth.token.**OAuthToken**(*scope: str | Literal['https://www.googleapis.com/auth/youtube']*, *token_type: str | Literal['Bearer']*, *access_token: str*, *refresh_token: str*, *expires_at: int = 0*, *expires_in: int = 0*) → None

Bases: *Token*

Wrapper for an OAuth token implementing expiration methods.

classmethod from_json(*file_path: pathlib.Path*) → *ytmusicapi.auth.oauth.token.OAuthToken*

Return type
OAuthToken

property is_expiring: `bool`

static is_oauth(*headers: requests.structures.CaseInsensitiveDict*) → bool

Return type
bool

update(*fresh_access:* ytmusicapi.auth.oauth.models.BaseTokenDict)

Update access_token and expiration attributes with a BaseTokenDict inplace. expires_at attribute set using current epoch, avoid expiration desync by passing only recently requested tokens dicts or updating values to compensate.

class ytmusicapi.auth.oauth.token.**RefreshingToken**(*scope: str | Literal['https://www.googleapis.com/auth/youtube']*, *token_type: str | Literal['Bearer']*, *access_token: str*, *refresh_token: str*, *expires_at: int = 0*, *expires_in: int = 0*, *credentials:* ytmusicapi.auth.oauth.credentials.Credentials *| None = None*, *_local_cache: pathlib.Path | None = None*) → None

Bases: *OAuthToken*

Compositional implementation of Token that automatically refreshes an underlying OAuthToken when required (credential expiration <= 1 min) upon access_token attribute access.

credentials: `Optional[Credentials] = None`

credentials used for access_token refreshing

property local_cache: Path | None

classmethod prompt_for_token(*credentials:* ytmusicapi.auth.oauth.credentials.Credentials, *open_browser: bool = False*, *to_file: str | None = None*) → *ytmusicapi.auth.oauth.token.RefreshingToken*

Method for CLI token creation via user inputs.

Parameters

- **credentials** (*Credentials*) – Client credentials
- **open_browser** (bool) – Optional. Open browser to OAuth consent url automatically. (Default: False).
- **to_file** (Optional[str]) – Optional. Path to store/sync json version of resulting token. (Default: None).

Return type
RefreshingToken

store_token(*path: str | None = None*) → None

Write token values to json file at specified path, defaulting to self.local_cache. Operation does not update instance local_cache attribute. Automatically called when local_cache is set post init.

Return type
None

class ytmusicapi.auth.oauth.token.**Token**(*scope: str | Literal['https://www.googleapis.com/auth/youtube']*, *token_type: str | Literal['Bearer']*, *access_token: str*, *refresh_token: str*, *expires_at: int = 0*, *expires_in: int = 0*) → None

Bases: object

Base class representation of the YouTubeMusicAPI OAuth token.

access_token: str

as_auth() → str

Returns Authorization header ready str of token_type and access_token.

Return type
str

as_dict() → *ytmusicapi.auth.oauth.models.RefreshableTokenDict*

Returns dictionary containing underlying token values.

Return type
RefreshableTokenDict

as_json() → str

Return type
str

expires_at: int = 0

expires_in: int = 0

property is_expiring: bool

static members()

refresh_token: str

scope: Union[str, Literal['https://www.googleapis.com/auth/youtube']]

token_type: Union[str, Literal['Bearer']]

Module contents

class ytmusicapi.auth.oauth.**OAuthCredentials**(*client_id: str*, *client_secret: str*, *session: requests.sessions.Session | None = None*, *proxies: dict | None = None*)

Bases: *Credentials*

Class for handling OAuth credential retrieval and refreshing.

client_id: str

client_secret: str

get_code() → *ytmusicapi.auth.oauth.models.AuthCodeDict*

Method for obtaining a new user auth code. First step of token creation.

Return type
AuthCodeDict

refresh_token(*refresh_token: str*) → *ytmusicapi.auth.oauth.models.BaseTokenDict*

Method for requesting a new access token for a given `refresh_token`. Token must have been created by the same OAuth client.

Parameters
refresh_token (str) – Corresponding `refresh_token` for a matching `access_token`. Obtained via

Return type
BaseTokenDict

token_from_code(*device_code: str*) → *ytmusicapi.auth.oauth.models.RefreshableTokenDict*

Method for verifying user auth code and conversion into a FullTokenDict.

Return type
RefreshableTokenDict

class ytmusicapi.auth.oauth.**OAuthToken**(*scope: str | Literal['https://www.googleapis.com/auth/youtube']*, *token_type: str | Literal['Bearer']*, *access_token: str*, *refresh_token: str*, *expires_at: int = 0*, *expires_in: int = 0*) → None

Bases: *Token*

Wrapper for an OAuth token implementing expiration methods.

classmethod from_json(*file_path: pathlib.Path*) → *ytmusicapi.auth.oauth.token.OAuthToken*

Return type
OAuthToken

property is_expiring: bool

static is_oauth(*headers: requests.structures.CaseInsensitiveDict*) → bool

Return type
bool

update(*fresh_access:* ytmusicapi.auth.oauth.models.BaseTokenDict)

Update access_token and expiration attributes with a BaseTokenDict inplace. expires_at attribute set using current epoch, avoid expiration desync by passing only recently requested tokens dicts or updating values to compensate.

class ytmusicapi.auth.oauth.**RefreshingToken**(*scope: str | Literal['https://www.googleapis.com/auth/youtube']*, *token_type: str | Literal['Bearer']*, *access_token: str*, *refresh_token: str*, *expires_at: int = 0*, *expires_in: int = 0*, *credentials:* ytmusicapi.auth.oauth.credentials.Credentials | *None = None*, *_local_cache: pathlib.Path | None = None*) → None

Bases: *OAuthToken*

Compositional implementation of Token that automatically refreshes an underlying OAuthToken when required (credential expiration <= 1 min) upon access_token attribute access.

access_token: str

credentials: Optional[*Credentials*] = None

credentials used for access_token refreshing

property local_cache: Path | None

classmethod prompt_for_token(*credentials:* ytmusicapi.auth.oauth.credentials.Credentials, *open_browser: bool = False*, *to_file: str | None = None*) → *ytmusicapi.auth.oauth.token.RefreshingToken*

Method for CLI token creation via user inputs.

Parameters

- **credentials** (*Credentials*) – Client credentials
- **open_browser** (bool) – Optional. Open browser to OAuth consent url automatically. (Default: False).
- **to_file** (Optional[str]) – Optional. Path to store/sync json version of resulting token. (Default: None).

Return type
RefreshingToken

refresh_token: str

scope: Union[str, Literal['https://www.googleapis.com/auth/youtube']]

store_token(*path: str | None = None*) → None

Write token values to json file at specified path, defaulting to self.local_cache. Operation does not update instance local_cache attribute. Automatically called when local_cache is set post init.

Return type
None

token_type: Union[str, Literal['Bearer']]

Submodules

ytmusicapi.auth.auth_parse module

ytmusicapi.auth.auth_parse.**determine_auth_type**(*auth_headers: requests.structures.CaseInsensitiveDict*) → *ytmusicapi.auth.types.AuthType*

Determine the type of auth based on auth headers.

Parameters
auth_headers (CaseInsensitiveDict) – auth headers dict

Return type
AuthType

Returns
AuthType enum

ytmusicapi.auth.auth_parse.**parse_auth_str**(*auth: str | dict*) → tuple[requests.structures.CaseInsensitiveDict, Optional[pathlib.Path]]

Parameters
auth (Union[str, dict]) – user-provided auth string or dict

Return type
tuple[CaseInsensitiveDict, Optional[Path]]

Returns
parsed header dict based on auth, optionally path to file if it auth was a path to a file

ytmusicapi.auth.browser module

ytmusicapi.auth.browser.**is_browser**(*headers: requests.structures.CaseInsensitiveDict*) → bool

Return type
bool

ytmusicapi.auth.browser.**setup_browser**(*filepath: str | None = None*, *headers_raw: str | None = None*) → str

Return type
str

ytmusicapi.auth.types module

enum representing types of authentication supported by this library

class ytmusicapi.auth.types.**AuthType**(*value*)

Bases: int, Enum

enum representing types of authentication supported by this library

BROWSER = 2

OAUTH_CUSTOM_CLIENT = 3

YTM instance is using a non-default OAuth client (id & secret)

OAUTH_CUSTOM_FULL = 4

allows fully formed OAuth headers to ignore browser auth refresh flow

UNAUTHORIZED = 1

Module contents

ytmusicapi.mixins package

Submodules

ytmusicapi.mixins.browsing module

class ytmusicapi.mixins.browsing.**BrowsingMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

ArtistOrderType

alias of Literal['Recency', 'Popularity', 'Alphabetical order']

get_album(*browseId: str*) → dict

Get information and tracks of an album

Parameters
browseId (str) – browseId of the album, for example returned by search()

Return type
dict

Returns
Dictionary with album and track metadata.

The result is in the following format:


```
{
  "title": "Revival",
  "type": "Album",
  "thumbnails": [],
  "description": "Revival is the...",
  "artists": [
    {
      "name": "Eminem",
      "id": "UCedvOgsKFzcK3hA5taf3KoQ"
    }
  ],
  "year": "2017",
  "trackCount": 19,
  "duration": "1 hour, 17 minutes",
  "audioPlaylistId": "OLAK5uy_nMr9h2VlS-2PULNz3M3XVXQj_P3C2bqaY",
  "tracks": [
    {
      "videoId": "iKLU7z_xdYQ",
      "title": "Walk On Water (feat. Beyoncé)",
      "artists": [
        {
```


(continues on next page)

(continued from previous page)


```
          "name": "Eminem",
          "id": "UCedvOgsKFzcK3hA5taf3KoQ"
        }
      ],
      "album": "Revival",
      "likeStatus": "INDIFFERENT",
      "thumbnails": null,
      "isAvailable": true,
      "isExplicit": true,
      "duration": "5:03",
      "duration_seconds": 303,
      "trackNumber": 0,
      "feedbackTokens": {
        "add": "AB9zfpK...",
        "remove": "AB9zfpK..."
      }
    }
  ],
  "other_versions": [
    {
      "title": "Revival",
      "year": "Eminem",
      "browseId": "MPREb_fefKFOTEZSp",
      "thumbnails": [...],
      "isExplicit": false
    },
  ],
  "duration_seconds": 4657
}
```


get_album_browse_id(*audioPlaylistId: str*) → str | None

Get an album's browseId based on its audioPlaylistId

Parameters
audioPlaylistId (str) – id of the audio playlist (starting with *OLAK5uy_*)

Return type
Optional[str]

Returns
browseId (starting with MPREb_)

get_artist(*channelId: str*) → dict

Get information about an artist and their top releases (songs, albums, singles, videos, and related artists). The top lists contain pointers for getting the full list of releases.

Possible content types for get_artist are:

- songs
- albums
- singles
- shows
- videos

- episodes
- podcasts
- related

Each of these content keys in the response contains `results` and possibly `browseId` and `params`.

- For songs/videos, pass the browseId to `get_playlist()`.
- For albums/singles/shows, pass browseId and params to *get_artist_albums()*.

Parameters
channelId (str) – channel id of the artist

Return type
dict

Returns
Dictionary with requested information.

Warning: The returned channelId is not the same as the one passed to the function. It should be used only with `subscribe_artists()`.

Example:

```
{
    "description": "Oasis were ...",
    "views": "3,693,390,359 views",
    "name": "Oasis",
    "channelId": "UCUDVBtnOQi4c7E8jebpjc9Q",
    "shuffleId": "RDAOkjHYJjL1a3xspEyVkhHAsg",
    "radioId": "RDEMkjHYJjL1a3xspEyVkhHAsg",
    "subscribers": "3.86M",
    "subscribed": false,
    "thumbnails": [...],
    "songs": {
        "browseId": "VLPLMpM3Z0118S42R1npOhcjoakLIv1aqnS1",
        "results": [
            {
                "videoId": "ZrOKjDZOtkA",
                "title": "Wonderwall (Remastered)",
                "thumbnails": [...],
                "artist": "Oasis",
                "album": "(What's The Story) Morning Glory? (Remastered)"
            }
        ]
    },
    "albums": {
        "results": [
            {
                "title": "Familiar To Millions",
                "thumbnails": [...],
                "year": "2018",
                "browseId": "MPREb_AYetWMZunqA"
```

(continues on next page)

(continued from previous page)


```
            }
        ],
        "browseId": "UCmMUZbaYdNH0bEd1PAlAqsA",
        "params": "6gPTAUNwc0JDbndLYlFBQV..."
    },
    "singles": {
        "results": [
            {
                "title": "Stand By Me (Mustique Demo)",
                "thumbnails": [...],
                "year": "2016",
                "browseId": "MPREb_7MPKLhibN5G"
            }
        ],
        "browseId": "UCmMUZbaYdNH0bEd1PAlAqsA",
        "params": "6gPTAUNwc0JDbndLYlFBQV..."
    },
    "videos": {
        "results": [
            {
                "title": "Wonderwall",
                "thumbnails": [...],
                "views": "358M",
                "videoId": "bx1Bh8ZvH84",
                "playlistId": "PLMpM3Z0118S5xuNckw1HUcj1D021AnMEB"
            }
        ],
        "browseId": "VLPLMpM3Z0118S5xuNckw1HUcj1D021AnMEB"
    },
    "related": {
        "results": [
            {
                "browseId": "UCt2KxZpY5D__kapeQ8cauQw",
                "subscribers": "450K",
                "title": "The Verve"
            },
            {
                "browseId": "UCwK2Grm574W1u-sBzLikldQ",
                "subscribers": "341K",
                "title": "Liam Gallagher"
            },
            ...
        ]
    }
}
```


get_artist_albums(*channelId: str*, *params: str*, *limit: int | None = 100*, *order: Literal['Recency', 'Popularity', 'Alphabetical order'] | None = None*) → list[dict]

Get the full list of an artist's albums, singles or shows

Parameters

- **channelId** (str) – browseId of the artist as returned by *get_artist()*

- **params** (str) – params obtained by *get_artist()*
- **limit** (Optional[int]) – Number of albums to return. None retrieves them all. Default: 100
- **order** (Optional[Literal['Recency', 'Popularity', 'Alphabetical order']]) – Order of albums to return. Allowed values: Recency, Popularity, *Alphabetical order*. Default: Default order.

Return type
list[dict]

Returns
List of albums in the format of get_library_albums(), except artists key is missing.

get_basejs_url() → str

Extract the URL for the *base.js* script from YouTube Music.

Return type
str

Returns
URL to *base.js*

get_home(*limit=3*) → list[dict]

Get the home page. The home page is structured as titled rows, returning 3 rows of music suggestions at a time. Content varies and may contain artist, album, song or playlist suggestions, sometimes mixed within the same row

Parameters
limit – Number of rows to return

Return type
list[dict]

Returns
List of dictionaries keyed with 'title' text and 'contents' list

Example list:

```
[
    {
        "title": "Your morning music",
        "contents": [
            { //album result
                "title": "Sentiment",
                "browseId": "MPREb_QtqXtd2xZMR",
                "thumbnails": [...]
            },
            { //playlist result
                "title": "r/EDM top submissions 01/28/2022",
                "playlistId": "PLz7-xrYmULdSLRZGk-6GKUtaBZcgQNwel",
                "thumbnails": [...],
                "description": "redditEDM • 161 songs",
                "count": "161",
                "author": [
                    {
                        "name": "redditEDM",
                        "id": "UCaTrZ9tPiIGHrkCe5bxOGwA"
```

(continues on next page)

(continued from previous page)

```
                    }
                ]
            }
        ]
    },
    {
        "title": "Your favorites",
        "contents": [
            { //artist result
                "title": "Chill Satellite",
                "browseId": "UCrPLFBWdOroD57bkqPbZJog",
                "subscribers": "374",
                "thumbnails": [...]
            }
            { //album result
                "title": "Dragon",
                "year": "Two Steps From Hell",
                "browseId": "MPREb_M9aDqLRbSeg",
                "thumbnails": [...]
            }
        ]
    },
    {
        "title": "Quick picks",
        "contents": [
            { //song quick pick
                "title": "Gravity",
                "videoId": "EludZd6lfts",
                "artists": [{
                        "name": "yetep",
                        "id": "UCSW0r7dClqCoCvQeqXiZBlg"
                    }],
                "thumbnails": [...],
                "album": {
                    "name": "Gravity",
                    "id": "MPREb_D6bICFcuuRY"
                }
            },
            { //video quick pick
                "title": "Gryffin & Illenium (feat. Daya) - Feel Good (L3V3LS␣
↪Remix)",
                "videoId": "bR5l0hJDnX8",
                "artists": [
                    {
                        "name": "L3V3LS",
                        "id": "UCCVNihbOdkOWw_-ajIYhAbQ"
                    }
                ],
                "thumbnails": [...],
                "views": "10M"
            }
        ]
```

(continues on next page)

(continued from previous page)

```
    }
]
```

get_lyrics(*browseId: str*, *timestamps: bool | None = False*) → *ytmusicapi.models.lyrics.Lyrics* | *ytmusicapi.models.lyrics.TimedLyrics* | None

Returns lyrics of a song or video. When *timestamps* is set, lyrics are returned with timestamps, if available.

Parameters

- **browseId** (str) – Lyrics browseId obtained from `get_watch_playlist()` (startswith `MPLYt...`).
- **timestamps** (Optional[bool]) – Optional. Whether to return bare lyrics or lyrics with timestamps, if available. (Default: *False*)

Return type

Union[*Lyrics*, *TimedLyrics*, None]

Returns

Dictionary with song lyrics or None, if no lyrics are found. The hasTimestamps-key determines the format of the data.

Example when *timestamps*=*False*, or no timestamps are available:

```
{
    "lyrics": "Today is gonna be the day\nThat they're gonna throw it␣
↪back to you\n",
    "source": "Source: LyricFind",
    "hasTimestamps": False
}
```

Example when *timestamps* is set to *True* and timestamps are available:

```
{
    "lyrics": [
        LyricLine(
            text="I was a liar",
            start_time=9200,
            end_time=10630,
            id=1
        ),
        LyricLine(
            text="I gave in to the fire",
            start_time=10680,
            end_time=12540,
            id=2
        ),
    ],
    "source": "Source: LyricFind",
    "hasTimestamps": True
}
```

get_signatureTimestamp(*url: str | None = None*) → int

Fetch the *base.js* script from YouTube Music and parse out the signatureTimestamp for use with *get_song()*.

Parameters
url (Optional[str]) – Optional. Provide the URL of the *base.js* script. If this isn't specified a call will be made to *get_basejs_url()*.

Return type
int

Returns
signatureTimestamp string

get_song(*videoId: str*, *signatureTimestamp: int | None = None*) → dict
Returns metadata and streaming information about a song or video.

Parameters
- **videoId** (str) – Video id
- **signatureTimestamp** (Optional[int]) – Provide the current YouTube signatureTimestamp. If not provided a default value will be used, which might result in invalid streaming URLs

Return type
dict

Returns
Dictionary with song metadata.

Example:


```
{
    "playabilityStatus": {
        "status": "OK",
        "playableInEmbed": true,
        "audioOnlyPlayability": {
            "audioOnlyPlayabilityRenderer": {
                "trackingParams": "CAEQx2kiEwiuv9X5i5H1AhWBvlUKHRoZAHk=",
                "audioOnlyAvailability": "FEATURE_AVAILABILITY_ALLOWED"
            }
        },
        "miniplayer": {
            "miniplayerRenderer": {
                "playbackMode": "PLAYBACK_MODE_ALLOW"
            }
        },
        "contextParams": "Q0FBU0FnZ0M="
    },
    "streamingData": {
        "expiresInSeconds": "21540",
        "adaptiveFormats": [
            {
                "itag": 140,
                "url": "https://rr1---sn-h0jelnez.c.youtube.com/videoplayback?
↪expire=1641080272...",
                "mimeType": "audio/mp4; codecs="mp4a.40.2"",
                "bitrate": 131007,
                "initRange": {
                    "start": "0",
                    "end": "667"
```


(continues on next page)

(continued from previous page)


```
                },
                "indexRange": {
                    "start": "668",
                    "end": "999"
                },
                "lastModified": "1620321966927796",
                "contentLength": "3967382",
                "quality": "tiny",
                "projectionType": "RECTANGULAR",
                "averageBitrate": 129547,
                "highReplication": true,
                "audioQuality": "AUDIO_QUALITY_MEDIUM",
                "approxDurationMs": "245000",
                "audioSampleRate": "44100",
                "audioChannels": 2,
                "loudnessDb": -1.3000002
            }
        ]
    },
    "playbackTracking": {
        "videostatsPlaybackUrl": {
          "baseUrl": "https://s.youtube.com/api/stats/playback?cl=491307275&
↪docid=AjXQiKP5kMs&ei=Nl2HY-6MH5WE8gPjnYnoDg&fexp=1714242%2C9405963%2C23804281
↪%2C23858057%2C23880830%2C23880833%2C23882685%2C23918597%2C23934970%2C23946420
↪%2C23966208%2C23983296%2C23998056%2C24001373%2C24002022%2C24002025%2C24004644
↪%2C24007246%2C24034168%2C24036947%2C24077241%2C24080738%2C24120820%2C24135310
↪%2C24135692%2C24140247%2C24161116%2C24162919%2C24164186%2C24169501%2C24175560
↪%2C24181174%2C24187043%2C24187377%2C24187854%2C24191629%2C24197450%2C24199724
↪%2C24200839%2C24209349%2C24211178%2C24217535%2C24219713%2C24224266%2C24241378
↪%2C24248091%2C24248956%2C24255543%2C24255545%2C24262346%2C24263796%2C24265426
↪%2C24267564%2C24268142%2C24279196%2C24280220%2C24283426%2C24283493%2C24287327
↪%2C24288045%2C24290971%2C24292955%2C24293803%2C24299747%2C24390674%2C24391018
↪%2C24391537%2C24391709%2C24392268%2C24392363%2C24392401%2C24401557%2C24402891
↪%2C24403794%2C24406605%2C24407200%2C24407665%2C24407914%2C24408220%2C24411766
↪%2C24413105%2C24413820%2C24414162%2C24415866%2C24416354%2C24420756%2C24421162
↪%2C24425861%2C24428962%2C24590921%2C39322504%2C39322574%2C39322694%2C39322707&
↪ns=yt&plid=AAXusD4TIOMjS5N4&el=detailpage&len=246&of=Jx1iRksbq-rB9N1KSijZLQ&
↪osid=MWU2NzBjYTI%3AAOeUNAagU8UyWDUJIki5raGHy29-60-yTA&uga=29&
↪vm=CAEQABgEOjJBUEV3RWxUNmYzMXNMMC1MYVpCVnRZTmZWMWw1OWVZX2ZOcUtCSkphQ245VFZwOXdTQWJbQVBta0tET
↪",
          "headers": [
            {
              "headerType": "USER_AUTH"
            },
            {
              "headerType": "VISITOR_ID"
            },
            {
              "headerType": "PLUS_PAGE_ID"
            }
          ]
        },
```


(continues on next page)

(continued from previous page)


```
        "videostatsDelayplayUrl": {(as above)},
        "videostatsWatchtimeUrl": {(as above)},
        "ptrackingUrl": {(as above)},
        "qoeUrl": {(as above)},
        "atrUrl": {(as above)},
        "videostatsScheduledFlushWalltimeSeconds": [
          10,
          20,
          30
        ],
        "videostatsDefaultFlushIntervalSeconds": 40
    },
    "videoDetails": {
        "videoId": "AjXQiKP5kMs",
        "title": "Sparks",
        "lengthSeconds": "245",
        "channelId": "UCvCk2zFqkCYzpnSgWfx0qOg",
        "isOwnerViewing": false,
        "isCrawlable": false,
        "thumbnail": {
            "thumbnails": []
        },
        "allowRatings": true,
        "viewCount": "12",
        "author": "Thomas Bergersen",
        "isPrivate": true,
        "isUnpluggedCorpus": false,
        "musicVideoType": "MUSIC_VIDEO_TYPE_PRIVATELY_OWNED_TRACK",
        "isLiveContent": false
    },
    "microformat": {
        "microformatDataRenderer": {
            "urlCanonical": "https://music.youtube.com/watch?v=AjXQiKP5kMs",
            "title": "Sparks - YouTube Music",
            "description": "Uploaded to YouTube via YouTube Music Sparks",
            "thumbnail": {
                "thumbnails": [
                    {
                        "url": "https://i.ytimg.com/vi/AjXQiKP5kMs/hqdefault.jpg
↪",
                        "width": 480,
                        "height": 360
                    }
                ]
            },
            "siteName": "YouTube Music",
            "appName": "YouTube Music",
            "androidPackage": "com.google.android.apps.youtube.music",
            "iosAppStoreId": "1017492454",
            "iosAppArguments": "https://music.youtube.com/watch?v=AjXQiKP5kMs",
            "ogType": "video.other",
            "urlApplinksIos": "vnd.youtube.music://music.youtube.com/watch?
```


(continues on next page)

(continued from previous page)


```
↪v=AjXQiKP5kMs&feature=applinks",
            "urlApplinksAndroid": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=applinks",
            "urlTwitterIos": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=twitter-deep-link",
            "urlTwitterAndroid": "vnd.youtube.music://music.youtube.com/watch?
↪v=AjXQiKP5kMs&feature=twitter-deep-link",
            "twitterCardType": "player",
            "twitterSiteHandle": "@YouTubeMusic",
            "schemaDotOrgType": "http://schema.org/VideoObject",
            "noindex": true,
            "unlisted": true,
            "paid": false,
            "familySafe": true,
            "pageOwnerDetails": {
                "name": "Music Library Uploads",
                "externalChannelId": "UCvCk2zFqkCYzpnSgWfx0qOg",
                "youtubeProfileUrl": "http://www.youtube.com/channel/
↪UCvCk2zFqkCYzpnSgWfx0qOg"
            },
            "videoDetails": {
                "externalVideoId": "AjXQiKP5kMs",
                "durationSeconds": "246",
                "durationIso8601": "PT4M6S"
            },
            "linkAlternates": [
                {
                    "hrefUrl": "android-app://com.google.android.youtube/http/
↪youtube.com/watch?v=AjXQiKP5kMs"
                },
                {
                    "hrefUrl": "ios-app://544007664/http/youtube.com/watch?
↪v=AjXQiKP5kMs"
                },
                {
                    "hrefUrl": "https://www.youtube.com/oembed?format=json&
↪url=https%3A%2F%2Fmusic.youtube.com%2Fwatch%3Fv%3DAjXQiKP5kMs",
                    "title": "Sparks",
                    "alternateType": "application/json+oembed"
                },
                {
                    "hrefUrl": "https://www.youtube.com/oembed?format=xml&
↪url=https%3A%2F%2Fmusic.youtube.com%2Fwatch%3Fv%3DAjXQiKP5kMs",
                    "title": "Sparks",
                    "alternateType": "text/xml+oembed"
                }
            ],
            "viewCount": "12",
            "publishDate": "1969-12-31",
            "category": "Music",
            "uploadDate": "1969-12-31"
        }
```


(continues on next page)

(continued from previous page)

```
    }
}
```

get_song_related(*browseId: str*)

Gets related content for a song. Equivalent to the content shown in the "Related" tab of the watch panel.

Parameters

browseId (str) – The `related` key in the `get_watch_playlist` response.

Example:

```
[
  {
    "title": "You might also like",
    "contents": [
      {
        "title": "High And Dry",
        "videoId": "7fv84nPfTH0",
        "artists": [{
            "name": "Radiohead",
            "id": "UCr_iyUANcn9OX_yy9piYoLw"
          }],
        "thumbnails": [
          {
            "url": "https://lh3.googleusercontent.com/
↪TWWT47cHLv3yAugk4h9eOzQ46FHmXc_g-KmBVy2d4sbg_F-Gv6xrPglztRVzp8D_l-yzOnvh-
↪QToM8s=w60-h60-l90-rj",
            "width": 60,
            "height": 60
          }
        ],
        "isExplicit": false,
        "album": {
          "name": "The Bends",
          "id": "MPREb_xsmDKhqhQrG"
        }
      }
    ]
  },
  {
    "title": "Recommended playlists",
    "contents": [
      {
        "title": "'90s Alternative Rock Hits",
        "playlistId": "RDCLAK5uy_m_h-nx7OCFaq9AlyXv78lG0AuloqW_NUA",
        "thumbnails": [...],
        "description": "Playlist • YouTube Music"
      }
    ]
  },
  {
    "title": "Similar artists",
    "contents": [
```

(continues on next page)

(continued from previous page)

```
      {
        "title": "Noel Gallagher",
        "browseId": "UCu7yYcX_wIZgG9azR3PqrxA",
        "subscribers": "302K",
        "thumbnails": [...]
      }
    ]
  },
  {
    "title": "Oasis",
    "contents": [
      {
        "title": "Shakermaker",
        "year": "2014",
        "browseId": "MPREb_WNGQWp5czjD",
        "thumbnails": [...]
      }
    ]
  },
  {
    "title": "About the artist",
    "contents": "Oasis were a rock band consisting of Liam Gallagher, Paul ...␣
↪(full description shortened for documentation)"
  }
]
```

get_tasteprofile() → dict

Fetches suggested artists from taste profile (music.youtube.com/tasteprofile). Tasteprofile allows users to pick artists to update their recommendations. Only returns a list of suggested artists, not the actual list of selected entries

Return type
dict

Returns
Dictionary with artist and their selection & impression value

Example:

```
{
    "Drake": {
        "selectionValue": "tastebuilder_selection=/m/05mt_q"
        "impressionValue": "tastebuilder_impression=/m/05mt_q"
    }
}
```

get_user(*channelId: str*) → dict

Retrieve a user's page. A user may own videos or playlists.

Use *get_user_playlists()* to retrieve all playlists:

```
result = get_user(channelId)
get_user_playlists(channelId, result["playlists"]["params"])
```

Similarly, use *get_user_videos()* to retrieve all videos:

```
get_user_videos(channelId, result["videos"]["params"])
```

Parameters
channelId (str) – channelId of the user

Return type
dict

Returns
Dictionary with information about a user.

Example:


```
{
  "name": "4Tune - No Copyright Music",
  "videos": {
    "browseId": "UC44hbeRoCZVVMVg5z0FfIww",
    "results": [
      {
        "title": "Epic Music Soundtracks 2019",
        "videoId": "bJonJjgS2mM",
        "playlistId": "RDAMVMbJonJjgS2mM",
        "thumbnails": [
          {
            "url": "https://i.ytimg.com/vi/bJon...",
            "width": 800,
            "height": 450
          }
        ],
        "views": "19K"
      }
    ]
  },
  "playlists": {
    "browseId": "UC44hbeRoCZVVMVg5z0FfIww",
    "results": [
      {
        "title": " Machinimasound | Playlist",
        "playlistId": "PLRm766YvPiO9ZqkBuEzSTt6Bk4eWIr3gB",
        "thumbnails": [
          {
            "url": "https://i.ytimg.com/vi/...",
            "width": 400,
            "height": 225
          }
        ]
      }
    ],
    "params": "6gO3AUNvWU..."
  }
}
```


get_user_playlists(*channelId: str*, *params: str*) → list[dict]

Retrieve a list of playlists for a given user. Call this function again with the returned `params` to get the full

list.

Parameters

- **channelId** (str) – channelId of the user.
- **params** (str) – params obtained by *get_user()*

Return type
list[dict]

Returns
List of user playlists in the format of get_library_playlists()

get_user_videos(*channelId: str*, *params: str*) → list[dict]

Retrieve a list of videos for a given user. Call this function again with the returned params to get the full list.

Parameters

- **channelId** (str) – channelId of the user.
- **params** (str) – params obtained by *get_user()*

Return type
list[dict]

Returns
List of user videos

set_tasteprofile(*artists: list[str]*, *taste_profile: dict | None = None*) → None

Favorites artists to see more recommendations from the artist. Use *get_tasteprofile()* to see which artists are available to be recommended

Parameters

- **artists** (list[str]) – A List with names of artists, must be contained in the tasteprofile
- **taste_profile** (Optional[dict]) – tasteprofile result from *get_tasteprofile()*. Pass this if you call *get_tasteprofile()* anyway to save an extra request.

Return type
None

Returns
None if successful

ytmusicapi.mixins.explore module

class ytmusicapi.mixins.explore.**ExploreMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

get_charts(*country: str = 'ZZ'*) → dict

Get latest charts data from YouTube Music: Top songs, top videos, top artists and top trending videos. Global charts have no Trending section, US charts have an extra Genres section with some Genre charts.

Parameters
country (str) – ISO 3166-1 Alpha-2 country code. Default: ZZ = Global

Return type
dict

Returns

Dictionary containing chart songs (only if authenticated with premium account), chart videos, chart artists and trending videos.

Example:


```
{
    "countries": {
        "selected": {
            "text": "United States"
        },
        "options": ["DE",
            "ZZ",
            "ZW"]
    },
    "songs": {
        "playlist": "VLPL4fGSI1pDJn6O1LS0XSdF3RyO0Rq_LDeI",
        "items": [
            {
                "title": "Outside (Better Days)",
                "videoId": "oT79YlRtXDg",
                "artists": [
                    {
                        "name": "MO3",
                        "id": "UCdFt4Cvhr7Okaxo6hZg5K8g"
                    },
                    {
                        "name": "OG Bobby Billions",
                        "id": "UCLusb4T2tW3gOpJS1fJ-A9g"
                    }
                ],
                "thumbnails": [...],
                "isExplicit": true,
                "album": {
                    "name": "Outside (Better Days)",
                    "id": "MPREb_fX4Yv8frUNv"
                },
                "rank": "1",
                "trend": "up"
            }
        ]
    },
    "videos": {
        "playlist": "VLPL4fGSI1pDJn69On1f-8NAvX_CYlx7QyZc",
        "items": [
            {
                "title": "EVERY CHANCE I GET (Official Music Video) (feat. Lil␣
↪Baby & Lil Durk)",
                "videoId": "BTivsHlVcGU",
                "playlistId": "PL4fGSI1pDJn69On1f-8NAvX_CYlx7QyZc",
                "thumbnails": [],
                "views": "46M"
            }
        ]
```


(continues on next page)

(continued from previous page)


```
    },
    "artists": {
        "playlist": null,
        "items": [
            {
                "title": "YoungBoy Never Broke Again",
                "browseId": "UCR28YDxjDE3ogQROaNdnRbQ",
                "subscribers": "9.62M",
                "thumbnails": [],
                "rank": "1",
                "trend": "neutral"
            }
        ]
    },
    "genres": [
        {
            "title": "Top 50 Pop Music Videos United States",
            "playlistId": "PL4fGSI1pDJn77aK7sAW2AT0oOzo5inWY8",
            "thumbnails": []
        }
    ],
    "trending": {
        "playlist": "VLPLrEnWoR732-DtKgaDdnPkezM_nDidBU9H",
        "items": [
            {
                "title": "Permission to Dance",
                "videoId": "CuklIb9d3fI",
                "playlistId": "PLrEnWoR732-DtKgaDdnPkezM_nDidBU9H",
                "artists": [
                    {
                        "name": "BTS",
                        "id": "UC9vrvNSL3xcWGSkV86REBSg"
                    }
                ],
                "thumbnails": [],
                "views": "108M"
            }
        ]
    }
}
```


get_mood_categories() → dict

Fetch "Moods & Genres" categories from YouTube Music.

Return type
dict

Returns
Dictionary of sections and categories.

Example:


```
{
```


(continues on next page)

(continued from previous page)

```
    'For you': [
        {
            'params': 'ggMPOg1uX1ZwN0pHT2NBT1Fk',
            'title': '1980s'
        },
        {
            'params': 'ggMPOg1uXzZQbDB5eThLRTQ3',
            'title': 'Feel Good'
        },
        ...
    ],
    'Genres': [
        {
            'params': 'ggMPOg1uXzVLbmZnaWI4STNs',
            'title': 'Dance & Electronic'
        },
        {
            'params': 'ggMPOg1uX3NjZllsNGVEMkZo',
            'title': 'Decades'
        },
        ...
    ],
    'Moods & moments': [
        {
            'params': 'ggMPOg1uXzVuc0dnZlhpV3Ba',
            'title': 'Chill'
        },
        {
            'params': 'ggMPOg1uX2ozUHlwbWM3ajNq',
            'title': 'Commute'
        },
        ...
    ],
}
```

get_mood_playlists(*params: str*) → list[dict]

Retrieve a list of playlists for a given "Moods & Genres" category.

Parameters

params (str) – params obtained by *get_mood_categories()*

Return type

list[dict]

Returns

List of playlists in the format of `get_library_playlists()`

ytmusicapi.mixins.library module

class ytmusicapi.mixins.library.**LibraryMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

add_history_item(*song*)

Add an item to the account's history using the playbackTracking URI obtained from `get_song()`. A `204` return code indicates success.

Usage:

```
song = yt_auth.get_song(videoId)
response = yt_auth.add_history_item(song)
```

Note: You need to use the same YTMusic instance as you used for `get_song()`.

Parameters
song – Dictionary as returned by `get_song()`

Returns
Full response. response.status_code is 204 if successful

edit_song_library_status(*feedbackTokens: list[str] | None = None*) → dict

Adds or removes a song from your library depending on the token provided.

Parameters
feedbackTokens (`Optional[list[str]]`) – List of feedbackTokens obtained from authenticated requests to endpoints that return songs (i.e. `get_album()`)

Return type
`dict`

Returns
Full response

get_account_info() → dict

Gets information about the currently authenticated user's account.

Return type
`dict`

Returns
Dictionary with user's account name, channel handle, and URL of their account photo.

Example:

```
{
    "accountName": "Sample User",
    "channelHandle": "@SampleUser
    "accountPhotoUrl": "https://yt3.ggpht.com/sample-user-photo"
}
```

get_history() → list[dict]

Gets your play history in reverse chronological order

Return type
`list[dict]`

Returns
List of playlistItems, see `get_playlist()` The additional property `played` indicates when the playlistItem was played The additional property `feedbackToken` can be used to remove items with *remove_history_items()*

get_library_albums(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the albums in the user's library.

Parameters

- **limit** (int) – Number of albums to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of albums to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of albums.

Each item is in the following format:

```
{
  "browseId": "MPREb_G8AiyN7RvFg",
  "playlistId": "OLAK5uy_lKgoGvlrWhX0EIPavQUXxyPed8Cj38AWc",
  "title": "Beautiful",
  "type": "Album",
  "thumbnails": [...],
  "artists": [{
    "name": "Project 46",
    "id": "UCXFv36m62USAN5rnVct9B4g"
  }],
  "year": "2015"
}
```

get_library_artists(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the artists of the songs in the user's library.

Parameters

- **limit** (int) – Number of artists to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of artists to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of artists.

Each item is in the following format:

```
{
  "browseId": "UCxEqaQWosMHaTih-tgzDqug",
  "artist": "WildVibes",
  "subscribers": "2.91K",
  "thumbnails": [...]
}
```

get_library_channels(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Get channels the user has added to the library

Parameters

- **limit** (int) – Number of channels to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of channels to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of channels.

Example:

```
[
    {
        "browseId": "UCRFF8xw5dg9mL4r5ryFOtKw",
        "artist": "Jumpers Jump",
        "subscribers": "1.54M",
        "thumbnails": [...]
    },
    {
        "browseId": "UCQ3f2_sO3NJyDkuCxCNSOVA",
        "artist": "BROWN BAG",
        "subscribers": "74.2K",
        "thumbnails": [...]
    }
]
```

get_library_playlists(*limit: int | None = 25*) → list[dict]

Retrieves the playlists in the user's library.

Parameters
limit (Optional[int]) – Number of playlists to retrieve. None retrieves them all.

Return type
list[dict]

Returns
List of owned playlists.

Each item is in the following format:

```
{
    'playlistId': 'PLQwVIlKxHM6rz0fDJVv_0UlXGEWf-bFys',
    'title': 'Playlist title',
    'thumbnails: [...],
    'count': 5
}
```

get_library_podcasts(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Get podcasts the user has added to the library

Parameters

- **limit** (int) – Number of podcasts to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of podcasts to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of podcasts. New Episodes playlist is the first podcast returned, but only if subscribed to relevant podcasts.

Example:

```
[
    {
        "title": "New Episodes",
        "channel":
        {
            "id": null,
            "name": "Auto playlist"
        },
        "browseId": "VLRDPN",
        "podcastId": "RDPN",
        "thumbnails": [...]
    },
    {
        "title": "5 Minuten Harry Podcast",
        "channel":
        {
            "id": "UCDIDXF4WM1qQzerrxeEfSdA",
            "name": "coldmirror"
        },
        "browseId": "MPSPPLDvBqWb1UAGeEt9n6vFH_zdGw65Obf3sH",
        "podcastId": "PLDvBqWb1UAGeEt9n6vFH_zdGw65Obf3sH",
        "thumbnails": [...]
    }
]
```

get_library_songs(*limit: int = 25*, *validate_responses: bool = False*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the songs in the user's library (liked videos are not included). To get liked songs and videos, use

get_liked_songs()

Parameters

- **limit** (int) – Number of songs to retrieve
- **validate_responses** (bool) – Flag indicating if responses from YTM should be validated and retried in case when some songs are missing. Default: False
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of songs to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of songs. Same format as get_playlist()

get_library_subscriptions(*limit: int = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the artists the user has subscribed to.

Parameters

- **limit** (int) – Number of artists to return
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of artists to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of artists. Same format as *get_library_artists()*

rate_playlist(*playlistId: str*, *rating: str = 'INDIFFERENT'*) → dict

Rates a playlist/album ("Add to library"/"Remove from library" interactions on YouTube Music) You can also dislike a playlist/album, which has an effect on your recommendations

Parameters

- **playlistId** (str) – Playlist id
- **rating** (str) – One of LIKE, DISLIKE, INDIFFERENT

  INDIFFERENT removes the playlist/album from the library

Return type
dict

Returns
Full response

rate_song(*videoId: str*, *rating: str = 'INDIFFERENT'*) → dict | None

Rates a song ("thumbs up"/"thumbs down" interactions on YouTube Music)

Parameters

- **videoId** (str) – Video id

- **rating** (str) – One of LIKE, DISLIKE, INDIFFERENT

  INDIFFERENT removes the previous rating and assigns no rating

Return type
Optional[dict]

Returns
Full response

remove_history_items(*feedbackTokens: list[str]*) → dict

Remove an item from the account's history. This method does currently not work with brand accounts

Parameters
feedbackTokens (list[str]) – Token to identify the item to remove, obtained from *get_history()*

Return type
dict

Returns
Full response

subscribe_artists(*channelIds: list[str]*) → dict

Subscribe to artists. Adds the artists to your library

Parameters
channelIds (list[str]) – Artist channel ids

Return type
dict

Returns
Full response

unsubscribe_artists(*channelIds: list[str]*) → dict

Unsubscribe from artists. Removes the artists from your library

Parameters
channelIds (list[str]) – Artist channel ids

Return type
dict

Returns
Full response

ytmusicapi.mixins.playlists module

class ytmusicapi.mixins.playlists.**PlaylistsMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

add_playlist_items(*playlistId: str*, *videoIds: list[str] | None = None*, *source_playlist: str | None = None*, *duplicates: bool = False*) → str | dict

Add songs to an existing playlist

Parameters

- **playlistId** (str) – Playlist id
- **videoIds** (Optional[list[str]]) – List of Video ids
- **source_playlist** (Optional[str]) – Playlist id of a playlist to add to the current playlist (no duplicate check)
- **duplicates** (bool) – If True, duplicates will be added. If False, an error will be returned if there are duplicates (no items are added to the playlist)

Return type
Union[str, dict]

Returns
Status String and a dict containing the new setVideoId for each videoId or full response

create_playlist(*title: str*, *description: str*, *privacy_status: str = 'PRIVATE'*, *video_ids: list | None = None*, *source_playlist: str | None = None*) → str | dict

Creates a new empty playlist and returns its id.

Parameters

- **title** (str) – Playlist title
- **description** (str) – Playlist description
- **privacy_status** (str) – Playlists can be PUBLIC, PRIVATE, or UNLISTED. Default: PRIVATE
- **video_ids** (Optional[list]) – IDs of songs to create the playlist with
- **source_playlist** (Optional[str]) – Another playlist whose songs should be added to the new playlist

Return type
Union[str, dict]

Returns
ID of the YouTube playlist or full response if there was an error

delete_playlist(*playlistId: str*) → str | dict

Delete a playlist.

Parameters
playlistId (str) – Playlist id

Return type
Union[str, dict]

Returns
Status String or full response

edit_playlist(*playlistId: str*, *title: str | None = None*, *description: str | None = None*, *privacyStatus: str | None = None*, *moveItem: str | tuple[str, str] | None = None*, *addPlaylistId: str | None = None*, *addToTop: bool | None = None*) → str | dict

Edit title, description or privacyStatus of a playlist. You may also move an item within a playlist or append another playlist to this playlist.

Parameters

- **playlistId** (str) – Playlist id
- **title** (Optional[str]) – Optional. New title for the playlist
- **description** (Optional[str]) – Optional. New description for the playlist

- **privacyStatus** (Optional[str]) – Optional. New privacy status for the playlist
- **moveItem** (Union[str, tuple[str, str], None]) – Optional. Move one item before another. Items are specified by setVideoId, which is the unique id of this playlist item. See *get_playlist()*
- **addPlaylistId** (Optional[str]) – Optional. Id of another playlist to add to this playlist
- **addToTop** (Optional[bool]) – Optional. Change the state of this playlist to add items to the top of the playlist (if True) or the bottom of the playlist (if False - this is also the default of a new playlist).

Return type
Union[str, dict]

Returns
Status String or full response

get_liked_songs(*limit: int = 100*) → dict

Gets playlist items for the 'Liked Songs' playlist

Parameters
limit (int) – How many items to return. Default: 100

Return type
dict

Returns
List of playlistItem dictionaries. See *get_playlist()*

get_playlist(*playlistId: str*, *limit: int | None = 100*, *related: bool = False*, *suggestions_limit: int = 0*) → dict

Returns a list of playlist items

Parameters

- **playlistId** (str) – Playlist id
- **limit** (Optional[int]) – How many songs to return. None retrieves them all. Default: 100
- **related** (bool) – Whether to fetch 10 related playlists or not. Default: False
- **suggestions_limit** (int) – How many suggestions to return. The result is a list of suggested playlist items (videos) contained in a "suggestions" key. 7 items are retrieved in each internal request. Default: 0

Return type
dict

Returns
Dictionary with information about the playlist. The key tracks contains a List of playlistItem dictionaries

The result is in the following format:

```
{
  "id": "PLQwVIlKxHM6qv-o99iX9R85og7IzF9YS_",
  "privacy": "PUBLIC",
  "title": "New EDM This Week 03/13/2020",
  "thumbnails": [...]
  "description": "Weekly r/EDM new release roundup. Created with github.com/
```

(continues on next page)

(continued from previous page)


```
↪sigma67/spotifyplaylist_to_gmusic",
  "author": "sigmatics",
  "year": "2020",
  "duration": "6+ hours",
  "duration_seconds": 52651,
  "trackCount": 237,
  "suggestions": [
      {
        "videoId": "HLCsfOykA94",
        "title": "Mambo (GATTÜSO Remix)",
        "artists": [{
            "name": "Nikki Vianna",
            "id": "UCMW5eSIO1moVlIBLQzq4PnQ"
          }],
        "album": {
          "name": "Mambo (GATTÜSO Remix)",
          "id": "MPREb_jLeQJsd7U9w"
        },
        "likeStatus": "LIKE",
        "thumbnails": [...],
        "isAvailable": true,
        "isExplicit": false,
        "duration": "3:32",
        "duration_seconds": 212,
        "setVideoId": "to_be_updated_by_client"
      }
  ],
  "related": [
      {
        "title": "Presenting MYRNE",
        "playlistId": "RDCLAK5uy_mbdO3_xdD4NtU1rWI0OmvRSRZ8NH4uJCM",
        "thumbnails": [...],
        "description": "Playlist • YouTube Music"
      }
  ],
  "tracks": [
    {
      "videoId": "bjGppZKiuFE",
      "title": "Lost",
      "artists": [
        {
          "name": "Guest Who",
          "id": "UCkgCRdnnqWnUeIH7EIc3dBg"
        },
        {
          "name": "Kate Wild",
          "id": "UCwR2l3JfJbvB6aq0RnnJfWg"
        }
      ],
      "album": {
        "name": "Lost",
        "id": "MPREb_PxmzvDuqOnC"
```


(continues on next page)

(continued from previous page)

```
      },
      "duration": "2:58",
      "duration_seconds": 178,
      "setVideoId": "748EE8..."
      "likeStatus": "INDIFFERENT",
      "thumbnails": [...],
      "isAvailable": True,
      "isExplicit": False,
      "videoType": "MUSIC_VIDEO_TYPE_OMV",
      "feedbackTokens": {
        "add": "AB9zfpJxtvrU...",
        "remove": "AB9zfpKTyZ..."
    }
  ]
}
```

The setVideoId is the unique id of this playlist item and needed for moving/removing playlist items

get_saved_episodes(*limit: int = 100*) → dict

Gets playlist items for the 'Liked Songs' playlist

Parameters
limit (int) – How many items to return. Default: 100

Return type
dict

Returns
List of playlistItem dictionaries. See *get_playlist()*

remove_playlist_items(*playlistId: str*, *videos: list[dict]*) → str | dict

Remove songs from an existing playlist

Parameters

- **playlistId** (str) – Playlist id
- **videos** (list[dict]) – List of PlaylistItems, see *get_playlist()*. Must contain videoId and setVideoId

Return type
Union[str, dict]

Returns
Status String or full response

ytmusicapi.mixins.podcasts module

class ytmusicapi.mixins.podcasts.**PodcastsMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

Podcasts Mixin

get_channel(*channelId: str*) → dict

Get information about a podcast channel (episodes, podcasts). For episodes, a maximum of 10 episodes are returned, the full list of episodes can be retrieved via *get_channel_episodes()*

Parameters
channelId (str) – channel id

Return type
dict

Returns
Dict containing channel info

Example:

```
{
    "title": 'Stanford Graduate School of Business',
    "thumbnails": [...]
    "episodes":
    {
        "browseId": "UCGwuxdEeCf0TIA2RbPOj-8g",
        "results":
        [
            {
                "index": 0,
                "title": "The Brain Gain: The Impact of Immigration on American␣
↪Innovation with Rebecca Diamond",
                "description": "Immigrants' contributions to America ...",
                "duration": "24 min",
                "videoId": "TS3Ovvk3VAA",
                "browseId": "MPEDTS3Ovvk3VAA",
                "videoType": "MUSIC_VIDEO_TYPE_PODCAST_EPISODE",
                "date": "Mar 6, 2024",
                "thumbnails": [...]
            },
        ],
        "params": "6gPiAUdxWUJXcFlCQ3BN..."
    },
    "podcasts":
    {
        "browseId": null,
        "results":
        [
            {
                "title": "Stanford GSB Podcasts",
                "channel":
                {
                    "id": "UCGwuxdEeCf0TIA2RbPOj-8g",
                    "name": "Stanford Graduate School of Business"
                },
                "browseId": "MPSPPLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9",
                "podcastId": "PLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9",
                "thumbnails": [...]
            }
       ]
    }
}
```

get_channel_episodes(*channelId: str*, *params: str*) → list[dict]

Get all channel episodes. This endpoint is currently unlimited

Parameters

- **channelId** (str) – channelId of the user
- **params** (str) – params obtained by *get_channel()*

Return type
list[dict]

Returns
List of channel episodes in the format of *get_channel()* "episodes" key

get_episode(*videoId: str*) → dict

Retrieve episode data for a single episode

Note: To save an episode, you need to call `add_playlist_items` to add it to the `SE` (saved episodes) playlist.

Parameters
videoId (str) – browseId (MPED..) or videoId for a single episode

Return type
dict

Returns
Dict containing information about the episode

The description elements are based on a custom dataclass, not shown in the example below The description text items also contain "n" to indicate newlines, removed below due to RST issues

Example:


```
{
    "author":
    {
        "name": "Stanford GSB Podcasts",
        "id": "MPSPPLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9"
    },
    "title": "124. Making Meetings Me...",
    "date": "Jan 16, 2024",
    "duration": "25 min",
    "saved": false,
    "playlistId": "MPSPPLxq_lXOUlvQDUNyoBYLkN8aVt5yAwEtG9",
    "description":
    [
        {
            "text": "Delve into why people hate meetings, ... Karin Reed ("
        },
        {
            "text": "https://speakerdynamics.com/team/",
            "url": "https://speakerdynamics.com/team/"
        },
        {
            "text": ")Chapters:("
```


(continues on next page)

(continued from previous page)

```
        },
        {
            "text": "00:00",
            "seconds": 0
        },
        {
            "text": ") Introduction Host Matt Abrahams...("
        },
        {
            "text": "01:30",
            "seconds": 90
        },
    ]
}
```

get_episodes_playlist(*playlist_id: str = 'RDPN'*) → dict

Get all episodes in an episodes playlist. Currently the only known playlist is the "New Episodes" auto-generated playlist

Parameters
playlist_id (str) – Playlist ID, defaults to "RDPN", the id of the New Episodes playlist

Return type
dict

Returns
Dictionary in the format of *get_podcast()*

get_podcast(*playlistId: str*, *limit: int | None = 100*) → dict

Returns podcast metadata and episodes

Note: To add a podcast to your library, you need to call `rate_playlist` on it

Parameters

- **playlistId** (str) – Playlist id
- **limit** (Optional[int]) – How many songs to return. None retrieves them all. Default: 100

Return type
dict

Returns
Dict with podcast information

Example:

```
{
    "author":
    {
        "name": "Stanford Graduate School of Business",
        "id": "UCGwuxdEeCf0TIA2RbPOj-8g"
    },
```

(continues on next page)

(continued from previous page)

```
    "title": "Think Fast, Talk Smart: The Podcast",
    "description": "Join Matt Abrahams, a lecturer of...",
    "saved": false,
    "episodes":
    [
        {
            "index": 0,
            "title": "132. Lean Into Failure: How to Make Mistakes That Work |␣
↪Think Fast, Talk Smart: Communication...",
            "description": "Effective and productive teams and...",
            "duration": "25 min",
            "videoId": "xAEGaW2my7E",
            "browseId": "MPEDxAEGaW2my7E",
            "videoType": "MUSIC_VIDEO_TYPE_PODCAST_EPISODE",
            "date": "Mar 5, 2024",
            "thumbnails": [...]
        }
    ]
}
```

ytmusicapi.mixins.search module

class ytmusicapi.mixins.search.**SearchMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

get_search_suggestions(*query: str*, *detailed_runs=False*) → list[str] | list[dict]

Get Search Suggestions

Parameters

- **query** (str) – Query string, i.e. 'faded'
- **detailed_runs** – Whether to return detailed runs of each suggestion. If True, it returns the query that the user typed and the remaining suggestion along with the complete text (like many search services usually bold the text typed by the user). Default: False, returns the list of search suggestions in plain text.

Return type

Union[list[str], list[dict]]

Returns

A list of search suggestions. If `detailed_runs` is False, it returns plain text suggestions.

If `detailed_runs` is True, it returns a list of dictionaries with detailed information.

Example response when `query` is 'fade' and `detailed_runs` is set to `False`:

```
[
  "faded",
  "faded alan walker lyrics",
  "faded alan walker",
  "faded remix",
  "faded song",
```

(continues on next page)

(continued from previous page)

```
  "faded lyrics",
  "faded instrumental"
]
```

Example response when `detailed_runs` is set to `True`:

```
[
  {
    "text": "faded",
    "runs": [
      {
        "text": "fade",
        "bold": true
      },
      {
        "text": "d"
      }
    ],
    "fromHistory": true,
    "feedbackToken": "AEEJK..."
  },
  {
    "text": "faded alan walker lyrics",
    "runs": [
      {
        "text": "fade",
        "bold": true
      },
      {
        "text": "d alan walker lyrics"
      }
    ],
    "fromHistory": false,
    "feedbackToken": None
  },
  {
    "text": "faded alan walker",
    "runs": [
      {
        "text": "fade",
        "bold": true
      },
      {
        "text": "d alan walker"
      }
    ],
    "fromHistory": false,
    "feedbackToken": None
  },
  ...
]
```

remove_search_suggestions(*suggestions: list[dict[str, Any]]*, *indices: list[int] | None = None*) → bool

Remove search suggestion from the user search history.

Parameters

- **suggestions** (list[dict[str, Any]]) – The dictionary obtained from the *get_search_suggestions()* (with detailed_runs=True)`
- **indices** (Optional[list[int]]) – Optional. The indices of the suggestions to be removed. Default: remove all suggestions.

Return type
bool

Returns

True if the operation was successful, False otherwise.

Example usage:

```
# Removing suggestion number 0
suggestions = ytmusic.get_search_suggestions(query="fade", detailed_
↪runs=True)
success = ytmusic.remove_search_suggestions(suggestions=suggestions,␣
↪indices=[0])
if success:
    print("Suggestion removed successfully")
else:
    print("Failed to remove suggestion")
```

search(*query: str*, *filter: str | None = None*, *scope: str | None = None*, *limit: int = 20*, *ignore_spelling: bool = False*) → list[dict]

Search YouTube music Returns results within the provided category.

Parameters

- **query** (str) – Query string, i.e. 'Oasis Wonderwall'
- **filter** (Optional[str]) – Filter for item types. Allowed values: songs, videos, albums, artists, playlists, community_playlists, featured_playlists, uploads. Default: Default search, including all types of items.
- **scope** (Optional[str]) – Search scope. Allowed values: library, uploads. Default: Search the public YouTube Music catalogue. Changing scope from the default will reduce the number of settable filters. Setting a filter that is not permitted will throw an exception. For uploads, no filter can be set. For library, community_playlists and featured_playlists filter cannot be set.
- **limit** (int) – Number of search results to return Default: 20
- **ignore_spelling** (bool) – Whether to ignore YTM spelling suggestions. If True, the exact search term will be searched for, and will not be corrected. This does not have any effect when the filter is set to uploads. Default: False, will use YTM's default behavior of autocorrecting the search.

Return type
list[dict]

Returns

List of results depending on filter. resultType specifies the type of item (important for default search). albums, artists and playlists additionally contain a browseId, corresponding to albumId, channelId and playlistId (browseId=`VL`+playlistId)

Example list for default search with one result per resultType for brevity. Normally there are 3 results per resultType and an additional `thumbnails` key:


```
[
  {
    "category": "Top result",
    "resultType": "video",
    "videoId": "vU05Eksc_iM",
    "title": "Wonderwall",
    "artists": [
      {
        "name": "Oasis",
        "id": "UCmMUZbaYdNH0bEd1PAlAqsA"
      }
    ],
    "views": "1.4M",
    "videoType": "MUSIC_VIDEO_TYPE_OMV",
    "duration": "4:38",
    "duration_seconds": 278
  },
  {
    "category": "Songs",
    "resultType": "song",
    "videoId": "ZrOKjDZOtkA",
    "title": "Wonderwall",
    "artists": [
      {
        "name": "Oasis",
        "id": "UCmMUZbaYdNH0bEd1PAlAqsA"
      }
    ],
    "album": {
      "name": "(What's The Story) Morning Glory? (Remastered)",
      "id": "MPREb_9nqEki4ZDpp"
    },
    "duration": "4:19",
    "duration_seconds": 259
    "isExplicit": false,
    "feedbackTokens": {
      "add": null,
      "remove": null
    }
  },
  {
    "category": "Albums",
    "resultType": "album",
    "browseId": "MPREb_IInSY5QXXrW",
    "playlistId": "OLAK5uy_kunInnOpcKECWIBQGB0Qj6ZjquxDvfckg",
    "title": "(What's The Story) Morning Glory?",
    "type": "Album",
    "artist": "Oasis",
    "year": "1995",
    "isExplicit": false
```


(continues on next page)

(continued from previous page)


```
  },
  {
    "category": "Community playlists",
    "resultType": "playlist",
    "browseId": "VLPLK1PkWQlWtnNfovRdGWpKffO1Wdi2kvDx",
    "title": "Wonderwall - Oasis",
    "author": "Tate Henderson",
    "itemCount": "174"
  },
  {
    "category": "Videos",
    "resultType": "video",
    "videoId": "bx1Bh8ZvH84",
    "title": "Wonderwall",
    "artists": [
      {
        "name": "Oasis",
        "id": "UCmMUZbaYdNH0bEd1PAlAqsA"
      }
    ],
    "views": "386M",
    "duration": "4:38",
    "duration_seconds": 278
  },
  {
    "category": "Artists",
    "resultType": "artist",
    "browseId": "UCmMUZbaYdNH0bEd1PAlAqsA",
    "artist": "Oasis",
    "shuffleId": "RDAOkjHYJjL1a3xspEyVkhHAsg",
    "radioId": "RDEMkjHYJjL1a3xspEyVkhHAsg"
  },
  {
    "category": "Profiles",
    "resultType": "profile",
    "title": "Taylor Swift Time",
    "name": "@TaylorSwiftTime",
    "browseId": "UCSCRK7XlVQ6fBdEl00kX6pQ",
    "thumbnails": ...
  }
]
```

ytmusicapi.mixins.uploads module

class ytmusicapi.mixins.uploads.**UploadsMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

delete_upload_entity(*entityId: str*) → str | dict

Deletes a previously uploaded song or album

Parameters
entityId (str) – The entity id of the uploaded song or album, e.g. retrieved from *get_library_upload_songs()*

Return type
Union[str, dict]

Returns
Status String or error

get_library_upload_album(*browseId: str*) → dict

Get information and tracks of an album associated with uploaded tracks

Parameters
browseId (str) – Browse id of the upload album, i.e. from i.e. from *get_library_upload_songs()*

Return type
dict

Returns
Dictionary with title, description, artist and tracks.

Example album:

```
{
  "title": "18 Months",
  "type": "Album",
  "thumbnails": [...],
  "trackCount": 7,
  "duration": "24 minutes",
  "audioPlaylistId": "MLPRb_po_55chars",
  "tracks": [
    {
      "entityId": "t_po_22chars",
      "videoId": "FVo-UZoPygI",
      "title": "Feel So Close",
      "duration": "4:15",
      "duration_seconds": 255,
      "artists": None,
      "album": {
        "name": "18 Months",
        "id": "FEmusic_library_privately_owned_release_detailb_po_55chars"
      },
      "likeStatus": "INDIFFERENT",
      "thumbnails": None
    },
```

get_library_upload_albums(*limit: int | None = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the albums of uploaded songs in the user's library.

Parameters

- **limit** (Optional[int]) – Number of albums to return. None retrives them all. Default: 25
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of albums to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of albums as returned by get_library_albums()

get_library_upload_artist(*browseId: str*, *limit: int = 25*) → list[dict]

Returns a list of uploaded tracks for the artist.

Parameters

- **browseId** (str) – Browse id of the upload artist, i.e. from *get_library_upload_songs()*
- **limit** (int) – Number of songs to return (increments of 25).

Return type
list[dict]

Returns
List of uploaded songs.

Example List:

```
[
  {
    "entityId": "t_po_CICr2crg7OWpchDKwoakAQ",
    "videoId": "Dtffhy8WJgw",
    "title": "Hold Me (Original Mix)",
    "artists": [
      {
        "name": "Jakko",
        "id": "FEmusic_library_privately_owned_artist_detaila_po_
↪CICr2crg7OWpchIFamFra28"
      }
    ],
    "album": null,
    "likeStatus": "LIKE",
    "thumbnails": [...]
  }
]
```

get_library_upload_artists(*limit: int | None = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Gets the artists of uploaded songs in the user's library.

Parameters

- **limit** (Optional[int]) – Number of artists to return. None retrieves them all. Default: 25
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of artists to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of artists as returned by get_library_artists()

get_library_upload_songs(*limit: int | None = 25*, *order: Literal['a_to_z', 'z_to_a', 'recently_added'] | None = None*) → list[dict]

Returns a list of uploaded songs

Parameters

- **limit** (Optional[int]) – How many songs to return. None retrieves them all. Default: 25
- **order** (Optional[Literal['a_to_z', 'z_to_a', 'recently_added']]) – Order of songs to return. Allowed values: a_to_z, z_to_a, recently_added. Default: Default order.

Return type
list[dict]

Returns
List of uploaded songs.

Each item is in the following format:

```
{
  "entityId": "t_po_CICr2crg7OWpchDpjPjrBA",
  "videoId": "Uise6RPKoek",
  "artists": [{
    'name': 'Coldplay',
    'id': 'FEmusic_library_privately_owned_artist_detaila_po_
↪CICr2crg7OWpchIIY29sZHBsYXk',
  }],
  "title": "A Sky Full Of Stars",
  "album": "Ghost Stories",
  "likeStatus": "LIKE",
  "thumbnails": [...]
}
```

upload_song(*filepath: str*) → *ytmusicapi.enums.ResponseStatus* | requests.models.Response

Uploads a song to YouTube Music

Parameters
filepath (str) – Path to the music file (mp3, m4a, wma, flac or ogg)

Return type
Union[*ResponseStatus*, Response]

Returns
Status String or full response

ytmusicapi.mixins.watch module

class ytmusicapi.mixins.watch.**WatchMixin**(**args*, ***kwargs*)

Bases: MixinProtocol

get_watch_playlist(*videoId: str | None = None*, *playlistId: str | None = None*, *limit=25*, *radio: bool = False*, *shuffle: bool = False*) → dict[str, Union[list[dict], str, NoneType]]

Get a watch list of tracks. This watch playlist appears when you press play on a track in YouTube Music.

Please note that the INDIFFERENT likeStatus of tracks returned by this endpoint may be either INDIFFERENT or DISLIKE, due to ambiguous data returned by YouTube Music.

Parameters

- **videoId** (Optional[str]) – videoId of the played video
- **playlistId** (Optional[str]) – playlistId of the played playlist or album
- **limit** – minimum number of watch playlist items to return
- **radio** (bool) – get a radio playlist (changes each time)
- **shuffle** (bool) – shuffle the input playlist. only works when the playlistId parameter is set at the same time. does not work if radio=True

Return type

dict[str, Union[list[dict], str, None]]

Returns

List of watch playlist items. The counterpart key is optional and only appears if a song has a corresponding video counterpart (UI song/video switcher).

Example:


```
{
    "tracks": [
        {
          "videoId": "9mWr4c_ig54",
          "title": "Foolish Of Me (feat. Jonathan Mendelsohn)",
          "length": "3:07",
          "thumbnail": [
            {
              "url": "https://lh3.googleusercontent.com/
↪ulK2YaLtOW0PzcN7ufltG6e4ae3WZ9Bvg8CCwhe6LOccu1lCKxJy2r5AsYrsHeMBSLrGJCNpJqXgwczk=w60-
↪h60-l90-rj",
              "width": 60,
              "height": 60
            }...
          ],
          "feedbackTokens": {
            "add": "AB9zfpIGg9XN4u2iJ...",
            "remove": "AB9zfpJdzWLcdZtC..."
          },
          "likeStatus": "INDIFFERENT",
          "videoType": "MUSIC_VIDEO_TYPE_ATV",
          "artists": [
            {
              "name": "Seven Lions",
```


(continues on next page)

(continued from previous page)


```
            "id": "UCYd2yzYRx7b9FYnBSlbnknA"
          },
          {
            "name": "Jason Ross",
            "id": "UCVCD9Iwnqn2ipN9JIF6B-nA"
          },
          {
            "name": "Crystal Skies",
            "id": "UCTJZESxeZ0J_M7JXyFUVmvA"
          }
        ],
        "album": {
          "name": "Foolish Of Me",
          "id": "MPREb_C8aRK1qmsDJ"
        },
        "year": "2020",
        "counterpart": {
          "videoId": "E0S4W34zFMA",
          "title": "Foolish Of Me [ABGT404] (feat. Jonathan Mendelsohn)",
          "length": "3:07",
          "thumbnail": [...],
          "feedbackTokens": null,
          "likeStatus": "LIKE",
          "artists": [
            {
              "name": "Jason Ross",
              "id": null
            },
            {
              "name": "Seven Lions",
              "id": null
            },
            {
              "name": "Crystal Skies",
              "id": null
            }
          ],
          "views": "6.6K"
        }
      },...
  ],
  "playlistId": "RDAMVM4y33h81phKU",
  "lyrics": "MPLYt_HNNclO0Ddoc-17"
}
```

Module contents

ytmusicapi.models package

Submodules

ytmusicapi.models.lyrics module

class ytmusicapi.models.lyrics.**LyricLine**(*text: str*, *start_time: int*, *end_time: int*, *id: int*) → None

Bases: object

Represents a line of lyrics with timestamps (in milliseconds).

Args:
text (str): The Songtext. start_time (int): Begin of the lyric in milliseconds. end_time (int): End of the lyric in milliseconds. id (int): A Metadata-Id that probably uniquely identifies each lyric line.

end_time: int

classmethod from_raw(*raw_lyric: dict*)

Converts lyrics in the format from the api to a more reasonable format

Parameters
raw_lyric (dict) – The raw lyric-data returned by the mobile api.

Return LyricLine
A *LyricLine*

id: int

start_time: int

text: str

class ytmusicapi.models.lyrics.**Lyrics**

Bases: TypedDict

hasTimestamps: Literal[False]

lyrics: str

source: Optional[str]

class ytmusicapi.models.lyrics.**TimedLyrics**

Bases: TypedDict

hasTimestamps: Literal[True]

lyrics: list[*LyricLine*]

source: Optional[str]

Module contents

class ytmusicapi.models.**LyricLine**(*text: str*, *start_time: int*, *end_time: int*, *id: int*) → None

Bases: object

Represents a line of lyrics with timestamps (in milliseconds).

Args:
text (str): The Songtext. start_time (int): Begin of the lyric in milliseconds. end_time (int): End of the lyric in milliseconds. id (int): A Metadata-Id that probably uniquely identifies each lyric line.

end_time: int

classmethod **from_raw**(*raw_lyric: dict*)

Converts lyrics in the format from the api to a more reasonable format

Parameters
raw_lyric (dict) – The raw lyric-data returned by the mobile api.

Return LyricLine
A *LyricLine*

id: int

start_time: int

text: str

class ytmusicapi.models.**Lyrics**

Bases: TypedDict

hasTimestamps: Literal[False]

lyrics: str

source: Optional[str]

class ytmusicapi.models.**TimedLyrics**

Bases: TypedDict

hasTimestamps: Literal[True]

lyrics: list[*LyricLine*]

source: Optional[str]

ytmusicapi.parsers package

Submodules

ytmusicapi.parsers.albums module

ytmusicapi.parsers.albums.**parse_album_header**(*response*)

ytmusicapi.parsers.albums.**parse_album_header_2024**(*response*)

ytmusicapi.parsers.browsing module

ytmusicapi.parsers.browsing.**parse_album**(*result*)

ytmusicapi.parsers.browsing.**parse_content_list**(*results*, *parse_func*, *key='musicTwoRowItemRenderer'*)

ytmusicapi.parsers.browsing.**parse_mixed_content**(*rows*)

ytmusicapi.parsers.browsing.**parse_playlist**(*data*)

ytmusicapi.parsers.browsing.**parse_related_artist**(*data*)

ytmusicapi.parsers.browsing.**parse_single**(*result*)

ytmusicapi.parsers.browsing.**parse_song**(*result*)

ytmusicapi.parsers.browsing.**parse_song_flat**(*data*)

ytmusicapi.parsers.browsing.**parse_video**(*result*)

ytmusicapi.parsers.browsing.**parse_watch_playlist**(*data*)

ytmusicapi.parsers.explore module

ytmusicapi.parsers.explore.**parse_chart_artist**(*data*)

ytmusicapi.parsers.explore.**parse_chart_song**(*data*)

ytmusicapi.parsers.explore.**parse_chart_trending**(*data*)

ytmusicapi.parsers.explore.**parse_ranking**(*data*)

ytmusicapi.parsers.i18n module

class ytmusicapi.parsers.i18n.**Parser**(*language*)

Bases: object

get_api_result_types()

get_search_result_types()

parse_channel_contents(*results: list*) → dict

Return type
dict

ytmusicapi.parsers.library module

ytmusicapi.parsers.library.**get_library_contents**(*response*, *renderer*)

Find library contents. This function is a bit messy now as it is supporting two different response types. Can be cleaned up once all users are migrated to the new responses. :type response: :param response: ytmusicapi response :type renderer: :param renderer: GRID or MUSIC_SHELF :return: library contents or None

ytmusicapi.parsers.library.**parse_albums**(*results*)

ytmusicapi.parsers.library.**parse_artists**(*results*, *uploaded=False*)

ytmusicapi.parsers.library.**parse_library_albums**(*response*, *request_func*, *limit*)

ytmusicapi.parsers.library.**parse_library_artists**(*response*, *request_func*, *limit*)

ytmusicapi.parsers.library.**parse_library_podcasts**(*response*, *request_func*, *limit*)

ytmusicapi.parsers.library.**parse_library_songs**(*response*)

ytmusicapi.parsers.library.**pop_songs_random_mix**(*results*) → None

remove the random mix that conditionally appears at the start of library songs

Return type
None

ytmusicapi.parsers.playlists module

ytmusicapi.parsers.playlists.**parse_audio_playlist**(*response: dict*, *limit: int | None*, *request_func*) → dict[str, Any]

Return type
dict[str, Any]

ytmusicapi.parsers.playlists.**parse_playlist_header**(*response: dict*) → dict[str, Any]

Return type
dict[str, Any]

ytmusicapi.parsers.playlists.**parse_playlist_header_meta**(*header: dict[str, Any]*) → dict[str, Any]

Return type
dict[str, Any]

ytmusicapi.parsers.playlists.**parse_playlist_item**(*data: dict*, *menu_entries: list[list] | None = None*, *is_album=False*) → dict | None

Return type
Optional[dict]

ytmusicapi.parsers.playlists.**parse_playlist_items**(*results*, *menu_entries: list[list] | None = None*, *is_album=False*)

ytmusicapi.parsers.playlists.**validate_playlist_id**(*playlistId: str*) → str

Return type
str

ytmusicapi.parsers.podcasts module

class ytmusicapi.parsers.podcasts.**Description**(**args*, ***kwargs*)

Bases: list[*DescriptionElement*]

classmethod from_runs(*description_runs: list[dict]*) → *ytmusicapi.parsers.podcasts.Description*

parse the description runs into a usable format

Parameters
description_runs (list[dict]) – the original description runs

Return type
Description

Returns
List of text (str), timestamp (int) and link values (Link object)

property text: str

class ytmusicapi.parsers.podcasts.**DescriptionElement**(*text: str*) → None

Bases: object

text: str

class ytmusicapi.parsers.podcasts.**Link**(*text: str*, *url: str*) → None

Bases: *DescriptionElement*

url: str

class ytmusicapi.parsers.podcasts.**Timestamp**(*text: str*, *seconds: int*) → None

Bases: *DescriptionElement*

seconds: int

ytmusicapi.parsers.podcasts.**parse_base_header**(*header: dict*) → dict

parse common left hand side (header) items of an episode or podcast page

Return type
dict

ytmusicapi.parsers.podcasts.**parse_episode**(*data*)

Parses a single episode under "Episodes" on a channel page or on a podcast page

ytmusicapi.parsers.podcasts.**parse_episode_header**(*header: dict*) → dict

Return type
dict

ytmusicapi.parsers.podcasts.**parse_podcast**(*data*)

Parses a single podcast under "Podcasts" on a channel page

ytmusicapi.parsers.podcasts.**parse_podcast_header**(*header: dict*) → dict

Return type
dict

ytmusicapi.parsers.search module

ytmusicapi.parsers.search.**get_search_params**(*filter*, *scope*, *ignore_spelling*)

ytmusicapi.parsers.search.**get_search_result_type**(*result_type_local*, *result_types_local*)

ytmusicapi.parsers.search.**parse_album_playlistid_if_exists**(*data: dict[str, Any]*) → str | None

the content of the data changes based on whether the user is authenticated or not

Return type
Optional[str]

ytmusicapi.parsers.search.**parse_search_result**(*data*, *api_search_result_types*, *result_type*, *category*)

ytmusicapi.parsers.search.**parse_search_results**(*results*, *api_search_result_types*, *resultType=None*, *category=None*)

ytmusicapi.parsers.search.**parse_search_suggestions**(*results: dict[str, Any]*, *detailed_runs: bool*) → list[str] | list[dict[str, Any]]

Return type
Union[list[str], list[dict[str, Any]]]

ytmusicapi.parsers.search.**parse_top_result**(*data*, *search_result_types*)

ytmusicapi.parsers.songs module

ytmusicapi.parsers.songs.**parse_like_status**(*service*)

ytmusicapi.parsers.songs.**parse_song_album**(*data*, *index*)

ytmusicapi.parsers.songs.**parse_song_artists**(*data*, *index*)

ytmusicapi.parsers.songs.**parse_song_artists_runs**(*runs*)

ytmusicapi.parsers.songs.**parse_song_library_status**(*item*) → bool

Returns True if song is in the library

Return type
bool

ytmusicapi.parsers.songs.**parse_song_menu_tokens**(*item*)

ytmusicapi.parsers.songs.**parse_song_runs**(*runs*)

ytmusicapi.parsers.uploads module

ytmusicapi.parsers.uploads.**parse_uploaded_items**(*results*)

ytmusicapi.parsers.watch module

ytmusicapi.parsers.watch.**get_tab_browse_id**(*watchNextRenderer*, *tab_id*)

ytmusicapi.parsers.watch.**parse_watch_playlist**(*results: list[dict[str, Any]]*) → list[dict[str, Any]]

Return type
list[dict[str, Any]]

ytmusicapi.parsers.watch.**parse_watch_track**(*data*)

Module contents

Submodules

ytmusicapi.constants module

ytmusicapi.continuations module

ytmusicapi.continuations.**get_continuation_contents**(*continuation*, *parse_func*)

ytmusicapi.continuations.**get_continuation_params**(*results*, *ctoken_path=''*)

ytmusicapi.continuations.**get_continuation_string**(*ctoken*)

ytmusicapi.continuations.**get_continuation_token**(*results: list[dict[str, Any]]*) → str | None

Return type
Optional[str]

ytmusicapi.continuations.**get_continuations**(*results*, *continuation_type*, *limit*, *request_func*, *parse_func*, *ctoken_path=''*, *reloadable=False*)

ytmusicapi.continuations.**get_continuations_2025**(*results*, *limit*, *request_func*, *parse_func*)

ytmusicapi.continuations.**get_parsed_continuation_items**(*response*, *parse_func*, *continuation_type*)

ytmusicapi.continuations.**get_reloadable_continuation_params**(*results*)

ytmusicapi.continuations.**get_validated_continuations**(*results*, *continuation_type*, *limit*, *per_page*, *request_func*, *parse_func*, *ctoken_path=''*)

ytmusicapi.continuations.**resend_request_until_parsed_response_is_valid**(*request_func*, *request_additional_params*, *parse_func*, *validate_func*, *max_retries*)

ytmusicapi.continuations.**validate_response**(*response*, *per_page*, *limit*, *current_count*)

ytmusicapi.enums module

class ytmusicapi.enums.**ResponseStatus**(*value*)

Bases: str, Enum

An enumeration.

SUCCEEDED = 'STATUS_SUCCEEDED'

ytmusicapi.exceptions module

custom exception classes for ytmusicapi

exception ytmusicapi.exceptions.**YTMusicError**

Bases: Exception

base error class

shall only be raised if none of the subclasses below are fitting

exception ytmusicapi.exceptions.**YTMusicServerError**

Bases: *YTMusicError*

error caused by the YouTube Music backend

exception ytmusicapi.exceptions.**YTMusicUserError**

Bases: *YTMusicError*

error caused by invalid usage of ytmusicapi

ytmusicapi.helpers module

ytmusicapi.helpers.**get_authorization**(*auth*)

ytmusicapi.helpers.**get_visitor_id**(*request_func*)

ytmusicapi.helpers.**initialize_context**()

ytmusicapi.helpers.**initialize_headers**()

ytmusicapi.helpers.**sapisid_from_cookie**(*raw_cookie*)

ytmusicapi.helpers.**sum_total_duration**(*item*)

ytmusicapi.helpers.**to_int**(*string*)

ytmusicapi.navigation module

commonly used navigation paths

ytmusicapi.navigation.**find_object_by_key**(*object_list*, *key*, *nested=None*, *is_key=False*)

ytmusicapi.navigation.**find_objects_by_key**(*object_list*, *key*, *nested=None*)

ytmusicapi.navigation.**nav**(*root: dict*, *items: list[Any]*, *none_if_absent: bool = False*) → Any | None

Access a nested object in root by item sequence.

Return type
Optional[Any]

ytmusicapi.setup module

ytmusicapi.setup.**main**() → *ytmusicapi.auth.oauth.token.RefreshingToken* | str

Return type
Union[*RefreshingToken*, str]

ytmusicapi.setup.**parse_args**(*args*)

ytmusicapi.setup.**setup**(*filepath: str | None = None*, *headers_raw: str | None = None*) → str

Requests browser headers from the user via command line and returns a string that can be passed to YTMusic()

Parameters

- **filepath** (Optional[str]) – Optional filepath to store headers to.
- **headers_raw** (Optional[str]) – Optional request headers copied from browser. Otherwise requested from terminal

Return type
str

Returns
configuration headers string

ytmusicapi.setup.**setup_oauth**(*client_id: str*, *client_secret: str*, *filepath: str | None = None*, *session: requests.sessions.Session | None = None*, *proxies: dict | None = None*, *open_browser: bool = False*) → *ytmusicapi.auth.oauth.token.RefreshingToken*

Starts oauth flow from the terminal and returns a string that can be passed to YTMusic()

Parameters

- **client_id** (str) – Optional. Used to specify the client_id oauth should use for authentication flow. If provided, client_secret MUST also be passed or both will be ignored.
- **client_secret** (str) – Optional. Same as client_id but for the oauth client secret.
- **session** (Optional[Session]) – Session to use for authentication
- **proxies** (Optional[dict]) – Proxies to use for authentication
- **filepath** (Optional[str]) – Optional filepath to store headers to.
- **open_browser** (bool) – If True, open the default browser with the setup link

Return type
RefreshingToken

Returns
configuration headers string

ytmusicapi.ytmusic module

class ytmusicapi.ytmusic.**YTMusic**(*auth: str | dict | None = None*, *user: str | None = None*, *requests_session: requests.sessions.Session | None = None*, *proxies: dict[str, str] | None = None*, *language: str = 'en'*, *location: str = ''*, *oauth_credentials:* ytmusicapi.auth.oauth.credentials.OAuthCredentials | *None = None*)

Bases: *YTMusicBase*, *BrowsingMixin*, *SearchMixin*, *WatchMixin*, *ExploreMixin*, *LibraryMixin*, *PlaylistsMixin*, *PodcastsMixin*, *UploadsMixin*

Allows automated interactions with YouTube Music by emulating the YouTube web client's requests. Permits both authenticated and non-authenticated requests. Authentication header data must be provided on initialization.

class ytmusicapi.ytmusic.**YTMusicBase**(*auth: str | dict | None = None*, *user: str | None = None*, *requests_session: requests.sessions.Session | None = None*, *proxies: dict[str, str] | None = None*, *language: str = 'en'*, *location: str = ''*, *oauth_credentials:* ytmusicapi.auth.oauth.credentials.OAuthCredentials | *None = None*)

Bases: `object`

as_mobile() → collections.abc.Iterator[None]

Return type
`Iterator[None]`

Not thread-safe!

Temporarily changes the *context* to enable different results from the API, meant for the Android mobile-app. All calls inside the *with*-statement with emulate mobile behavior.

This context-manager has no *enter_result*, as it operates in-place and only temporarily alters the underlying *YTMusic*-object.

Example:

```
with yt.as_mobile():
    yt._send_request(...)  # results as mobile-app

yt._send_request(...)  # back to normal, like web-app
```

property base_headers: CaseInsensitiveDict

property headers: CaseInsensitiveDict

proxies: Optional[dict[str, str]]

params for session modification

Module contents

class ytmusicapi.**OAuthCredentials**(*client_id: str*, *client_secret: str*, *session: requests.sessions.Session | None = None*, *proxies: dict | None = None*)

Bases: *Credentials*

Class for handling OAuth credential retrieval and refreshing.

client_id: str

client_secret: str

get_code() → *ytmusicapi.auth.oauth.models.AuthCodeDict*

Method for obtaining a new user auth code. First step of token creation.

Return type
AuthCodeDict

refresh_token(*refresh_token: str*) → *ytmusicapi.auth.oauth.models.BaseTokenDict*

Method for requesting a new access token for a given `refresh_token`. Token must have been created by the same OAuth client.

Parameters
refresh_token (str) – Corresponding `refresh_token` for a matching `access_token`. Obtained via

Return type
BaseTokenDict

token_from_code(*device_code: str*) → *ytmusicapi.auth.oauth.models.RefreshableTokenDict*

Method for verifying user auth code and conversion into a FullTokenDict.

Return type
RefreshableTokenDict

class ytmusicapi.**YTMusic**(*auth: str | dict | None = None*, *user: str | None = None*, *requests_session: requests.sessions.Session | None = None*, *proxies: dict[str, str] | None = None*, *language: str = 'en'*, *location: str = ''*, *oauth_credentials:* ytmusicapi.auth.oauth.credentials.OAuthCredentials | *None = None*)

Bases: *YTMusicBase*, *BrowsingMixin*, *SearchMixin*, *WatchMixin*, *ExploreMixin*, *LibraryMixin*, *PlaylistsMixin*, *PodcastsMixin*, *UploadsMixin*

Allows automated interactions with YouTube Music by emulating the YouTube web client's requests. Permits both authenticated and non-authenticated requests. Authentication header data must be provided on initialization.

ytmusicapi.**setup**(*filepath: str | None = None*, *headers_raw: str | None = None*) → str

Requests browser headers from the user via command line and returns a string that can be passed to YTMusic()

Parameters

- **filepath** (Optional[str]) – Optional filepath to store headers to.
- **headers_raw** (Optional[str]) – Optional request headers copied from browser. Otherwise requested from terminal

Return type
str

Returns
configuration headers string

ytmusicapi.**setup_oauth**(*client_id: str*, *client_secret: str*, *filepath: str | None = None*, *session: requests.sessions.Session | None = None*, *proxies: dict | None = None*, *open_browser: bool = False*) → *ytmusicapi.auth.oauth.token.RefreshingToken*

Starts oauth flow from the terminal and returns a string that can be passed to YTMusic()

Parameters

- **client_id** (str) – Optional. Used to specify the client_id oauth should use for authentication flow. If provided, client_secret MUST also be passed or both will be ignored.
- **client_secret** (str) – Optional. Same as client_id but for the oauth client secret.
- **session** (Optional[Session]) – Session to use for authentication
- **proxies** (Optional[dict]) – Proxies to use for authentication
- **filepath** (Optional[str]) – Optional filepath to store headers to.
- **open_browser** (bool) – If True, open the default browser with the setup link

Return type
RefreshingToken

Returns
configuration headers string

5.4 FAQ

Frequently asked questions for `ytmusicapi`. Contributions are welcome, please submit a PR.

5.4.1 Setup

My library results are empty even though I set up my cookie correctly.

Please make sure that you don't have multiple Google accounts. `ytmusicapi` might be returning results from a different account which is currently empty. You can set your account using `X-Goog-AuthUser` in your headers file (numeric index) or by providing the id of a brand account with `ytmusic = YTMusic(headers, "1234..")`. For more details see the *Reference*.

5.4.2 Usage

How do I add content to my library?

- **songs**: edit_song_library_status . Liking a song using rate_song does *not* add it to your library, only to your liked songs playlist.
- **albums, playlists**: rate_playlist
- **artists**: subscribe_artists . This will add the artist to your Subscriptions tab. The Artists tab is determined by the songs/albums you have added to your library.
- **podcasts**: rate_playlist
- **episodes**: add_playlist_items("SE", episode_id)

How can I get the radio playlist for a song, video, playlist or album?

- **songs, videos**: `RDAMVM` + `videoId`
- **playlists, albums**: `RDAMPL` + `playlistId`

See also What is a browseId below.

How can I get the shuffle playlist for a playlist or album?

Use get_watch_playlist_shuffle with the `playlistId` or `audioPlaylistId` (albums).

How can I get all my public playlists in a single request?

Call get_user_playlists with your own `channelId`.

Can I download songs?

You can use youtube-dl for this purpose.

How do I package ytmusicapi with `pyinstaller`?

To package ytmusicapi correctly, you need to add the locales files to your executable.

You can use `--add-data path-to-ytmusicapi/locales` or `--collect-all ytmusicapi` to accomplish this.

5.4.3 YouTube Music API Internals

Is there a difference between songs and videos?

Yes. Videos are regular videos from YouTube, which can be uploaded by any user. Songs are actual songs uploaded by artists.

You can also add songs to your library, while you can't add videos.

Is there a rate limit?

There most certainly is, although you shouldn't run into it during normal usage. See related issues:

- Creating playlists
- Uploads

What is a browseId?

A `browseId` is an internal, globally unique identifier used by YouTube Music for browsable content.

In most cases you can compose it yourself:

Prefix	Main	Content
VM	playlistId	Playlist
RDAMVM	playlistId	Video-based Radio
RDAMPL	videoId	Playlist-based Radio
MPLA	channelId	Artist
MPRE	custom	Album
MPSP	playlistId	Podcast
MPED	videoId	Episode

Which videoTypes exist and what do they mean?

`videoType` is prefixed with `MUSIC_VIDEO_TYPE_`, i.e. `MUSIC_VIDEO_TYPE_OMV`. Currently the following variants of `videoType` are known:

- `OMV`: Original Music Video - uploaded by original artist with actual video content
- `UGC`: User Generated Content - uploaded by regular YouTube user
- `ATV`: High quality song uploaded by original artist with cover image
- `OFFICIAL_SOURCE_MUSIC`: Official video content, but not for a single track

Why is ytmusicapi returning more results than requested with the limit parameter?

YouTube Music always returns increments of a specific pagination value, usually between 20 and 100 items at a time. This is the case if a ytmusicapi method supports the `limit` parameter. The default value of the `limit` parameter indicates the server-side pagination increment. ytmusicapi will keep fetching continuations from the server until it has reached at least the `limit` parameter, and return all of these results.

Which values can I use for languages?

The `language` parameter determines the language of the returned results. `ytmusicapi` only supports a subset of the languages supported by YouTube Music, as translations need to be done manually. Contributions are welcome, see here for instructions.

For the list of values you can use for the `language` parameter, see below:

Language	Value
Arabic	ar
German	de
English (default)	en
Spanish	es
French	fr
Hindi	hi
Italian	it
Japanese	ja
Korean	ko
Dutch	nl
Portuguese	pt
Russian	ru
Turkish	tr
Urdu	ur
Chinese (Mainland)	zh_CN
Chinese (Taiwan)	zh_TW

Which values can I use for locations?

Pick a value from the list below for your desired location and pass it using the `location` parameter.

Location	Value
Algeria	DZ
Argentina	AR
Australia	AU
Austria	AT
Azerbaijan	AZ
Bahrain	BH
Bangladesh	BD
Belarus	BY
Belgium	BE
Bolivia	BO
Bosnia and Herzegovina	BA
Brazil	BR
Bulgaria	BG
Cambodia	KH
Canada	CA
Chile	CL
Colombia	CO
Costa Rica	CR
Croatia	HR
Cyprus	CY
Czechia	CZ
Denmark	DK
Dominican Republic	DO
Ecuador	EC
Egypt	EG
El Salvador	SV
Estonia	EE
Finland	FI
France	FR
Georgia	GE
Germany	DE
Ghana	GH
Greece	GR
Guatemala	GT
Honduras	HN
Hong Kong	HK
Hungary	HU
Iceland	IS
India	IN
Indonesia	ID
Iraq	IQ
Ireland	IE
Israel	IL
Italy	IT
Jamaica	JM
Japan	JP
Jordan	JO

continues on next page

Table 1 – continued from previous page

Kazakhstan	KZ
Kenya	KE
Kuwait	KW
Laos	LA
Latvia	LV
Lebanon	LB
Libya	LY
Liechtenstein	LI
Lithuania	LT
Luxembourg	LU
Malaysia	MY
Malta	MT
Mexico	MX
Montenegro	ME
Morocco	MA
Nepal	NP
Netherlands	NL
New Zealand	NZ
Nicaragua	NI
Nigeria	NG
North Macedonia	MK
Norway	NO
Oman	OM
Pakistan	PK
Panama	PA
Papua New Guinea	PG
Paraguay	PY
Peru	PE
Philippines	PH
Poland	PL
Portugal	PT
Puerto Rico	PR
Qatar	QA
Romania	RO
Russia	RU
Saudi Arabia	SA
Senegal	SN
Serbia	RS
Singapore	SG
Slovakia	SK
Slovenia	SI
South Africa	ZA
South Korea	KR
Spain	ES
Sri Lanka	LK
Sweden	SE
Switzerland	CH
Taiwan	TW
Tanzania	TZ
Thailand	TH
Tunisia	TN
Turkey	TR

continues on next page

Table 1 – continued from previous page

Uganda	UG
Ukraine	UA
United Arab Emirates	AE
United Kingdom	GB
United States	US
Uruguay	UY
Venezuela	VE
Vietnam	VN
Yemen	YE
Zimbabwe	ZW

PYTHON MODULE INDEX

y

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Y