

---

# asyncspotify Documentation

*Release 0.12.1*

**RUNIE**

**Oct 06, 2021**



---

## Contents:

---

<b>1</b>	<b>Quickstart</b>	<b>3</b>
1.1	Installing . . . . .	3
1.2	Getting started . . . . .	3
<b>2</b>	<b>Examples</b>	<b>7</b>
2.1	Getting Spotify Objects . . . . .	7
2.2	Getting playlists and adding tracks . . . . .	7
2.3	Searching . . . . .	8
2.4	Getting users/yourself . . . . .	8
<b>3</b>	<b>API Reference</b>	<b>9</b>
3.1	Client . . . . .	9
3.2	Spotify Objects . . . . .	16
3.2.1	Track . . . . .	16
3.2.2	Artist . . . . .	17
3.2.3	Playlist . . . . .	18
3.2.4	Album . . . . .	19
3.2.5	Audio Features . . . . .	21
3.2.6	Audio Analysis . . . . .	21
3.2.7	Image . . . . .	22
3.2.8	User . . . . .	22
3.2.9	Playing Objects . . . . .	23
3.2.10	Device . . . . .	24
3.3	Authenticators . . . . .	25
3.3.1	ClientCredentialsFlow . . . . .	25
3.3.2	EasyAuthorizationCodeFlow . . . . .	25
3.3.3	AuthorizationCodeFlow . . . . .	26
3.4	Scope . . . . .	26
3.5	Exceptions . . . . .	27
3.6	Utilities . . . . .	28
<b>4</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Index</b>	<b>31</b>



asyncspotify is an asynchronous, object-oriented wrapper for the Spotify Web API.

**Primary goals of this library:**

- easy and intuitive asynchronous interface
- object-oriented inheritance and design
- automatic rate limiting handling
- easy authentication



This page will guide into installing and setting up a client.

## 1.1 Installing

The recommended way of installing this library is from PyPI.

```
pip install asyncspotify
```

Run pip as a module to install for a specific version of Python:

```
python3.7 -m pip install asyncspotify
```

To test your installation, you can import `asyncspotify` and print the version string:

```
import asyncspotify
print(asyncspotify.__version__)
```

## 1.2 Getting started

To communicate with the Spotify Web API, you have to create a Spotify Application first. Go to [this page](#) and create an app.

After having made an app, you will be forwarded to a page showing miscellaneous stats. The client id and client secret provided here is what you'll use when authenticating. If you're going to use the `EasyAuthenticationCodeFlow` authorizer, you have to click edit and add `http://localhost/` to the list of redirect URIs.

To authenticate, you have to create an authenticator. If you do *not* need to access or modify personal data, you can simply use the `ClientCredentialsFlow` class:

```
import asyncio
import asyncspotify

async def main():
    # authenticate using the Client Credentials flow
    auth = asyncspotify.ClientCredentialsFlow(
        client_id='your client id',
        client_secret='your client secret',
    )

    # create and authorize your spotify client
    sp = asyncspotify.Client(auth)
    await sp.authorize()

    # done!
    playlist = await sp.get_playlist('1MG01HhbCvVhH9NmXhd9GC')
    async for track in playlist:
        print(track.name)

asyncio.run(main())
```

If you *do* need to access and modify personal data, you will have to use the Authentication Code flow and specify the *scope* you require. The easiest way to do this is to use the `EasyAuthenticationCodeFlow` class, which will handle storing your tokens and fetching them when your program restarts:

```
# this flow is "scoped", which means we have to list the resources we want access to.
# you can specify them like this, or do Scope.all() or Scope.none()
scope = asyncspotify.Scope(
    playlist_modify_public=True,
    playlist_modify_private=True,
    playlist_read_private=True,
    playlist_read_collaborative=True
)

# this is where our tokens will be stored
token_file = 'secret.json'

# create our authenticator
auth = asyncspotify.EasyAuthorizationCodeFlow(
    client_id='your client_id',
    client_secret='your client_secret',
    scope=scope,
    storage=token_file
)

# pass it to our new spotify client and authorize
sp = asyncspotify.Client(auth)
await sp.authorize()

# now we can do anything :)
print(await sp.get_me())
```

The `EasyAuthenticationCodeFlow` requires a first-time setup, please follow the steps in your console carefully. Remember to add `http://localhost/` to the list of redirect URIs on your Spotify Application page!

If you need more granular control of how tokens are stored, you can extend `AuthenticationCodeFlow` with



your own methods.

To see some basic usage of the API, see the examples. For everything else, see the API Reference.



# CHAPTER 2

## Examples

Here are some various examples on how to misc. operations.

### 2.1 Getting Spotify Objects

```
track = await sp.get_track('track_id')
# <FullTrack id='track_id' name='track name here'>

album = await sp.get_artist('artist_id')
# <FullArtist id='artist_id' name='artist name here'>

# and so on for playlists and albums...

# to get several instances:

albums = await sp.get_albums('album_id_1', 'album_id_2')
# [<FullAlbum id='album_id_1' ...>, <FullAlbum id='album_id_2' ...>]
```

### 2.2 Getting playlists and adding tracks

```
# get a playlist
playlist = await sp.get_playlist('1wPvaRtuI8mt10CpP2Kn1O')

# iterate tracks
async for track in playlist:
    print(track)

# get the user we're logged in as
me = await sp.get_me()
# <PrivateUser id='runiel3'>
```

(continues on next page)

(continued from previous page)

```
# create a playlist
my_playlist = await me.create_playlist(name='My playlist!', description='This is the_
↳playlist description.')
# <FullPlaylist id='0FECu9cwvviV3rwOjDqU9j' name='My playlist! '>

# add tracks from the first playlist to newly created second playlist
await my_playlist.add_tracks(*playlist.tracks)
```

## 2.3 Searching

```
# get multiple tracks from query
tracks = await sp.search_tracks(q='involvers', limit=2)
# [<SimpleTrack id='5xoJhWwvzPJD9k8j8BE2xO' name='27'>, <SimpleTrack id=
↳'0WUTBejxPUhURFCfFSYbDc' name='Fighting My Fight'>]

# get *one* track from query
track = await sp.search_track(q='norton commander')
# <SimpleTrack id='5KZiiK8dvTgXaVnegsvvBz' name='Norton Commander (All We Need) '>

# get one album from query
album = await sp.search_album('hiatus kaiyote')
# <SimpleAlbum id='3qzmmmRmVBiOuMvrerfW4z' name='Choose Your Weapon'>

# equivalent methods exist for track, album, artist and playlists.
# a general search() method also exists, but check the api reference first!
```

## 2.4 Getting users/yourself

```
# to get a user you can use the User getter:
user = await sp.get_user('user_id')
# <PublicUser id='user_id'>

# to get the user you're logged in as, use get_me:
me = await sp.get_me()
# <PrivateUser id='your_user_id'>

# user objects have methods themselves, such as create_playlist()
# check the api reference for a complete list!
```

## 3.1 Client

The Spotify API wrapper client itself.

You can start a client in a context using the `async with` statement, see below. Do note the client calls `Client.authenticate()` itself, so you don't have to do this.

```
async with asyncspotify.Client(auth) as sp:
    # your code here
```

Otherwise, you would start a client like this:

```
sp = asyncspotify.Client(auth)
await sp.authenticate()

# your code here..

# close the client session before you exit
await sp.close()
```

**class** `asyncspotify.Client(auth)`

Client interface for the API.

This is the class you should be interfacing with when fetching Spotify objects.

**auth: Authenticator** Authenticator instance used for authenticating with the API.

**\_\_init\_\_**(auth)

Creates a Spotify Client instance.

**Parameters** **auth** – Instance of `Authenticator`

**authorize**()

Tell the authenticator to authorize this client.

**close()**

Close this client session.

**create\_playlist** (*user*, *name*, *public=False*, *collaborative=False*, *description=None*) → `asyncspotify.playlist.FullPlaylist`

Create a new playlist.

#### Parameters

- **user** – User instance or Spotify ID.
- **name** (*str*) – Name of the new playlist.
- **description** (*str*) – Description of the new playlist.
- **public** (*bool*) – Whether the playlist should be public.
- **collaborative** (*bool*) – Whether the playlist should be collaborative (anyone can edit it).

**Returns** A `FullPlaylist` instance.

**edit\_playlist** (*playlist*, *name=None*, *description=None*, *public=None*, *collaborative=None*)

Edit a playlist.

#### Parameters

- **playlist** – Playlist instance or Spotify ID.
- **name** (*str*) – New name of the playlist.
- **description** (*str*) – New description of the playlist.
- **public** (*bool*) – New public state of the playlist.
- **collaborative** (*bool*) – New collaborative state of the playlist.

**following** (*type*, *\*ids*, *limit=None*, *after=None*)

Follow artists or users.

#### Parameters

- **type** (*str*) – The ID type: either artist or user.
- **ids** (*str*) – Spotify ID of the artist or the user.
- **limit** – Maximum number of items to return.
- **after** – The last artist ID retrieved from the previous request.

**get\_album** (*album\_id*, *market=None*) → `asyncspotify.album.FullAlbum`

Get an album.

#### Parameters

- **album\_id** (*str*) – Spotify ID of album.
- **market** – ISO-3166-1 country code.

**Returns** `FullAlbum` instance.

**get\_album\_tracks** (*album*, *limit=20*, *offset=None*, *market=None*) → `List[asyncspotify.track.SimpleTrack]`

Get tracks from an album.

#### Parameters

- **album** – Album or Spotify ID of album.

- **limit** – How many tracks to fetch.
- **offset** – What pagination offset to start from.
- **market** – ISO-3166-1 country code.

Returns List[*SimpleTrack*]

**get\_albums** (\*album\_ids, market=None) → List[asyncspotify.album.FullAlbum]  
Get several albums.

#### Parameters

- **album\_ids** (*str*) – Spotify ID of album.
- **market** – ISO-3166-1 country code.

Returns List[*FullAlbum*]

**get\_artist** (artist\_id) → asyncspotify.artist.FullArtist  
Get an artist.

**Parameters** **artist\_id** (*str*) – Spotify ID of artist.

**Returns** *FullArtist* instance.

**get\_artist\_albums** (artist\_id, limit=20, include\_groups=None, country=None, offset=None) → List[asyncspotify.album.SimpleAlbum]  
Get an artist's albums.

---

**Note:** This endpoint does *not* return the track objects for each album. If you need those, you have to fetch them manually afterwards.

---

#### Parameters

- **artist\_id** (*str*) – Spotify ID of artist.
- **limit** (*int*) – How many albums to return.
- **kwargs** – other query params for this method

Returns List[*SimpleAlbum*]

**get\_artist\_related\_artists** (artist) → List[asyncspotify.artist.FullArtist]  
Get an artists related artists.

**Parameters** **artist** – Artist or Spotify ID.

**Returns** A list of maximum 20 *FullArtist* instances.

**get\_artist\_top\_tracks** (artist, market=None) → List[asyncspotify.track.FullTrack]  
Returns the top tracks for an artist.

#### Parameters

- **artist** – Artist instance or Spotify ID.
- **market** – ISO-3166-1 country code. Leave blank to let the library auto-resolve this.

**Returns** A list of maximum 10 *FullTrack* instances.

**get\_artists** (\*artist\_ids) → List[asyncspotify.artist.FullArtist]  
Get several artists.

**Parameters** **artist\_ids** – List of artist Spotify IDs.

**Returns** List[*FullArtist*]

**get\_audio\_analysis** (*track*) → asyncspotify.audioanalysis.AudioAnalysis  
Get Audio Analysis of a track.

**Parameters** *track* – Track instance or Spotify ID.

**Returns** *AudioAnalysis*

**get\_audio\_features** (*track*) → asyncspotify.audiofeatures.AudioFeatures  
Get Audio Features of a track.

**Parameters** *track* – Track instance or Spotify ID.

**Returns** *AudioFeatures*

**get\_audio\_features\_multiple\_tracks** (*\*tracks*) → List[asyncspotify.audiofeatures.AudioFeatures]  
Get Audio Features for multiple tracks.

**Parameters** *tracks* (*str*) – Track or a comma seperated list of Spotify IDs

**Returns** list[*AudioFeatures*]

**get\_devices** () → List[asyncspotify.device.Device]  
Get a list of user devices.

**Returns** A list of maximum 20 devices.

**get\_followed\_artists** (*limit=20, after=None*) → List[asyncspotify.artist.SimpleArtist]  
Get user's followed artists

**Parameters**

- **limit** (*int*) – The maximum number of items to return. Default - infinity
- **after** – What artist ID to start the fetching from.

**Returns** List[*SimpleArtist*]

**get\_me** () → asyncspotify.user.PrivateUser  
Gets the current user.

**Returns** A *PrivateUser* instance of the current user.

**get\_me\_top\_artists** (*limit=20, offset=0, time\_range='medium\_term'*) → List[asyncspotify.artist.SimpleArtist]  
Get the top artists of the current user.

**Parameters**

- **limit** (*int*) – How many artists to return. Maximum is 50.
- **offset** (*int*) – The index of the first result to return.
- **time\_range** (*str*) – The time period for which data are selected to form a top.

**Valid values for time\_range**

- *long\_term* (calculated from several years of data and including all new data as it becomes available),
- *medium\_term* (approximately last 6 months),
- *short\_term* (approximately last 4 weeks).

**Returns** List[*SimpleArtist*]



**get\_me\_top\_tracks** (*limit=20*, *offset=None*, *time\_range=None*) → List[asyncspotify.track.SimpleTrack]

Gets the top tracks of the current user.

Requires scope `user-top-read`.

#### Parameters

- **limit** (*int*) – How many tracks to return. Maximum is 50.
- **offset** (*int*) – The index of the first result to return.
- **time\_range** (*str*) – The time period for which data are selected to form a top.

#### Valid values for **time\_range**

- `long_term` (calculated from several years of data and including all new data as it becomes available),
- `medium_term` (approximately last 6 months),
- `short_term` (approximately last 4 weeks).

**Returns** List[*SimpleTrack*]

**get\_player** (*\*\*kwargs*) → asyncspotify.playing.CurrentlyPlayingContext  
Get a context for what user is currently playing.

**Parameters** **kwargs** – query params for this request

**Returns** *PlayingContext*

**get\_playlist** (*playlist\_id*) → asyncspotify.playlist.FullPlaylist  
Get a pre-existing playlist.

**Parameters** **playlist\_id** (*str*) – Spotify ID of the playlist.

**Returns** *FullPlaylist* instance.

**get\_playlist\_tracks** (*playlist*) → List[asyncspotify.track.PlaylistTrack]  
Get tracks from a playlist.

**Parameters** **playlist** – *Playlist* instance or Spotify ID.

**Returns** List[*PlaylistTrack*]

**get\_track** (*track\_id*) → asyncspotify.track.FullTrack  
Get a track.

**Parameters** **track\_id** (*str*) – Spotify ID of track.

**Returns** *FullTrack* instance.

**get\_tracks** (*\*track\_ids*) → List[asyncspotify.track.FullTrack]  
Get several tracks.

**Parameters** **track\_ids** (*str*) – List of track Spotify IDs.

**Returns** List[*FullTrack*]

**get\_user** (*user\_id*) → asyncspotify.user.PublicUser  
Get a user.

**Parameters** **user\_id** (*str*) – Spotify ID of user.

**Returns** A *PublicUser* instance.

**get\_user\_playlists** (*user*) → List[asynspotify.playlist.SimplePlaylist]

Get a list of attainable playlists a user owns.

**Parameters** *user* – User instance or Spotify ID.

**Returns** List[SimplePlaylist]

**player\_next** (*device=None*)

Skip to the next track in a player.

**Parameters** *device* – Device or Spotify ID.

**player\_pause** (*device=None*)

Stop playback on a device.

**Parameters** *device* – Device or Spotify ID.

**player\_play** (*device=None*, *\*\*kwargs*)

Start playback on device.

**Parameters**

- **device** – Device or Spotify ID.
- **kwargs** – body params of the request.

Example:

```
player_play(
    context_uri='spotify:album:6xKK037rfCf2f6gf30SpvL',
    offset=dict(uri='spotify:track:2beor6qrB0XJxWlCM6X9x2'),
    position_ms=98500
)
```

**player\_prev** (*device=None*)

Play the previous track.

**Parameters** *device* – Device or Spotify ID.

**player\_repeat** (*state*, *device=None*)

Set player repeat mode.

**Parameters**

- **state** (*str*) – Can be ‘track’, ‘context’ or ‘off’.
- **device** – Device or Spotify ID.

**player\_seek** (*time*, *device=None*)

Seek to a point in the currently playing track.

**Parameters**

- **time** – timedelta object or milliseconds (integer)
- **device** – Device or Spotify ID.

**player\_shuffle** (*state*, *device=None*)

Set player shuffle mode.

**Parameters**

- **state** (*bool*) – Shuffle mode state.
- **device** – Device or Spotify ID.

**player\_volume** (*volume*, *device=None*)

Set player volume.

**Parameters**

- **volume** (*int*) – Value from 0 to 100.
- **device** – *Device* or Spotify ID.

**playlist\_add\_tracks** (*playlist*, *\*tracks*, *position=None*)

Add several tracks to a playlist.

**Parameters**

- **playlist** – *Playlist* instance or Spotify ID.
- **tracks** – List of Spotify IDs or *Track* instance (or a mix).
- **position** (*int*) – Position in the playlist to insert tracks.

**refresh** ()

Tell the authenticator to refresh this client, if applicable.

**search** (*\*types*, *q*, *limit=20*, *market=None*, *offset=None*, *include\_external=None*) → dict

Searches for tracks, artists, albums and/or playlists.

**Parameters**

- **types** – One or more of the strings *track*, *album*, *artist*, *playlist* or the class equivalents.
- **q** (*str*) – The search query. See Spotifys' query construction guide [here](#).
- **limit** (*int*) – How many results of each type to return.
- **market** – [ISO-3166-1](#) country code or the string *from\_token*.
- **offset** – Where to start the pagination.
- **include\_external** – If this is equal to *audio*, the specified the response will include any relevant audio content that is hosted externally.

**Returns** A dict with a key for each type, whose values are a list of instances.

**search\_album** (*q=None*) → *asyncspotify.album.SimpleAlbum*

Returns the top album for the query.

**Returns** *SimpleAlbum*

**search\_albums** (*q*, *limit=20*, *market=None*, *offset=None*, *include\_external=None*) →

List[*asyncspotify.album.SimpleAlbum*]

Alias for *Client.search('album', ...)*

**Returns** List[*SimpleAlbum*]

**search\_artist** (*q=None*) → *asyncspotify.artist.FullArtist*

Returns the top artist for the query.

**Returns** *SimpleArtist* or *None*

**search\_artists** (*q*, *limit=20*, *market=None*, *offset=None*, *include\_external=None*) →

List[*asyncspotify.artist.FullArtist*]

Alias for *Client.search('artist', ...)*

**Returns** List[*FullArtist*]

**search\_playlist** (*q=None*) → *asyncspotify.playlist.SimplePlaylist*

Returns the top playlist for the query.

Returns *SimplePlaylist*

**search\_playlists** (*q*, *limit*=20, *market*=None, *offset*=None, *include\_external*=None) → List[asyncspotify.playlist.SimplePlaylist]  
Alias for `Client.search('playlist', ...)`

Returns List[*SimplePlaylist*]

**search\_track** (*q*=None) → asyncspotify.track.SimpleTrack  
Returns the top track for the query.

Returns *SimpleTrack* or None

**search\_tracks** (*q*, *limit*=20, *market*=None, *offset*=None, *include\_external*=None) → List[asyncspotify.track.SimpleTrack]  
Alias for `Client.search('track', ...)`

Returns List[*SimpleTrack*]

## 3.2 Spotify Objects

---

**Note:** None of these objects should be instantiated manually. They are returned by convenience methods in *Client*.

---

**class** `asyncspotify.SpotifyObject` (*client*, *data*)

Represents a generic Spotify Object.

**id:** str Spotify ID of the object.

**name:** str Name of the object.

**uri:** str Spotify URI of the object.

### 3.2.1 Track

**class** `asyncspotify.SimpleTrack` (*client*, *data*)

Represents a Track object.

**id:** str Spotify ID of the track.

**name:** str Name of the track.

**artists:** List[*Artist*] List of artists that appear on the track.

**images:** List[*Image*] List of associated images, such as album cover in different sizes.

**uri:** str Spotify URI of the album.

**link:** str Spotify URL of the album.

**type:** str Plaintext string of object type: `track`.

**available\_markets:** List[str] or None Markets where the album is available in [ISO-3166-1](#) form.

**disc\_number:** int What disc the track appears on. Usually 1 unless there are several discs in the album.

**duration:** *timedelta* timedelta instance representing the length of the track.

**explicit:** bool Whether the track is explicit or not.

**external\_urls:** dict Dictionary that maps type to url.

**is\_playable:** bool tbc

**linked\_from:** `LinkedTrack` tbc

**restrictions:** restrictions object tbc

**preview\_url:** str An URL to a 30 second preview (MP3) of the track.

**track\_number:** int The number of the track on the album.

**is\_local:** bool Whether the track is from a local file.

**audio\_analysis** () → `asyncspotify.audioanalysis.AudioAnalysis`  
Get 'Audio Analysis' of the track.

**Parameters** **track** – `Track` instance or Spotify ID of track.

**Returns** `AudioAnalysis`

**audio\_features** () → `asyncspotify.audiofeatures.AudioFeatures`  
Get 'Audio Features' of the track.

**Parameters** **track** – `Track` instance or Spotify ID of track.

**Returns** `AudioFeatures`

**available\_in** (market)  
Check if track is available in a market.

**Parameters** **market** – ISO-3166-1 value.

**Returns**

**class** `asyncspotify.FullTrack` (client, data)  
Represents a complete Track object.

This type has some additional attributes not existent in `SimpleTrack`.

**album:** `SimpleAlbum` An instance of the album the track appears on.

**popularity:** int An indicator of the popularity of the track, 0 being least popular and 100 being the most.

**external\_ids:** dict Dictionary of external IDs.

**class** `asyncspotify.PlaylistTrack` (client, data)  
Represents a Track object originating from a playlist.

This type has some additional attributes not existent in `SimpleTrack` or `FullTrack`.

**added\_at:** datetime Indicates when the track was added to the playlist.

**added\_by:** User object Indicates who added the track to the playlist. The information provided from the API is not enough to instantiate a `PublicUser` object, so it's a plain copy of the returned json object.

### 3.2.2 Artist

**class** `asyncspotify.SimpleArtist` (client, data)  
Represents an Artist object.

**id:** str Spotify ID of the artist.

**name:** str Name of the artist.

**uri:** str Spotify URI of the artist.

**link:** str Spotify URL of the artist.

**external\_urls:** **dict** Dictionary that maps type to url.

**albums** (*limit=20, include\_groups=None, country=None, offset=None*)  
Get artists albums

**Returns** List[*SimpleAlbum*]

**related\_artists** ()  
Get related artists. Maximum of 20 artists.

**Returns** List[*FullArtist*]

**top\_tracks** (*market=None*)  
Returns this artists top tracks.

**Parameters** **market** – Market to find tracks for. Auto-resolved by the library if left blank.

**Returns** List[*FullTrack*]

**class** `asyncspotify.FullArtist` (*client, data*)  
Represents a complete Artist object.

This type has some additional attributes not existent in *SimpleArtist*.

**follow\_count:** **int** Follow count of the artist.

**genres:** **List[str]** Genres associated with the artist.

**popularity:** **int** An indicator of the popularity of the track, 0 being least popular and 100 being the most.

**images:** **List[Image]** List of associated images.

### 3.2.3 Playlist

**class** `asyncspotify.SimplePlaylist` (*client, data*)  
Represents a playlist object.

---

**Note:** To iterate all tracks, you have to use the `async for` construct or fill the object with `.fill()` before iterating `.tracks`.

---

**id:** **str** Spotify ID of the playlist.

**name:** **str** Name of the playlist.

**tracks:** **List[SimpleTrack]** All tracks in the playlist.

**track\_count:** **int** The expected track count as advertised by the last paging object. `is_filled()` can return True even if fewer tracks than this exists in `tracks`, since some fetched tracks from the API can be None for various reasons.

**uri:** **str** Spotify URI of the playlist.

**link:** **str** Spotify URL of the playlist.

**snapshot\_id:** **str** Spotify ID of the current playlist snapshot. Read about snapshots [here](#).

**collaborative:** **bool** Whether the playlist is collaborative.

**public:** **bool** Whether the playlist is public.

**owner:** *PublicUser* Owner of the playlist.

**external\_urls:** **dict** Dictionary that maps type to url.

**images:** List[[Image](#)] List of associated images.

**async for track in playlist**

Create a pager and iterate all tracks in this object. Also updates the `tracks` cache (same as calling `fill()`).

**add\_track** (*track*, *position=None*)

Add a track to the playlist.

**Parameters**

- **track** – Spotify ID or `Track` instance.
- **position** (*int*) – Position in the playlist to insert tracks.

**add\_tracks** (*\*tracks*, *position=None*)

Add several tracks to the playlist.

**Parameters**

- **tracks** – Several Spotify IDs or `Track` instances (or a mix).
- **position** (*int*) – Position in the playlist to insert tracks.

**edit** (*name=None*, *public=None*, *collaborative=None*, *description=None*)

Edit the playlist.

**Parameters**

- **name** (*str*) – New name of the playlist.
- **description** (*str*) – New description of the playlist.
- **public** (*bool*) – New public state of the playlist.
- **collaborative** (*bool*) – New collaborative state of the playlist.

**fill** ()

Update this objects `tracks` cache.

**has\_track** (*track*)

Check if this object has a track.

**is\_filled** ()

Whether this object contains as many tracks as advertised by the previous pager.

**class** `asyncspotify.FullPlaylist` (*client*, *data*)

Represents a complete playlist object.

This type has some additional attributes not existent in [SimplePlaylist](#).

**description:** `str` Description of the playlist, as set by the owner.

**primary\_color:** `str` Primary color of the playlist, for aesthetic purposes.

**follower\_count:** `int` Follower count of the playlist.

### 3.2.4 Album

**class** `asyncspotify.SimpleAlbum` (*client*, *data*)

Represents an Album object.

---

**Note:** To iterate all tracks, you have to use the `async for` construct or fill the object with `.fill()` before iterating `.tracks`.

---

**id:** `str` Spotify ID of the album.

**name:** `str` Name of the album.

**tracks:** `List[Track]` List of tracks on the album.

**artists:** `List[Artist]` List of artists that appear on the album.

**images:** `List[Image]` List of associated images, such as album cover in different sizes.

**track\_count:** `int` The expected track count as advertised by the last paging object. `is_filled()` can return `True` even if fewer tracks than this exists in `tracks`, since some fetched tracks from the API can be `None` for various reasons.

**uri:** `str` Spotify URI of the album.

**link:** `str` Spotify URL of the album.

**type:** `str` Plaintext string of object type: `album`.

**album\_type:** Type of album, e.g. `album`, `single` or `compilation`.

**available\_markets:** `List[str]` or `None` Markets where the album is available: [ISO-3166-1](#).

**external\_urls:** `dict` Dictionary that maps type to url.

**release\_date:** `datetime` Date (and maybe time) of album release.

**release\_date\_precision:** `str` Precision of `release_date`. Can be `year`, `month`, or `day`.

**album\_group:** `str` or `None` Type of album, e.g. `album`, `single`, `compilation` or `appears_on`.

**async for track in album**

Create a pager and iterate all tracks in this object. Also updates the `tracks` cache (same as calling `fill()`).

**fill()**

Update this objects `tracks` cache.

**has\_track(track)**

Check if this object has a track.

**is\_filled()**

Whether this object contains as many tracks as advertised by the previous pager.

**class** `asyncspotify.FullAlbum(client, data)`

Represents a complete Album object.

This type has some additional attributes not existent in [SimpleAlbum](#).

**genres:** `List[str]` List of genres associated with the album.

**label:** `str` The label for the album.

**popularity:** `int` An indicator of the popularity of the album, 0 being least popular and 100 being the most.

**copyrights:** `dict` List of copyright objects.

**external\_ids:** `dict` Dictionary of external IDs.



### 3.2.5 Audio Features

**class** `asyncspotify.AudioFeatures` (*client*, *data*)

Represents an Audio Features object.

**id:** `str` The Spotify ID of the track.

**uri:** `str` Spotify URI of the album.

**analysis\_url:** `str` An HTTP URL to access the full audio analysis of this track.

**track\_href:** `str` A link to the Web API endpoint providing full details of the track.

**duration:** `timedelta` The duration of the track.

**key:** `int` The estimated overall key of the track.

**mode:** `int` Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.

**time\_signature:** `int` An estimated overall time signature of a track.

**acousticness:** `float` A confidence measure from 0.0 to 1.0 of whether the track is acoustic.

**danceability:** `float` A measure of how suitable the track is for dancing.

**energy:** `float` Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.

**instrumentalness:** `float` Predicts whether a track contains no vocals.

**liveness:** `float` Detects the presence of an audience in the recording.

**loudness:** `float` The overall loudness of a track in decibels (dB).

**speechiness:** `float` Speechiness detects the presence of spoken words in a track.

**valence:** `float` A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.

**tempo:** `float` The overall estimated tempo of a track in beats per minute (BPM).

### 3.2.6 Audio Analysis

**class** `asyncspotify.AudioAnalysis` (*client*, *data*)

Represents an Audio Analysis object.

This page only skims the details on this object. *Please* read the official Spotify documentation [here](#).

**bars:** `List[TimeInterval]` The time intervals of the bars throughout the track. A bar (or measure) is a segment of time defined as a given number of beats.

**beats:** `List[TimeInterval]` The time intervals of beats throughout the track. A beat is the basic time unit of a piece of music; for example, each tick of a metronome.

**sections:** `List[Section]` List of sections of the track. Sections are defined by large variations in rhythm or timbre, e.g. chorus, verse, bridge, guitar solo, etc.

**segments:** `List[Segment]` List of audio segments of the track. Audio segments attempts to subdivide a song into many segments, with each segment containing a roughly consistent sound throughout its duration.

**tatums:** `List[TimeInterval]` The time intervals of tatums throughout the track. A tatum represents the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events (segments).

### 3.2.7 Image

**class** `asyncspotify.Image` (*data*)  
 Represents an image.

**url:** `str` URL of the image.

**width:** `int` Width of the image

**height:** `int` Height of the image.

### 3.2.8 User

**class** `asyncspotify.PublicUser` (*client, data*)  
 Represents a User object.

**id:** `str` Spotify ID of the user.

**name:** `str` Name of the user. Also aliased to the `display_name` attribute.

**images:** `List[Image]` List of associated images, such as the users profile picture.

**uri:** `str` Spotify URI of the user.

**link:** `str` Spotify URL of the user.

**follower\_count:** `int or None` Follower count of the user.

**external\_urls:** `dict` Dictionary that maps type to url.

**playlists** ()  
 Get the users playlists.

Alias of `Client.get_user_playlists()`

**class** `asyncspotify.PrivateUser` (*client, data*)  
 Represents a private User object, usually fetched through the `me` endpoint.

This type has some additional attributes not existent in `PublicUser`.

**country:** `str` ISO-3166-1 code of users country.

**email:** `str` Email of user. Please do not this email is note necessarily verified by Spotify.

**product:** `str` Users Spotify subscription level, could be `free`, `open` or `premium`. `free` and `open` are synonyms.

**create\_playlist** (*name, public=False, collaborative=False, description=None*)  
 Create a new playlist.

#### Parameters

- **name** (*str*) – Name of the new playlist.
- **description** (*str*) – Description of the new playlist.
- **public** (*bool*) – Whether the playlist should be public.
- **collaborative** (*bool*) – Whether the playlist should be collaborative (anyone can edit it).

**Returns** A `FullPlaylist` instance.

**playlists()**

Get the users playlists.

Alias of `Client.get_user_playlists()`

**top\_artists** (*limit=20, offset=None, time\_range=None*)

Get the top artists of the current user.

#### Parameters

- **limit** (*int*) – How many artists to return. Maximum is 50.
- **offset** (*int*) – The index of the first result to return.
- **time\_range** (*str*) – The time period for which data are selected to form a top.

#### Valid values for **time\_range**

- **long\_term** (calculated from several years of data and including all new data as it becomes available),
- **medium\_term** (approximately last 6 months),
- **short\_term** (approximately last 4 weeks).

**Returns** List[`SimpleArtist`]

**top\_tracks** (*limit=20, offset=None, time\_range=None*)

Gets the top tracks of the current user.

Requires scope `user-top-read`.

#### Parameters

- **limit** (*int*) – How many tracks to return. Maximum is 50.
- **offset** (*int*) – The index of the first result to return.
- **time\_range** (*str*) – The time period for which data are selected to form a top.

#### Valid values for **time\_range**

- **long\_term** (calculated from several years of data and including all new data as it becomes available),
- **medium\_term** (approximately last 6 months),
- **short\_term** (approximately last 4 weeks).

**Returns** List[`SimpleTrack`]

## 3.2.9 Playing Objects

**class** `asyncspotify.CurrentlyPlaying` (*client, data*)

Represents a Currently Playing object.

**timestamp:** `datetime` When the object information was created by the Spotify API.

**progress:** `timedelta` How far into the current track the player is.

**is\_playing:** `bool` Whether the track is playing or not.

**track:** `Track` What track is currently playing, can be `None`

**currently\_playing\_type:** `str` What is currently playing, can be `track`, `episode`, `ad` or `unknown`.

**class** `asynspotify.CurrentlyPlayingContext` (*client*, *data*)

Represents a Player object, extends `CurrentlyPlaying`

This type has some additional attributes not existent in `CurrentlyPlaying`.

**device:** `Device` What device is owns this context.

**repeat\_state:** `str` The repeat state of the player. Can be `off`, `track` or `context`.

**shuffle\_state:** `bool` The shuffle state of the player. Can be `True` or `False`.

**next** ()

Skips to the next track.

**pause** ()

Pauses playback.

**play** (*\*\*kwargs*)

Starts playback.

**Parameters** *kwargs* – Body parameters of the request.

```
player_play(
    context_uri='spotify:album:1Je1IMU1BXcx1Fz0WE7oPT',
    offset=dict(uri='spotify:track:1301WleyT98MSxVHPZCA6M'),
    position_ms=1000
)
```

**prev** ()

Goes to the previous track.

**repeat** (*state*)

Set player repeat mode.

**Parameters** *state* (*str*) – Can be ‘track’, ‘context’ or ‘off’.

**seek** (*time*)

Seeks to a specified time in the current track.

**Parameters** *time* – `timedelta` object or milliseconds (integer)

**shuffle** (*state*)

Set player shuffle mode.

**Parameters** *state* (*bool*) – Shuffle mode state.

**volume** (*volume*)

Set player volume.

**Parameters** *volume* (*int*) – Value from 0 to 100.

### 3.2.10 Device

**class** `asynspotify.Device` (*client*, *data*)

Represents a Device object.

**is\_active:** `bool` Whether this device is currently the active device.

**is\_private\_session:** `bool` If the device session is private.

**is\_restricted:** `bool` Whether controlling this device is restricted. If this is `true`, no API commands will work on it.

**name:** `str` Name of this device.

**type:** `str` Equal to `device`.

**volume\_percent:** `int` Volume of this device. Integer between 0 to 100.

## 3.3 Authenticators

A guide on how authentication works is located [here](#).

Examples can also be found under the quickstart guide.

---

**Note:** You do not have to worry about when your access token expires as the library will refresh the tokens automatically. Unless you're rolling your own authenticator, obviously.

---

### 3.3.1 ClientCredentialsFlow

Only requires a client id and secret to authenticate. Does *not* give access to private resources. No refresh token is used here. To extend, it simply authorizes again.

```
class asyncspotify.ClientCredentialsFlow(client_id, client_secret, re-
                                         sponse_class=<class 'asyncspo-
                                         tify.oauth.response.AuthenticationResponse'>)
```

Implements the Client Credentials flow.

You can only access public resources using this authenticator.

**authorize()**

Authorize using this authenticator.

**refresh(start\_task=True)**

Refresh this authenticator.

### 3.3.2 EasyAuthorizationCodeFlow

Extends `AuthorizationCodeFlow` and requires one extra argument, `storage`, which tells the authenticator which file to store tokens in.

```
class asyncspotify.EasyAuthorizationCodeFlow(client_id, client_secret, scope=<Scope
                                              value=0>, storage='secret.json',
                                              response_class=<class 'asyncspo-
                                              tify.oauth.response.AuthorizationCodeFlowResponse'>)
```

**authorize()**

Authorize the client. Reads from the file specified by `store`.

**create\_authorize\_route()**

Craft the `Route` for the user to use for authorizing the client.

**get\_code\_from\_redirect(url)**

Extract the authorization code from the redirect uri.

**refresh(start\_task=True)**

Refresh this authenticator.

### 3.3.3 AuthorizationCodeFlow

Exposes helper methods for implementing a version of the Authorization Code flow. *EasyAuthorizationCodeFlow* inherits from this and is recommended for most if access to private resources is required.

```
class asynspotify.AuthorizationCodeFlow(client_id, client_secret, scope, redirect_uri, response_class=<class 'asynspotify.oauth.response.AuthorizationCodeFlowResponse'>)
```

Implements the Authorization Code flow.

---

**Note:** This class is not for general use, please use *EasyAuthorizationCodeFlow* or subclass this and implement your own `load()`, `store(response)` and `setup()` methods.

---

**client\_id:** **str** Your application client id.

**client\_secret:** **str** Your application client secret.

**scope:** *Scope* The scope you're requesting.

**redirect\_uri:** **str** Where the user will be redirected to after accepting the client.

**response\_class:** The type that is expected to be returned from `load()` and `setup()`, and is passed to `store(response)` when a token refresh happens. Should be *AuthorizationCodeFlowResponse* or inherit from it.

**create\_authorize\_route()**

Craft the *Route* for the user to use for authorizing the client.

**get\_code\_from\_redirect(url)**

Extract the authorization code from the redirect uri.

**authorize()**

Authorize the client. Reads from the file specified by *store*.

**refresh(start\_task=True)**

Refresh this authenticator.

## 3.4 Scope

You can create a scope with specific permissions by passing kwargs in, like:

```
scope = Scope(
    user_top_read=True,
    playlist_modify_private=True
)
```

**class** asynspotify.**Scope**(*value=0*, **\*\*kwargs**)

Flags representing Spotify scopes.

**ugc\_image\_upload** Write access to user-provided images.

**user\_modify\_playback\_state** Write access to a user's playback state.

**user\_read\_playback\_state** Read access to a user's player state.

**user\_read\_currently\_playing** Read access to a user's currently playing content.

**user\_top\_read** Read access to a user's top artists and tracks.

**user\_read\_playback\_position** Read access to a user's playback position in a content.

**user\_read\_recently\_played** Read access to a user's recently played tracks.

**user\_library\_modify** Write/delete access to a user's "Your Music" library.

**user\_library\_read** Read access to a user's "Your Music" library.

**user\_follow\_modify** Write/delete access to the list of artists and other users that the user follows.

**user\_follow\_read** Read access to the list of artists and other users that the user follows.

**playlist\_read\_private** Read access to user's private playlists.

**playlist\_modify\_public** Write access to a user's public playlists.

**playlist\_modify\_private** Write access to a user's private playlists.

**playlist\_read\_collaborative** Include collaborative playlists when requesting a user's playlists.

**user\_read\_private** Read access to user's subscription details (type of user account).

**user\_read\_email** Read access to user's email address.

**app\_remote\_control** Remote control playback of Spotify. This scope is currently available to Spotify iOS and Android SDKs.

**streaming** Control playback of a Spotify track. This scope is currently available to the Web Playback SDK. The user must have a Spotify Premium account.

**all()**  
Return *Scope* with all scopes enabled.

**none()**  
Return *Scope* with no scopes enabled.

**string()**  
Get a string representation of the enabled scopes. Used when authenticating.

## 3.5 Exceptions

**class** `asyncspotify.SpotifyException`  
Base exception of all exceptions thrown by this library.

**class** `asyncspotify.HTTPException` (*response*, *message=None*)  
Bases: `asyncspotify.exceptions.SpotifyException`  
Base exception of all HTTP related exceptions.

**response:** `aiohttp.ClientResponse` The response of the failed HTTP request.

**message:** `Optional[str]` Message about what went wrong.

**class** `asyncspotify.BadRequest` (*response*, *message=None*)  
Bases: `asyncspotify.exceptions.HTTPException`  
400 Bad Request  
Base class for all 4xx status code exceptions.

```
class asyncspotify.Unauthorized (response, message=None)
    Bases: asyncspotify.exceptions.BadRequest
    401 Unauthorized

class asyncspotify.Forbidden (response, message=None)
    Bases: asyncspotify.exceptions.BadRequest
    403 Forbidden

class asyncspotify.NotFound (response, message=None)
    Bases: asyncspotify.exceptions.BadRequest
    404 Not Found

class asyncspotify.NotAllowed (response, message=None)
    Bases: asyncspotify.exceptions.BadRequest
    405 Method Not Allowed
```

## 3.6 Utilities

asyncspotify.utils.**get** (*items, \*\*kwargs*)  
 Get an item from a list of items.

### Parameters

- **items** – List or iterator containing `Object` s
- **kwargs** – kwargs that should match with the objects attributes.

**Returns** First item that matched.

asyncspotify.utils.**find** (*items, \*\*kwargs*)  
 Same as `get ()` except it returns a list of all matching items.

### Parameters

- **items** – List or iterator containing `Object`
- **kwargs** – kwargs that should match with the objects attributes.

**Returns** `List[Object]`



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



## Symbols

`__init__()` (*asyncspotify.Client* method), 9

## A

`add_track()` (*asyncspotify.SimplePlaylist* method), 19

`add_tracks()` (*asyncspotify.SimplePlaylist* method), 19

`albums()` (*asyncspotify.SimpleArtist* method), 18

`all()` (*asyncspotify.Scope* method), 27

`audio_analysis()` (*asyncspotify.SimpleTrack* method), 17

`audio_features()` (*asyncspotify.SimpleTrack* method), 17

`AudioAnalysis` (class in *asyncspotify*), 21

`AudioFeatures` (class in *asyncspotify*), 21

`AuthorizationCodeFlow` (class in *asyncspotify*), 26

`authorize()` (*asyncspotify.AuthorizationCodeFlow* method), 26

`authorize()` (*asyncspotify.Client* method), 9

`authorize()` (*asyncspotify.ClientCredentialsFlow* method), 25

`authorize()` (*asyncspotify.EasyAuthorizationCodeFlow* method), 25

`avaliabile_in()` (*asyncspotify.SimpleTrack* method), 17

## B

`BadRequest` (class in *asyncspotify*), 27

## C

`Client` (class in *asyncspotify*), 9

`ClientCredentialsFlow` (class in *asyncspotify*), 25

`close()` (*asyncspotify.Client* method), 9

`create_authorize_route()` (*asyncspotify.AuthorizationCodeFlow* method), 26

`create_authorize_route()` (*asyncspotify.EasyAuthorizationCodeFlow* method), 25

`create_playlist()` (*asyncspotify.Client* method), 10

`create_playlist()` (*asyncspotify.PrivateUser* method), 22

`CurrentlyPlaying` (class in *asyncspotify*), 23

`CurrentlyPlayingContext` (class in *asyncspotify*), 24

## D

`Device` (class in *asyncspotify*), 24

## E

`EasyAuthorizationCodeFlow` (class in *asyncspotify*), 25

`edit()` (*asyncspotify.SimplePlaylist* method), 19

`edit_playlist()` (*asyncspotify.Client* method), 10

## F

`fill()` (*asyncspotify.SimpleAlbum* method), 20

`fill()` (*asyncspotify.SimplePlaylist* method), 19

`find()` (in module *asyncspotify.utils*), 28

`following()` (*asyncspotify.Client* method), 10

`Forbidden` (class in *asyncspotify*), 28

`FullAlbum` (class in *asyncspotify*), 20

`FullArtist` (class in *asyncspotify*), 18

`FullPlaylist` (class in *asyncspotify*), 19

`FullTrack` (class in *asyncspotify*), 17

## G

`get()` (in module *asyncspotify.utils*), 28

`get_album()` (*asyncspotify.Client* method), 10

`get_album_tracks()` (*asyncspotify.Client* method), 10

`get_albums()` (*asyncspotify.Client* method), 11

`get_artist()` (*asyncspotify.Client* method), 11

`get_artist_albums()` (*asynspotify.Client method*), 11  
`get_artist_related_artists()` (*asynspotify.Client method*), 11  
`get_artist_top_tracks()` (*asynspotify.Client method*), 11  
`get_artists()` (*asynspotify.Client method*), 11  
`get_audio_analysis()` (*asynspotify.Client method*), 12  
`get_audio_features()` (*asynspotify.Client method*), 12  
`get_audio_features_multiple_tracks()` (*asynspotify.Client method*), 12  
`get_code_from_redirect()` (*asynspotify.AuthorizationCodeFlow method*), 26  
`get_code_from_redirect()` (*asynspotify.EasyAuthorizationCodeFlow method*), 25  
`get_devices()` (*asynspotify.Client method*), 12  
`get_followed_artists()` (*asynspotify.Client method*), 12  
`get_me()` (*asynspotify.Client method*), 12  
`get_me_top_artists()` (*asynspotify.Client method*), 12  
`get_me_top_tracks()` (*asynspotify.Client method*), 12  
`get_player()` (*asynspotify.Client method*), 13  
`get_playlist()` (*asynspotify.Client method*), 13  
`get_playlist_tracks()` (*asynspotify.Client method*), 13  
`get_track()` (*asynspotify.Client method*), 13  
`get_tracks()` (*asynspotify.Client method*), 13  
`get_user()` (*asynspotify.Client method*), 13  
`get_user_playlists()` (*asynspotify.Client method*), 13

## H

`has_track()` (*asynspotify.SimpleAlbum method*), 20  
`has_track()` (*asynspotify.SimplePlaylist method*), 19  
`HTTPException` (*class in asynspotify*), 27

## I

`Image` (*class in asynspotify*), 22  
`is_filled()` (*asynspotify.SimpleAlbum method*), 20  
`is_filled()` (*asynspotify.SimplePlaylist method*), 19

## N

`next()` (*asynspotify.CurrentlyPlayingContext method*), 24  
`none()` (*asynspotify.Scope method*), 27  
`NotAllowed` (*class in asynspotify*), 28  
`NotFound` (*class in asynspotify*), 28

## P

`pause()` (*asynspotify.CurrentlyPlayingContext method*), 24  
`play()` (*asynspotify.CurrentlyPlayingContext method*), 24  
`player_next()` (*asynspotify.Client method*), 14  
`player_pause()` (*asynspotify.Client method*), 14  
`player_play()` (*asynspotify.Client method*), 14  
`player_prev()` (*asynspotify.Client method*), 14  
`player_repeat()` (*asynspotify.Client method*), 14  
`player_seek()` (*asynspotify.Client method*), 14  
`player_shuffle()` (*asynspotify.Client method*), 14  
`player_volume()` (*asynspotify.Client method*), 14  
`playlist_add_tracks()` (*asynspotify.Client method*), 15  
`playlists()` (*asynspotify.PrivateUser method*), 22  
`playlists()` (*asynspotify.PublicUser method*), 22  
`PlaylistTrack` (*class in asynspotify*), 17  
`prev()` (*asynspotify.CurrentlyPlayingContext method*), 24  
`PrivateUser` (*class in asynspotify*), 22  
`PublicUser` (*class in asynspotify*), 22

## R

`refresh()` (*asynspotify.AuthorizationCodeFlow method*), 26  
`refresh()` (*asynspotify.Client method*), 15  
`refresh()` (*asynspotify.ClientCredentialsFlow method*), 25  
`refresh()` (*asynspotify.EasyAuthorizationCodeFlow method*), 25  
`related_artists()` (*asynspotify.SimpleArtist method*), 18  
`repeat()` (*asynspotify.CurrentlyPlayingContext method*), 24

## S

`Scope` (*class in asynspotify*), 26  
`search()` (*asynspotify.Client method*), 15  
`search_album()` (*asynspotify.Client method*), 15  
`search_albums()` (*asynspotify.Client method*), 15  
`search_artist()` (*asynspotify.Client method*), 15  
`search_artists()` (*asynspotify.Client method*), 15  
`search_playlist()` (*asynspotify.Client method*), 15  
`search_playlists()` (*asynspotify.Client method*), 16  
`search_track()` (*asynspotify.Client method*), 16  
`search_tracks()` (*asynspotify.Client method*), 16  
`seek()` (*asynspotify.CurrentlyPlayingContext method*), 24  
`shuffle()` (*asynspotify.CurrentlyPlayingContext method*), 24

SimpleAlbum (*class in asyncspotify*), 19  
 SimpleArtist (*class in asyncspotify*), 17  
 SimplePlaylist (*class in asyncspotify*), 18  
 SimpleTrack (*class in asyncspotify*), 16  
 SpotifyException (*class in asyncspotify*), 27  
 SpotifyObject (*class in asyncspotify*), 16  
 string() (*asyncspotify.Scope method*), 27

## T

top\_artists() (*asyncspotify.PrivateUser method*),  
 23  
 top\_tracks() (*asyncspotify.PrivateUser method*), 23  
 top\_tracks() (*asyncspotify.SimpleArtist method*), 18

## U

Unauthorized (*class in asyncspotify*), 27

## V

volume() (*asyncspotify.CurrentlyPlayingContext  
 method*), 24