# spotify.py

***Release 0.10.2***

**Apr 09, 2021**

# Contents:

**1 What is *spotify.py*?** **3**
1.1 Quick example . . . . . 3
1.1.1 Introduction . . . . . 3
1.1.2 API . . . . . 6

**2 Indices and tables** **43**

**Index** **45**

# CHAPTER 1

# What is *spotify.py*?

Spotify.py is a modern, friendly, and Pythonic API library for the Spotify API.

## 1.1 Quick example

This example shows effectively using the library to iterate over an albums tracks.

```python
import asyncio

import spotify

ALBUM_URI: str = "foo bar baz"
CLIENT_ID: str = "lorem ipsum"
CLIENT_SECRET: str = "dolor sit amet"

async def main(ident: str, secret: str, album_uri: str) -> None:
    # Useful tip: use a context manager to handle
    # automatically closing any underlying http sessions
    async with spotify.Client(ident, secret) as client:
        album = await client.get_album(album_uri)

        async for track in album:
            print(repr(track))

asyncio.run(main(CLIENT_ID, CLIENT_SECRET, ALBUM_URI))
```

### 1.1.1 Introduction

#### Getting Started

## API Coverage

Currently the library offers full http api coverage, this includes the regular REST API and the Connect Web API (used for maniplulating playback smoothly.)

If there is missing coverage of a Spotify API feature feel free to open a Github Issue and we can sort out the implementation from there.

## Concepts

## Abstractions!

The library is abstracted into mainly three components:

- The very low level (`spotify.http`)
- The very high level (`spotify.models`, `spotify.oauth` and `spotify.utils`)
- The synchronous interface (`spotify.sync`)

### `spotify.http`

The HTTP submodule is ultimately comprised of two main components:

- `spotify.http.HTTPClient`
- `spotify.http.HTTPUserClient`

### `spotify.models`

All the models are located under `spotify.models`.

- `spotify.SpotifyBase`
- `spotify.URIBase`
- `spotify.Device`
- `spotify.Context`
- `spotify.Image`
- `spotify.Artist`
- `spotify.Track`
- `spotify.PlaylistTrack`
- `spotify.Player`
- `spotify.Album`
- `spotify.Library`
- `spotify.Playlist`
- `spotify.User`

#### `spotify.oauth`

The oauth module concerns itself will all OAuth2 related logic.

- `spotify.OAuth2`
- `spotify.get_required_scopes`

#### `spotify.utils`

The utils module aims to provide usefull helpers.

- `spotify.to_id`

#### `spotify.sync`

The sync module aims to provide a one to one interface with the regular module. Whilst hiding any *async/await* shennanigans so that users don't need to be restricted by their executing environment.

## Guidelines

### Writing a Query

Queries are mainly done through *`spotify.Client.search()`*.

### Keyword matching

Matching of search keywords is not case-sensitive. Operators, however, should be specified in uppercase. Unless surrounded by double quotation marks, keywords are matched in any order.

For example:

- `q="roadhouse blues"` matches both "`Blues Roadhouse`" and "`Roadhouse of the Blues`".
- `q="\"roadhouse blues\""` matches "`My Roadhouse Blues`" but not "`Roadhouse of the Blues`".

### Searching

Searching for playlists returns results where the query keyword(s) match any part of the playlist's name or description. Only popular public playlists are returned.

### Operators

**Note:** Operators must be specified in uppercase. Otherwise, they are handled as normal keywords to be matched.

The operator `NOT` can be used to exclude results.

For example: `q="roadhouse NOT blues"` returns items that match "`roadhouse`" but excludes those that also contain the keyword "`blues`".

Similarly, the `OR` operator can be used to broaden the search: `q="roadhouse OR blues"` returns all the results that include either of the terms.

**Warning:** Only one `OR` operator can be used in a query.

### Wildcards

The asterisk (`*`) character can, with some limitations, be used as a wildcard (maximum: 2 per query). It matches a variable number of non-white-space characters.

It cannot be used:

- in a quoted phrase
- in a field filter
- when there is a dash (`-`) in the query
- or as the first character of the keyword string Field filters: By default, results are returned when a match is found in any field of the target object type. Searches can be made more specific by specifying an album, artist or track field filter.

For example: The query `q="album:gold artist:abba", types=["album"]` returns only albums with the text "`gold`" in the album name and the text "`abba`" in the artist name.

To limit the results to a particular year, use the field filter year with album, artist, and track searches.

For example: `q="bob year:2014"`

Or with a date range. For example: `q="bob year:1980-2020"`

To retrieve only albums released in the last two weeks, use the field filter `tag:new` in album searches.

To retrieve only albums with the lowest 10% popularity, use the field filter `tag:hipster` in album searches.

**Note:** This field filter only works with album searches.

Depending on object types being searched for, other field filters, include genre (applicable to tracks and artists), upc, and isrc. For example: `q="lil genre:\"southern hip hop\"", types=["artist"]`. Use double quotation marks around the genre keyword string if it contains spaces.

## 1.1.2 API

### Client

**class** `spotify.`**`Client`**(*client_id: str*, *client_secret: str*, ***, *loop: Optional[asyncio.events.AbstractEventLoop] = None*)

Represents a Client app on Spotify.

This class is used to interact with the Spotify API.

**Parameters**

- **`client_id`** (`str`) – The client id provided by spotify for the app.

- **`client_secret`** (`str`) – The client secret for the app.
- **`loop`** (Optional[`asyncio.AbstractEventLoop`]) – The event loop the client should run on, if no loop is specified *asyncio.get_event_loop()* is called and used instead.

**`client_id`**
The applications client_id, also aliased as *id*

**Type** `str`

**`http`**
The HTTPClient that is being used.

**Type** *`HTTPClient`*

**`loop`**
The event loop the client is running on.

**Type** Optional[`asyncio.AbstractEventLoop`]

**`client_id`**
`str` - The Spotify client ID.

**`close`**`()` → None
Close the underlying HTTP session to Spotify.

**`get_album`**(*spotify_id: str*, ***, *market: str = 'US'*) → spotify.models.album.Album
Retrive an album with a spotify ID.

**Parameters**

- **`spotify_id`** (`str`) – The ID to search for.
- **`market`** (Optional[`str`]) – An ISO 3166-1 alpha-2 country code

**Returns** **album** – The album from the ID

**Return type** *`spotify.Album`*

**`get_albums`**(**ids*, *market: str = 'US'*) → List[spotify.models.album.Album]
Retrive multiple albums with a list of spotify IDs.

**Parameters**

- **`ids`** (*`List[str]`*) – the ID to look for
- **`market`** (*`Optional[str]`*) – An ISO 3166-1 alpha-2 country code

**Returns** **albums** – The albums from the IDs

**Return type** List[*Album*]

**`get_artist`**(*spotify_id: str*) → spotify.models.artist.Artist
Retrive an artist with a spotify ID.

**Parameters** **`spotify_id`** (*`str`*) – The ID to search for.

**Returns** **artist** – The artist from the ID

**Return type** *Artist*

**`get_artists`**(**ids*) → List[spotify.models.artist.Artist]
Retrive multiple artists with a list of spotify IDs.

**Parameters** **`ids`** (List[`str`]) – The IDs to look for.

**Returns** **artists** – The artists from the IDs

**Return type** List[*Artist*]

**get_episode** (*id: str*, *market: Optional[str] = 'US'*) → spotify.models.podcast.Episode
Get Spotify catalog information for a single episode identified by its unique Spotify ID.

**Parameters**

- **spotify_id** (*str*) – The spotify_id to for the show.
- **market** (*Optional[str]*) – An ISO 3166-1 alpha-2 country code.

**Returns** The episode of the given ID.

**Return type** episode `Episode`

**get_multiple_shows** (*ids: List[str], market: Optional[str] = 'US'*) → List[spotify.models.podcast.Show]
Get Spotify catalog information for several shows based on their Spotify IDs.

**Parameters**

- **ids** (List[`str`]) – A list of the Spotify IDs.
- **market** (*Optional[str]*) – An ISO 3166-1 alpha-2 country code.

**Returns** **shows** – The shows from given IDs.

**Return type** List[:class: *Show*]

**get_track** (*spotify_id: str*) → spotify.models.track.Track
Retrive an track with a spotify ID.

**Parameters** **spotify_id** (*str*) – The ID to search for.

**Returns** **track** – The track from the ID

**Return type** Track

**get_user** (*spotify_id: str*) → spotify.models.user.User
Retrive an user with a spotify ID.

**Parameters** **spotify_id** (*str*) – The ID to search for.

**Returns** **user** – The user from the ID

**Return type** *User*

**id**
`str` - The Spotify client ID.

**oauth2_url** (*redirect_uri: str, scopes: Union[Iterable[str], Dict[str, bool], None] = None, state: Optional[str] = None*) → str
Generate an oauth2 url for user authentication.

This is an alias to `OAuth2.url_only()` but the difference is that the client id is autmatically passed in to the constructor.

**Parameters**

- **redirect_uri** (`str`) – Where spotify should redirect the user to after authentication.
- **scopes** (Optional[Iterable[`str`], Dict[`str`, `bool`]]) – The scopes to be requested.
- **state** (Optional[`str`]) – Using a state value can increase your assurance that an incoming connection is the result of an authentication request.

**Returns** **url** – The OAuth2 url.

**Return type** `str`

**search** (*q: str*, ***, *types: Iterable[str] = ('track', 'playlist', 'artist', 'album')*, *limit: int = 20*, *offset: int = 0*, *market: str = 'US'*, *should_include_external: bool = False*) → spotify.client.SearchResults

Access the spotify search functionality.

```
>>> results = client.search('Cadet', types=['artist'])
>>> for artist in result.get('artists', []):
...     if artist.name.lower() == 'cadet':
...         print(repr(artist))
...         break
```

**Parameters**

- **q** (`str`) – the search query
- **types** (Optional[Iterable[*:class:'str*]]) – A sequence of search types (can be any of *track*, *playlist*, *artist* or *album*) to refine the search request. A *ValueError* may be raised if a search type is found that is not valid.
- **limit** (Optional[`int`]) – The limit of search results to return when searching. Maximum limit is 50, any larger may raise a *`HTTPException`*
- **offset** (Optional[`int`]) – The offset from where the api should start from in the search results.
- **market** (Optional[`str`]) – An ISO 3166-1 alpha-2 country code. Provide this parameter if you want to apply Track Relinking.
- **should_include_external** (`bool`) – If *True* is specified, the response will include any relevant audio content that is hosted externally. By default external content is filtered out from responses.

**Returns** **results** – The results of the search.

**Return type** `SearchResults`

**Raises**

- `TypeError` – Raised when a parameter with a bad type is passed.
- `ValueError` – Raised when a bad search type is passed with the *types* argument.

**user_from_token** (*token: str*) → spotify.models.user.User

Create a user session from a token.

**Note:** This code is equivelent to *User.from_token(client, token)*

**Parameters** **token** (`str`) – The token to attach the user session to.

**Returns** **user** – The user from the ID

**Return type** *`spotify.User`*

## HTTPClient

**class** spotify.**HTTPClient** (*client_id: str*, *client_secret: str*, *loop=None*)

A class responsible for handling all HTTP logic.

This class combines a small amount of stateful logic control with the *`request()`* method and a very thin wrapper over the raw HTTP API.

All endpoint methods mirror the default arguments the API uses and is best described as a series of "good defaults" for the routes.

**Parameters**

- **`client_id`** (`str`) – The client id provided by spotify for the app.
- **`client_secret`** (`str`) – The client secret for the app.
- **`loop`** (`Optional[event loop]`) – The event loop the client should run on, if no loop is specified *asyncio.get_event_loop()* is called and used instead.

**`loop`**

The loop the client is running with.

**Type** AbstractEventLoop

**`client_id`**

The client id of the app.

**Type** str

**`client_secret`**

The client secret.

**Type** str

**`add_playlist_tracks`** (*playlist_id: str, tracks: Sequence[str], position: Optional[int] = None*) → Awaitable[T_co]

Add one or more tracks to a user's playlist.

**Parameters**

- **`playlist_id`** (`str`) – The Spotify ID for the playlist.
- **`tracks`** (Sequence[Union[`str`]]) – A sequence of track URIs.
- **`position`** (Optional[`int`]) – The position to insert the tracks, a zero-based index.

**`album`** (*spotify_id: str, market: Optional[str] = 'US'*) → Awaitable[T_co]

Get Spotify catalog information for a single album.

**Parameters**

- **`spotify_id`** (`str`) – The spotify_id to search by.
- **`market`** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**`album_tracks`** (*spotify_id: str, limit: Optional[int] = 20, offset: Optional[int] = 0, market='US'*) → Awaitable[T_co]

Get Spotify catalog information about an album's tracks.

**Parameters**

- **`spotify_id`** (`str`) – The spotify_id to search by.
- **`limit`** (`Optional[int]`) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **`offset`** (`Optiona[int]`) – The offset of which Spotify should start yielding from.
- **`market`** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**`albums`** (*spotify_ids, market='US'*) → Awaitable[T_co]

Get Spotify catalog information for multiple albums identified by their Spotify IDs.

**Parameters**

- **spotify_ids** (*List[str]*) – The spotify_ids to search by.
- **market** (*Optional[str]*) – An ISO 3166-1 alpha-2 country code.

**artist** (*spotify_id*) → Awaitable[T_co]
Get Spotify catalog information for a single artist identified by their unique Spotify ID.

**Parameters** **spotify_id** (*str*) – The spotify_id to search by.

**artist_albums** (*spotify_id*, *include_groups=None*, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*, *market='US'*)
Get Spotify catalog information about an artist's albums.

**Parameters**

- **spotify_id** (*str*) – The spotify_id to search by.
- **include_groups** (*INCLUDE_GROUPS_TP*) – INCLUDE_GROUPS
- **limit** (*Optional[int]*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optiona[int]*) – The offset of which Spotify should start yielding from.
- **market** (*Optional[str]*) – An ISO 3166-1 alpha-2 country code.

**artist_related_artists** (*spotify_id*) → Awaitable[T_co]
Get Spotify catalog information about artists similar to a given artist.

Similarity is based on analysis of the Spotify community's listening history.

**Parameters** **spotify_id** (*str*) – The spotify_id to search by.

**artist_top_tracks** (*spotify_id*, *country*) → Awaitable[T_co]
Get Spotify catalog information about an artist's top tracks by country.

**Parameters**

- **spotify_id** (*str*) – The spotify_id to search by.
- **country** (*COUNTRY_TP*) – COUNTRY

**artists** (*spotify_ids*) → Awaitable[T_co]
Get Spotify catalog information for several artists based on their Spotify IDs.

**Parameters** **spotify_id** (*List[str]*) – The spotify_ids to search with.

**audio_features** (*track_ids: List[str]*) → Awaitable[T_co]
Get audio features for multiple tracks based on their Spotify IDs.

**Parameters** **track_ids** (List[str]) – A comma-separated list of the Spotify IDs for the tracks. Maximum: 100 IDs.

**available_devices** () → Awaitable[T_co]
Get information about a user's available devices.

**categories** (*limit: Optional[int] = 20*, *offset: Optional[int] = 0*, *country=None*, *locale=None*) → Awaitable[T_co]
Get a list of categories used to tag items in Spotify (on, for example, the Spotify player's "Browse" tab).

**Parameters**

- **limit** (*Optional[int]*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optional[int]*) – The index of the first item to return. Default: 0
- **country** (*COUNTRY_TP*) – COUNTRY

- **locale** (*LOCALE_TP*) – LOCALE

**category** (*category_id*, *country=None*, *locale=None*) → Awaitable[T_co]
Get a single category used to tag items in Spotify (on, for example, the Spotify player's "Browse" tab).

**Parameters**

- **category_id** (`str`) – The Spotify category ID for the category.
- **country** (*COUNTRY_TP*) – COUNTRY
- **locale** (*LOCALE_TP*) – LOCALE

**category_playlists** (*category_id*, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*, *country=None*) → Awaitable[T_co]
Get a list of Spotify playlists tagged with a particular category.

**Parameters**

- **category_id** (`str`) – The Spotify category ID for the category.
- **limit** (`Optional[int]`) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (`Optional[int]`) – The index of the first item to return. Default: 0
- **country** (*COUNTRY_TP*) – COUNTRY

**change_playlist_details** (*playlist_id: str*, ***, *name: Optional[str] = None*, *public: Optional[bool] = None*, *collaborative: Optional[bool] = None*, *description: Optional[str] = None*) → Awaitable[T_co]
Change a playlist's name and public/private state. (The user must, of course, own the playlist.)

**Parameters**

- **playlist_id** (`str`) – The Spotify ID for the playlist.
- **name** (`str`) – The name for the new playlist
- **public** (Optional[`bool`]) – Defaults to true . If true the playlist will be public, if false it will be private
- **collaborative** (Optional[`bool`]) – Defaults to false . If true the playlist will be collaborative.

**Note:** to create a collaborative playlist you must also set public to false

- **description** (Optional[`str`]) – The value for playlist description as displayed in Spotify Clients and in the Web API.

**check_saved_shows** (*ids: List[str]*) → Awaitable[T_co]
Check if one or more shows is already saved in the current Spotify user's library.

**Parameters** **ids** (List[`str`]) – A list of the Spotify IDs.

**close** ()
Close the underlying HTTP session.

**create_playlist** (*user_id: str*, ***, *name: str*, *public: Optional[bool] = True*, *collaborative: Optional[bool] = False*, *description: Optional[str] = ''*) → Awaitable[T_co]
Create a playlist for a Spotify user. (The playlist will be empty until you add tracks.)

**Parameters**

- **user_id** (`str`) – The user's Spotify user ID.

- **name** (str) – The name for the new playlist
- **public** (Optional[bool]) – Defaults to true . If true the playlist will be public, if false it will be private
- **collaborative** (Optional[bool]) – Defaults to false . If true the playlist will be collaborative.

**Note:** to create a collaborative playlist you must also set public to false

- **description** (Optional[str]) – The value for playlist description as displayed in Spotify Clients and in the Web API.

**current_player**(**, market: Optional[str] = None*) → Awaitable[T_co]
Get information about the user's current playback state, including track, track progress, and active device.

**Parameters market** (Optional[str]) – An ISO 3166-1 alpha-2 country code or the string from_token. Provide this parameter if you want to apply Track Relinking.

**current_playlists**(**, limit: Optional[int] = 20, offset: Optional[int] = 0*) → Awaitable[T_co]
Get a list of the playlists owned or followed by the current Spotify user.

**Parameters**

- **limit** (Optional[str]) – The maximum number of playlists to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (Optional[str]) – he index of the first playlist to return. Default: 0 (the first object). Maximum offset: 100.000.

**current_user**() → Awaitable[T_co]
Get detailed profile information about the current user (including the current user's username).

**currently_playing**(**, market: Optional[str] = None*) → Awaitable[T_co]
Get the object currently being played on the user's Spotify account.

**Parameters market** (Optional[str]) – An ISO 3166-1 alpha-2 country code or the string from_token. Provide this parameter if you want to apply Track Relinking.

**delete_saved_albums**(*ids: List[str]*) → Awaitable[T_co]
Remove one or more albums from the current user's 'Your Music' library.

**Parameters ids** (List[str]) – A list of the Spotify IDs.

**delete_saved_tracks**(*ids: List[str]*) → Awaitable[T_co]
Remove one or more tracks from the current user's 'Your Music' library.

**Parameters ids** (List[str]) – A list of the Spotify IDs.

**featured_playlists**(*locale=None, country=None, timestamp=None, limit: Optional[int] = 20, offset: Optional[int] = 0*)
Get a list of Spotify featured playlists (shown, for example, on a Spotify player's 'Browse' tab).

**Parameters**

- **locale** (*LOCALE_TP*) – LOCALE
- **country** (*COUNTRY_TP*) – COUNTRY
- **timestamp** (*TIMESTAMP_TP*) – TIMESTAMP
- **limit** (*Optional[int]*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.

- **`offset`** (*`Optional[int]`*) – The index of the first item to return. Default: 0

**`follow_artist_or_user`** (*type_: str, ids: List[str]*) → Awaitable[T_co]
Add the current user as a follower of one or more artists or other Spotify users.

**Parameters**

- **`type`** (`str`) – either artist or user.
- **`ids`** (List[`str`]) – A list of the artist or the user Spotify IDs.

**`follow_playlist`** (*playlist_id: str*, ***, *public: Optional[bool] = True*) → Awaitable[T_co]
Add the current user as a follower of a playlist.

**Parameters**

- **`playlist_id`** (`str`) – The Spotify ID of the playlist. Any playlist can be followed, regardless of its public/private status, as long as you know its playlist ID.
- **`public`** (Optional[`bool`]) – Defaults to true. If true the playlist will be included in user's public playlists, if false it will remain private.

**`followed_artists`** (***, *limit: Optional[int] = 20*, *after: Optional[str] = None*) → Awaitable[T_co]
Get the current user's followed artists.

**Parameters**

- **`limit`** (Optional[`int`]) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **`after`** (Optional[`str`]) – The last artist ID retrieved from the previous request.

**`following_artists_or_users`** (*ids*, ***, *type_='artist'*) → Awaitable[T_co]
Check to see if the current user is following one or more artists or other Spotify users.

**Parameters**

- **`ids`** (List[`str`]) – A comma-separated list of the artist or the user Spotify IDs to check. A maximum of 50 IDs can be sent in one request.
- **`type`** (Optional[`str`]) – The ID type: either "artist" or "user". Default: "artist"

**`following_playlists`** (*playlist_id: str, ids: List[str]*) → Awaitable[T_co]
Check to see if one or more Spotify users are following a specified playlist.

**Parameters**

- **`playlist_id`** (`str`) – The Spotify ID of the playlist.
- **`ids`** (List[`str`]) – A list of the artist or the user Spotify IDs. A maximum of five IDs are allowed.

**`get_bearer_info`** (*client_id: Optional[str] = None*, *client_secret: Optional[str] = None*, *session: Optional[aiohttp.client.ClientSession] = None*)
Get the application bearer token from client_id and client_secret.

**Raises** *`SpotifyException`* – This will be raised when either *client_id* or *client_secret* is *None*

**`get_episode`** (*spotify_id: str*, *market: Optional[str] = 'US'*) → Awaitable[T_co]
Get Spotify catalog information for a single episode identified by its unique Spotify ID.

**Parameters**

- **`spotify_id`** (*`str`*) – The spotify_id to for the show.
- **`market`** (*`Optional[str]`*) – An ISO 3166-1 alpha-2 country code.

**get_multiple_episodes** (*ids: List[str], market: Optional[str] = 'US'*) → Awaitable[T_co]
Get Spotify catalog information for several episodes based on their Spotify IDs.

**Parameters**

- **ids** (List[`str`]) – A list of the Spotify IDs.
- **market** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**get_multiple_shows** (*ids: List[str], market: Optional[str] = 'US'*) → Awaitable[T_co]
Get Spotify catalog information for several shows based on their Spotify IDs.

**Parameters**

- **ids** (List[`str`]) – A list of the Spotify IDs.
- **market** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**get_playlist** (*playlist_id: str*, ***, *fields: Optional[str] = None*, *market: Optional[str] = None*) → Awaitable[T_co]
Get a playlist owned by a Spotify user.

**Parameters**

- **playlist_id** (`str`) – The Spotify ID for the playlist.
- **fields** (Optional[`str`]) – Filters for the query: a comma-separated list of the fields to return. If omitted, all fields are returned. For example, to get just the total number of tracks and the request limit: *fields=total,limit*

  A dot separator can be used to specify non-reoccurring fields, while parentheses can be used to specify reoccurring fields within objects. For example, to get just the added date and user ID of the adder: *fields=items(added_at,added_by.id)*

  Use multiple parentheses to drill down into nested objects, for example: *fields=items(track(name,href,album(name,href)))*

  Fields can be excluded by prefixing them with an exclamation mark, for example: *fields=items.track.album(!external_urls,images)*
- **market** (Optional[`str`]) – An ISO 3166-1 alpha-2 country code or the string "from_token". Provide this parameter if you want to apply Track Relinking.

**get_playlist_cover_image** (*playlist_id: str*) → Awaitable[T_co]
Get the current image associated with a specific playlist.

**Parameters** **playlist_id** (`str`) – The Spotify ID for the playlist.

**get_playlist_tracks** (*playlist_id: str*, ***, *fields: Optional[str] = None*, *market: Optional[str] = None*, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*) → Awaitable[T_co]
Get full details of the tracks of a playlist owned by a Spotify user.

**Parameters**

- **playlist_id** (`str`) – The Spotify ID for the playlist.
- **fields** (Optional[`str`]) – Filters for the query: a comma-separated list of the fields to return. If omitted, all fields are returned. For example, to get just the total number of tracks and the request limit: *fields=total,limit*

  A dot separator can be used to specify non-reoccurring fields, while parentheses can be used to specify reoccurring fields within objects. For example, to get just the added date and user ID of the adder: *fields=items(added_at,added_by.id)*

Use multiple parentheses to drill down into nested objects, for example: *fields=items(track(name,href,album(name,href)))*

Fields can be excluded by prefixing them with an exclamation mark, for example: *fields=items.track.album(!external_urls,images)*

- **`limit`** (Optional[`str`]) – The maximum number of tracks to return. Default: 100. Minimum: 1. Maximum: 100.
- **`offset`** (Optional[`str`]) – The index of the first track to return. Default: 0 (the first object).
- **`market`** (Optional[`str`]) – An ISO 3166-1 alpha-2 country code or the string "from_token". Provide this parameter if you want to apply Track Relinking.

**`get_playlists`** (*user_id: str*, ***, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*) → Awaitable[T_co]

Get a list of the playlists owned or followed by a Spotify user.

**Parameters**

- **`user_id`** (`str`) – The user's Spotify user ID.
- **`limit`** (Optional[`str`]) – The maximum number of playlists to return. Default: 20. Minimum: 1. Maximum: 50.
- **`offset`** (Optional[`str`]) – he index of the first playlist to return. Default: 0 (the first object). Maximum offset: 100.000.

**`get_saved_shows`** (*limit: int = 20*, *offset: int = 0*) → Awaitable[T_co]

Get a list of shows saved in the current Spotify user's library. Optional parameters can be used to limit the number of shows returned.

**Parameters**

- **`limit`** (*`Optional[int]`*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **`offset`** (*`Optiona[int]`*) – The offset of which Spotify should start yielding from.

**`get_show`** (*spotify_id: str*, *market: Optional[str] = 'US'*) → Awaitable[T_co]

Get Spotify catalog information for a single show identified by its unique Spotify ID.

**Parameters**

- **`spotify_id`** (*`str`*) – The spotify_id to for the show.
- **`market`** (*`Optional[str]`*) – An ISO 3166-1 alpha-2 country code.

**`get_shows_episodes`** (*spotify_id: str*, *market: Optional[str] = 'US'*, *limit: int = 20*, *offset: int = 0*) → Awaitable[T_co]

Get Spotify catalog information about an show's episodes. Optional parameters can be used to limit the number of episodes returned.

**Parameters**

- **`spotify_id`** (*`str`*) – The spotify_id to for the show.
- **`market`** (*`Optional[str]`*) – An ISO 3166-1 alpha-2 country code.
- **`limit`** (*`Optional[int]`*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **`offset`** (*`Optiona[int]`*) – The offset of which Spotify should start yielding from.

**is_saved_album**(*ids: List[str]*) → Awaitable[T_co]
Check if one or more albums is already saved in the current Spotify user's 'Your Music' library.

**Parameters** **ids** (List[str]) – A list of the Spotify IDs.

**is_saved_track**(*ids: List[str]*) → Awaitable[T_co]
Check if one or more tracks is already saved in the current Spotify user's 'Your Music' library.

**Parameters** **ids** (List[str]) – A list of the Spotify IDs.

**new_releases**(**, country=None, limit: Optional[int] = 20, offset: Optional[int] = 0*) → Awaitable[T_co]
Get a list of new album releases featured in Spotify (shown, for example, on a Spotify player's "Browse" tab).

**Parameters**

- **limit** (*Optional[int]*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optional[int]*) – The index of the first item to return. Default: 0
- **country** (*COUNTRY_TP*) – COUNTRY

**pause_playback**(**, device_id: Optional[str] = None*) → Awaitable[T_co]
Pause playback on the user's account.

**Parameters** **device_id** (Optional[str]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**play_playback**(*context_uri: Union[str, Sequence[str]], *, offset: Union[str, int, None] = None, device_id: Optional[str] = None, position_ms: Optional[int] = 0*) → Awaitable[T_co]
Start a new context or resume current playback on the user's active device.

**Note:** In order to resume playback set the context_uri to None.

**Parameters**

- **context_uri** (Union[str, Sequence[str]]) – The context to play, if it is a string then it must be a uri of a album, artist or playlist.

  Otherwise a sequece of strings can be passed in and they must all be track URIs
- **offset** (Optional[Union[str, int]]) – The offset of which to start from, must either be an integer or a track URI.
- **device_id** (Optional[str]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.
- **position_ms** (Optional[int]) – indicates from what position to start playback. Must be a positive number. Passing in a position that is greater than the length of the track will cause the player to start playing the next song.

**playback_queue**(**, uri: str, device_id: Optional[str] = None*) → Awaitable[T_co]
Add an item to the end of the user's current playback queue.

**Parameters**

- **uri** (str) – The uri of the item to add to the queue. Must be a track or an episode uri.
- **device_id** (str) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**recently_played**(**, limit: Optional[int] = 20, before: Optional[str] = None, after: Optional[str] = None*) → Awaitable[T_co]

Get tracks from the current user's recently played tracks.

Returns the most recent 50 tracks played by a user. Note that a track currently playing will not be visible in play history until it has completed. A track must be played for more than 30 seconds to be included in play history.

Any tracks listened to while the user had "Private Session" enabled in their client will not be returned in the list of recently played tracks.

The endpoint uses a bidirectional cursor for paging. Follow the next field with the before parameter to move back in time, or use the after parameter to move forward in time. If you supply no before or after parameter, the endpoint will return the most recently played songs, and the next link will page back in time.

**Parameters**

- **limit** (Optional[`int`]) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **after** (Optional[`str`]) – A Unix timestamp in milliseconds. Returns all items after (but not including) this cursor position. If after is specified, before must not be specified.
- **before** (Optional[`str`]) – A Unix timestamp in milliseconds. Returns all items before (but not including) this cursor position. If before is specified, after must not be specified.

**recommendations**(*seed_artists, seed_genres, seed_tracks, *, limit: Optional[int] = 20, market=None, **filters*)

Get Recommendations Based on Seeds.

**Parameters**

- **seed_artists** (*`str`*) – A comma separated list of Spotify IDs for seed artists. Up to 5 seed values may be provided.
- **seed_genres** (*`str`*) – A comma separated list of any genres in the set of available genre seeds. Up to 5 seed values may be provided.
- **seed_tracks** (*`str`*) – A comma separated list of Spotify IDs for a seed track. Up to 5 seed values may be provided.
- **limit** (*`Optional[int]`*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **market** (*`Optional[str]`*) – An ISO 3166-1 alpha-2 country code.
- **max_*** (*`Optional[Keyword arguments]`*) – For each tunable track attribute, a hard ceiling on the selected track attribute's value can be provided.
- **min_*** (*`Optional[Keyword arguments]`*) – For each tunable track attribute, a hard floor on the selected track attribute's value can be provided.
- **target_*** (*`Optional[Keyword arguments]`*) – For each of the tunable track attributes (below) a target value may be provided.

**remove_playlist_tracks**(*playlist_id: str, tracks: Sequence[Union[str, Dict[str, Any]]], *, snapshot_id: str = None*) → Awaitable[T_co]

Remove one or more tracks from a user's playlist.

**Parameters**

- **playlist_id** (*`str`*) – The id of the playlist to target

- **tracks** (*Sequence[Union[str, Dict[str, Union[str, int]]]]*) – Either a sequence of track URIs to remove a specific occurence of a track or for targeted removal pass in a dict that looks like *{'uri': URI, 'position': POSITIONS}* where *URI* is track URI and *POSITIONS* is an list of integers
- **snapshot_id** (*Optional[str]*) – The snapshot to target.

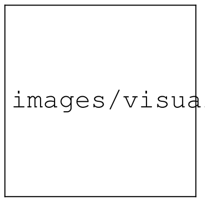
**remove_saved_shows** (*ids: List[str], market: Optional[str] = 'US'*) → Awaitable[T_co]
Delete one or more shows from current Spotify user's library.

**Parameters**

- **ids** (List[str]) – A list of the Spotify IDs.
- **market** (*Optional[str]*) – An ISO 3166-1 alpha-2 country code.

**reorder_playlists_tracks** (*playlist_id: str, range_start: int, range_length: int, insert_before: int, *, snapshot_id: Optional[str] = None*) → Awaitable[T_co]
Reorder a track or a group of tracks in a playlist.

Visualization of how reordering tracks works


images/visualization-reordering-tracks.png

**Note:** When reordering tracks, the timestamp indicating when they were added and the user who added them will be kept untouched. In addition, the users following the playlists won't be notified about changes in the playlists when the tracks are reordered.

**Parameters**

- **playlist_id** (str) – The Spotify ID for the playlist.
- **range_start** (int) – The position of the first track to be reordered.
- **range_length** (int) – The amount of tracks to be reordered. Defaults to 1 if not set.

  The range of tracks to be reordered begins from the range_start position, and includes the range_length subsequent tracks.
- **insert_before** (int) – The position where the tracks should be inserted.

  To reorder the tracks to the end of the playlist, simply set insert_before to the position after the last track.
- **snapshot_id** (Optional[str]) – The playlist's snapshot ID against which you want to make the changes.

**repeat_playback** (*state: str, *, device_id: Optional[str] = None*) → Awaitable[T_co]
Set the repeat mode for the user's playback. Options are repeat-track, repeat-context, and off.

**Parameters**

- **state** (str) –

  **"track", "context" or "off".**

  – track will repeat the current track.

    – context will repeat the current context.
    – off will turn repeat off.

  - **device_id** (*Optional[str]*) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**replace_playlist_tracks**(*playlist_id: str, tracks: Sequence[str]*) → Awaitable[T_co]
Replace all the tracks in a playlist, overwriting its existing tracks.

**Note:** This powerful request can be useful for replacing tracks, re-ordering existing tracks, or clearing the playlist.

**Parameters**

- **playlist_id** (str) – The Spotify ID for the playlist.
- **tracks** (Sequence[str]) – A list of tracks to replace with.

**request**(*route*, ***kwargs*)
Make a request to the spotify API with the current bearer credentials.

**Parameters**

- **route** (*Tuple[str, str]*) – A tuple of the method and url gained from *route()*.
- ****kwargs** (*Any*) – keyword arguments to pass into `aiohttp.ClientSession.request`

**static route**(*method: str*, *path: str*, ***, *base: str = 'https://api.spotify.com/v1'*, ***kwargs*) → Tuple[str, str]
Used for constructing URLs for API endpoints.

**Parameters**

- **method** (*str*) – The HTTP/REST method used.
- **path** (*str*) – A path to be formatted.
- **kwargs** (*Any*) – The arguments used to format the path.

**Returns** **route** – A tuple of the method and formatted url path to use.

**Return type** Tuple[str, str]

**save_albums**(*ids: List[str]*) → Awaitable[T_co]
Save one or more albums to the current user's 'Your Music' library.

**Parameters** **ids** (List[str]) – A list of the Spotify IDs.

**save_shows**(*ids: List[str]*) → Awaitable[T_co]
Save one or more shows to current Spotify user's library.

**Parameters** **ids** (List[str]) – A list of the Spotify IDs.

**save_tracks**(*ids: List[str]*) → Awaitable[T_co]
Save one or more tracks to the current user's 'Your Music' library.

**Parameters** **ids** (List[str]) – A list of the Spotify IDs.

**saved_albums**(***, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*, *market: Optional[str] = None*) → Awaitable[T_co]
Get a list of the albums saved in the current Spotify user's 'Your Music' library.

**Parameters**

- **limit** (Optional[str]) – The maximum number of objects to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (Optional[str]) – The index of the first object to return. Default: 0 (i.e., the first object). Use with limit to get the next set of objects.
- **market** (Optional[str]) – An ISO 3166-1 alpha-2 country code or the string from_token. Provide this parameter if you want to apply Track Relinking.

**saved_tracks** (**, limit: Optional[int] = 20, offset: Optional[int] = 0, market: Optional[str] = None*) → Awaitable[T_co]

Get a list of the songs saved in the current Spotify user's 'Your Music' library.

**Parameters**

- **limit** (Optional[str]) – The maximum number of objects to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (Optional[str]) – The index of the first object to return. Default: 0 (i.e., the first object). Use with limit to get the next set of objects.
- **market** (Optional[str]) – An ISO 3166-1 alpha-2 country code or the string from_token. Provide this parameter if you want to apply Track Relinking.

**search** (*q: str, query_type: str = 'track, playlist, artist, album', market: str = 'US', limit: int = 20, offset: int = 0, include_external: Optional[str] = None*) → Awaitable[T_co]

Get Spotify Catalog information about artists, albums, tracks or playlists that match a keyword string.

**Parameters**

- **q** (str) – Search query keywords and optional field filters and operators. e.g. *roadhouse blues.*
- **query_type** (Optional[str]) – A comma-separated list of item types to search across. (default: "track,playlist,artist,album") Valid types are: album, artist, playlist, and track. Search results include hits from all the specified item types.
- **market** (Optional[str]) – An ISO 3166-1 alpha-2 country code or the string "from_token". (default: "US") If a country code is specified, only artists, albums, and tracks with content that is playable in that market is returned.

  **Note:**

  – Playlist results are not affected by the market parameter.

  – **If market is set to "from_token", and a valid access token is specified in the request header, only** content playable in the country associated with the user account, is returned.

  – **Users can view the country that is associated with their account in the account settings. A user must** grant access to the user-read-private scope prior to when the access token is issued.

- **limit** (Optional[int]) – Maximum number of results to return. (Default: 20, Minimum: 1, Maximum: 50)

  **Note:** The limit is applied within each type, not on the total response. For example, if the limit value is 3 and the type is artist,album, the response contains 3 artists and 3 albums.

- **offset** (Optional[`int`]) – The index of the first result to return. Default: 0 (the first result). Maximum offset (including limit): 10,000. Use with limit to get the next page of search results.
- **include_external** (Optional[`str`]) – Possible values: *audio* If *include_external=audio* is specified the response will include any relevant audio content that is hosted externally. By default external content is filtered out from responses.

**seek_playback** (*position_ms: int*, **, *device_id: Optional[str] = None*) → Awaitable[T_co]

Seeks to the given position in the user's currently playing track.

**Parameters**

- **position_ms** (`int`) – The position in milliseconds to seek to. Must be a positive number. Passing in a position that is greater than the length of the track will cause the player to start playing the next song.
- **device_id** (Optional[`str`]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**set_playback_volume** (*volume: int*, **, *device_id: Optional[str] = None*) → Awaitable[T_co]

Set the volume for the user's current playback device.

**Parameters**

- **volume** (`int`) – The volume to set. Must be a value from 0 to 100 inclusive.
- **device_id** (Optional[`str`]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**shuffle_playback** (*state: bool*, **, *device_id: Optional[str] = None*) → Awaitable[T_co]

Toggle shuffle on or off for user's playback.

**Parameters**

- **state** (`bool`) – True : Shuffle user's playback False : Do not shuffle user's playback.
- **device_id** (Optional[`str`]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**skip_next** (**, *device_id: Optional[str] = None*) → Awaitable[T_co]

Skips to next track in the user's queue.

**Parameters** **device_id** (Optional[`str`]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**skip_previous** (**, *device_id: Optional[str] = None*) → Awaitable[T_co]

Skips to previous track in the user's queue.

**Parameters** **device_id** (Optional[`str`]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**top_artists_or_tracks** (*type_: str*, **, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*, *time_range: Optional[str] = None*) → Awaitable[T_co]

Get the current user's top artists or tracks based on calculated affinity.

Affinity is a measure of the expected preference a user has for a particular track or artist. It is based on user behavior, including play history, but does not include actions made while in incognito mode. Light or infrequent users of Spotify may not have sufficient play history to generate a full affinity data set.

As a user's behavior is likely to shift over time, this preference data is available over three time spans.

For each time range, the top 50 tracks and artists are available for each user. In the future, it is likely that this restriction will be relaxed. This data is typically updated once each day for each user.

**Parameters**

- **`type`** (*`:class;`str``*) – The type of entity to return. Valid values: "artists" or "tracks".
- **`limit`** (Optional[`int`]) – The number of entities to return. Default: 20. Minimum: 1. Maximum: 50. For example: limit=2
- **`offset`** (Optional[`int`]) – The index of the first entity to return. Default: 0 (i.e., the first track). Use with limit to get the next set of entities.
- **`time_range`** (Optional[`str`]) – Over what time frame the affinities are computed. Valid values: - "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). Default: medium_term.

**`track`**(*track_id: str*, *market: Optional[str] = None*) → Awaitable[T_co]

Get Spotify catalog information for a single track identified by its unique Spotify ID.

**Parameters**

- **`track_id`** (`str`) – The Spotify ID for the track.
- **`market`** (Optional[`str`]) – An ISO 3166-1 alpha-2 country code or the string "from_token". Provide this parameter if you want to apply Track Relinking.

**`track_audio_analysis`**(*track_id: str*) → Awaitable[T_co]

Get a detailed audio analysis for a single track identified by its unique Spotify ID.

The Audio Analysis endpoint provides low-level audio analysis for all of the tracks in the Spotify catalog. The Audio Analysis describes the track's structure and musical content, including rhythm, pitch, and timbre. All information is precise to the audio sample.

Many elements of analysis include confidence values, a floating-point number ranging from 0.0 to 1.0. Confidence indicates the reliability of its corresponding attribute. Elements carrying a small confidence value should be considered speculative. There may not be sufficient data in the audio to compute the attribute with high certainty.

**Parameters** **`track_id`** (`str`) – The Spotify ID for the track.

**`track_audio_features`**(*track_id: str*) → Awaitable[T_co]

Get audio feature information for a single track identified by its unique Spotify ID.

**Parameters** **`track_id`** (`str`) – The Spotify ID for the track.

**`tracks`**(*track_ids: List[str], market: Optional[str] = None*) → Awaitable[T_co]

Get Spotify catalog information for multiple tracks based on their Spotify IDs.

**Parameters**

- **`track_ids`** (List[`str`]) – A comma-separated list of the Spotify IDs for the tracks. Maximum: 50 IDs.
- **`market`** (Optional[`str`]) – An ISO 3166-1 alpha-2 country code or the string "from_token". Provide this parameter if you want to apply Track Relinking.

**`transfer_player`**(*device_id: str*, ***, *play: Optional[bool] = False*) → Awaitable[T_co]

Transfer playback to a new device and determine if it should start playing.

**Parameters**

- **`device_id`** (`str`) – A Spotify Device ID

- **`play`** (Optional[`bool`]) – True: ensure playback happens on new device. False or not provided: keep the current playback state.

**`unfollow_artists_or_users`**(*type_: str, ids: List[str]*) → Awaitable[T_co]
Remove the current user as a follower of one or more artists or other Spotify users.

**Parameters**

- **`type`** (`str`) – either artist or user.
- **`ids`** (List[`str`]) – A list of the artist or the user Spotify IDs.

**`unfollow_playlist`**(*playlist_id: str*) → Awaitable[T_co]
Remove the current user as a follower of a playlist.

**Parameters** **`playlist_id`** (`str`) – The Spotify ID of the playlist that is to be no longer followed.

**`upload_playlist_cover_image`**(*playlist_id: str, file: BinaryIO*) → Awaitable[T_co]
Replace the image used to represent a specific playlist.

**Parameters**

- **`playlist_id`** (`str`) – The Spotify ID for the playlist.
- **`file`** (*`File-like object`*) – An file-like object that supports reading the contents that are being read should be `bytes`

**`user`**(*user_id: str*) → Awaitable[T_co]
Get public profile information about a Spotify user.

**Parameters** **`user_id`** (class:*str*) – The user's Spotify user ID.

## Models

### Album

**class** `spotify.`**`Album`**(*client, data*)
A Spotify Album.

**`artists`**
The artists for the album.

**Type** List[*Artist*]

**`id`**
The ID of the album.

**Type** str

**`name`**
The name of the album.

**Type** str

**`href`**
The HTTP API URL for the album.

**Type** str

**`uri`**
The URI for the album.

**Type** str

**`album_group`**
ossible values are "album", "single", "compilation", "appears_on". Compare to album_type this field represents relationship between the artist and the album.

**Type** str

**`album_type`**
The type of the album: one of "album" , "single" , or "compilation".

**Type** str

**`release_date`**
The date the album was first released.

**Type** str

**`release_date_precision`**
The precision with which release_date value is known: year, month or day.

**Type** str

**`genres`**
A list of the genres used to classify the album.

**Type** List[str]

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

**Type** str

**`popularity`**
The popularity of the album. The value will be between 0 and 100, with 100 being the most popular.

**Type** int

**`copyrights`**
The copyright statements of the album.

**Type** List[Dict]

**`markets`**
The markets in which the album is available: ISO 3166-1 alpha-2 country codes.

**Type** List[str]

**`get_all_tracks`**(**, market: Optional[str] = 'US'*) → List[spotify.models.track.Track]
loads all of the albums tracks, depending on how many the album has this may be a long operation.

**Parameters** **`market`** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code. Provide this parameter if you want to apply Track Relinking.

**Returns** **tracks** – The tracks of the artist.

**Return type** List[`spotify.Track`]

**`get_tracks`**(**, limit: Optional[int] = 20, offset: Optional[int] = 0*) → List[spotify.models.track.Track]
get the albums tracks from spotify.

**Parameters**

- **`limit`** (`Optional[int]`) – The limit on how many tracks to retrieve for this album (default is 20).
- **`offset`** (`Optional[int]`) – The offset from where the api should start from in the tracks.

**Returns** **tracks** – The tracks of the artist.

**Return type** List[Track]

## Artist

**class** `spotify.`**`Artist`**(*client*, *data*)

A Spotify Artist.

**`id`**

The Spotify ID of the artist.

**Type** str

**`uri`**

The URI of the artist.

**Type** str

**`url`**

The open.spotify URL.

**Type** str

**`href`**

A link to the Web API endpoint providing full details of the artist.

**Type** str

**`name`**

The name of the artist.

**Type** str

**`genres`**

A list of the genres the artist is associated with. For example: "Prog Rock" , "Post-Grunge". (If not yet classified, the array is empty.)

**Type** List[str]

**`followers`**

The total number of followers.

**Type** Optional[int]

**`popularity`**

The popularity of the artist. The value will be between 0 and 100, with 100 being the most popular. The artist's popularity is calculated from the popularity of all the artist's tracks.

**Type** int

**`images`**

Images of the artist in various sizes, widest first.

**Type** List[*Image*]

**`get_albums`**(*, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*, *include_groups=None*, *market: Optional[str] = None*) → List[spotify.Album]

Get the albums of a Spotify artist.

**Parameters**

- **`limit`** (*`Optional[int]`*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.

- **`offset`** (`Optiona[int]`) – The offset of which Spotify should start yielding from.
- **`include_groups`** (`INCLUDE_GROUPS_TP`) – INCLUDE_GROUPS
- **`market`** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**Returns** **albums** – The albums of the artist.

**Return type** List[*Album*]

**`get_all_albums`**(*, *market='US'*) → List[spotify.Album]
loads all of the artists albums, depending on how many the artist has this may be a long operation.

**Parameters** **`market`** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**Returns** **albums** – The albums of the artist.

**Return type** List[*Album*]

**`related_artists`**() → List[spotify.models.artist.Artist]
Get Spotify catalog information about artists similar to a given artist.

Similarity is based on analysis of the Spotify community's listening history.

**Returns** **artists** – The artists deemed similar.

**Return type** List[*Artist*]

**`top_tracks`**(*country: str = 'US'*) → List[spotify.Track]
Get Spotify catalog information about an artist's top tracks by country.

**Parameters** **`country`** (`str`) – The country to search for, it defaults to 'US'.

**Returns** **tracks** – The artists top tracks.

**Return type** List[Track]

**`total_albums`**(*, *market: str = None*) → int
get the total amout of tracks in the album.

**Parameters** **`market`** (`Optional[str]`) – An ISO 3166-1 alpha-2 country code.

**Returns** **total** – The total amount of albums.

**Return type** int

## User

**class** `spotify.`**`User`**(*client: spotify.Client*, *data: dict*, ***kwargs*)
A Spotify User.

**`id`**
The Spotify user ID for the user.

**Type** `str`

**`uri`**
The Spotify URI for the user.

**Type** `str`

**`url`**
The open.spotify URL.

**Type** `str`

**href**
A link to the Web API endpoint for this user.

**Type** `str`

**display_name**
The name displayed on the user's profile. *None* if not available.

**Type** `str`

**followers**
The total number of followers.

**Type** `int`

**images**
The user's profile image.

**Type** List[*Image*]

**email**
The user's email address, as entered by the user when creating their account.

**Type** `str`

**country**
The country of the user, as set in the user's account profile. An ISO 3166-1 alpha-2 country code.

**Type** `str`

**birthdate**
The user's date-of-birth.

**Type** `str`

**product**
The user's Spotify subscription level: "premium", "free", etc. (The subscription level "open" can be considered the same as "free".)

**Type** `str`

**add_tracks**(*playlist: Union[str, spotify.models.playlist.Playlist], *tracks*) → str
Add one or more tracks to a user's playlist.

**Parameters**

- **playlist** (Union[`str`, Playlist]) – The playlist to modify
- **tracks** (Sequence[Union[`str`, Track]]) – Tracks to add to the playlist

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** `str`

**create_playlist**(*name*, *, *public=True*, *collaborative=False*, *description=None*)
Create a playlist for a Spotify user.

**Parameters**

- **name** (`str`) – The name of the playlist.
- **public** (*Optional[bool]*) – The public/private status of the playlist. *True* for public, *False* for private.
- **collaborative** (*Optional[bool]*) – If *True*, the playlist will become collaborative and other users will be able to modify the playlist.

- **description** (Optional[str]) – The playlist description

**Returns** **playlist** – The playlist that was created.

**Return type** *Playlist*

**currently_playing**() → Dict[str, Union[spotify.models.track.Track, spotify.models.common.Context, str]]

Get the users currently playing track.

**Returns** **context, track** – A tuple of the context and track.

**Return type** Dict[str, Union[Track, *Context*, str]]

**edit_playlist**(*playlist*, ***, *name=None*, *public=None*, *collaborative=None*, *description=None*)

Change a playlist's name and public/private, collaborative state and description.

**Parameters**

- **playlist** (Union[str, Playlist]) – The playlist to modify
- **name** (Optional[str]) – The new name of the playlist.
- **public** (*Optional[bool]*) – The public/private status of the playlist. *True* for public, *False* for private.
- **collaborative** (*Optional[bool]*) – If *True*, the playlist will become collaborative and other users will be able to modify the playlist.
- **description** (Optional[str]) – The new playlist description

**follow_playlist**(*playlist: Union[str, spotify.models.playlist.Playlist]*, ***, *public: bool = True*) → None

follow a playlist

**Parameters**

- **playlist** (Union[str, Playlist]) – The playlist to modify
- **public** (*Optional[bool]*) – The public/private status of the playlist. *True* for public, *False* for private.

**classmethod from_code**(*client: spotify.Client*, *code: str*, ***, *redirect_uri: str*)

Create a *User* object from an authorization code.

**Parameters**

- **client** (*spotify.Client*) – The spotify client to associate the user with.
- **code** (str) – The authorization code to use to further authenticate the user.
- **redirect_uri** (str) – The rediriect URI to use in tandem with the authorization code.

**classmethod from_refresh_token**(*client: spotify.Client*, *refresh_token: str*)

Create a *User* object from a refresh token. It will poll the spotify API for a new access token and use that to initialize the spotify user.

**Parameters**

- **client** (*spotify.Client*) – The spotify client to associate the user with.
- **refresh_token** (*str*) – Used to acquire token.

**classmethod from_token**(*client: spotify.Client, token: Optional[str], refresh_token: Optional[str] = None*)

Create a *User* object from an access token.

**Parameters**

- **client** (*spotify.Client*) – The spotify client to associate the user with.
- **token** (str) – The access token to use for http requests.
- **refresh_token** (str) – Used to acquire new token when it expires.

**get_all_playlists**() → List[spotify.models.playlist.Playlist]

Get all of the users playlists from spotify.

**Returns** **playlists** – A list of the users playlists.

**Return type** List[*Playlist*]

**get_devices**() → List[spotify.models.common.Device]

Get information about the users avaliable devices.

**Returns** **devices** – The devices the user has available.

**Return type** List[*Device*]

**get_player**() → spotify.models.player.Player

Get information about the users current playback.

**Returns** **player** – A player object representing the current playback.

**Return type** *Player*

**get_playlists**(*, *limit: int = 20*, *offset: int = 0*) → List[spotify.models.playlist.Playlist]

get the users playlists from spotify.

**Parameters**

- **limit** (*Optional[int]*) – The limit on how many playlists to retrieve for this user (default is 20).
- **offset** (*Optional[int]*) – The offset from where the api should start from in the playlists.

**Returns** **playlists** – A list of the users playlists.

**Return type** List[*Playlist*]

**get_podcasts**(*, *limit: int = 20*, *offset: int = 0*) → List[spotify.models.podcast.Podcast]

Get the current user's saved podcasts, shows.

**Parameters**

- **limit** (*Optional[int]*) – The number of entities to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optional[int]*) – The index of the first entity to return. Default: 0

**Returns** **podcasts** – The saved podcasts of the user.

**Return type** List[Podcast]

**recently_played**(*, *limit: int = 20*, *before: Optional[str] = None*, *after: Optional[str] = None*) → List[Dict[str, Union[spotify.models.track.Track, spotify.models.common.Context, str]]]

Get tracks from the current users recently played tracks.

**Returns** **playlist_history** – A list of playlist history object. Each object is a dict with a timestamp, track and context field.

**Return type** List[Dict[str, Union[Track, Context, str]]]

**remove_tracks**(*playlist*, **tracks*)
Remove one or more tracks from a user's playlist.

**Parameters**

- **playlist** (Union[str, Playlist]) – The playlist to modify
- **tracks** (Sequence[Union[str, Track]]) – Tracks to remove from the playlist

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** str

**reorder_tracks**(*playlist*, *start*, *insert_before*, *length=1*, ***, *snapshot_id=None*)
Reorder a track or a group of tracks in a playlist.

**Parameters**

- **playlist** (Union[str, Playlist]) – The playlist to modify
- **start** (*int*) – The position of the first track to be reordered.
- **insert_before** (*int*) – The position where the tracks should be inserted.
- **length** (*Optional[int]*) – The amount of tracks to be reordered. Defaults to 1 if not set.
- **snapshot_id** (str) – The playlist's snapshot ID against which you want to make the changes.

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** str

**replace_tracks**(*playlist*, **tracks*) → None
Replace all the tracks in a playlist, overwriting its existing tracks.

This powerful request can be useful for replacing tracks, re-ordering existing tracks, or clearing the playlist.

**Parameters**

- **playlist** (Union[str, PLaylist]) – The playlist to modify
- **tracks** (Sequence[Union[str, Track]]) – Tracks to place in the playlist

**top_artists**(***data*) → List[spotify.models.artist.Artist]
Get the current user's top artists based on calculated affinity.

**Parameters**

- **limit** (*Optional[int]*) – The number of entities to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optional[int]*) – The index of the first entity to return. Default: 0
- **time_range** (Optional[str]) – Over what time frame the affinities are computed. (long_term, short_term, medium_term)

**Returns** **tracks** – The top artists for the user.

**Return type** List[*Artist*]

**top_tracks**(***data*) → List[spotify.models.track.Track]
Get the current user's top tracks based on calculated affinity.

**Parameters**

- **limit** (*Optional[int]*) – The number of entities to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optional[int]*) – The index of the first entity to return. Default: 0
- **time_range** (Optional[str]) – Over what time frame the affinities are computed. (long_term, short_term, medium_term)

**Returns** **tracks** – The top tracks for the user.

**Return type** List[Track]

## Playlist

**class** spotify.**Playlist**(*client: spotify.Client, data: Union[dict, Playlist], *, http: Optional[spotify.http.HTTPClient] = None*)

A Spotify Playlist.

**collaborative**

Returns true if context is not search and the owner allows other users to modify the playlist. Otherwise returns false.

**Type** bool

**description**

The playlist description. Only returned for modified, verified playlists, otherwise null.

**Type** str

**url**

The open.spotify URL.

**Type** str

**followers**

The total amount of followers

**Type** int

**href**

A link to the Web API endpoint providing full details of the playlist.

**Type** str

**id**

The Spotify ID for the playlist.

**Type** str

**images**

Images for the playlist. The array may be empty or contain up to three images. The images are returned by size in descending order. If returned, the source URL for the image ( url ) is temporary and will expire in less than a day.

**Type** List[*spotify.Image*]

**name**

The name of the playlist.

**Type** str

**owner**

The user who owns the playlist

**Type** *spotify.User*

**public**

**The playlist's public/private status:** true the playlist is public, false the playlist is private, null the playlist status is not relevant.

**Type** :class'bool'

**snapshot_id**

The version identifier for the current playlist.

**Type** str

**tracks**

A tuple of *PlaylistTrack* objects or *None*.

**Type** Optional[Tuple[*PlaylistTrack*]]

**add_tracks**(**tracks*) → str

Add one or more tracks to a user's playlist.

**Parameters** **tracks** (Iterable[Union[str, Track]]) – Tracks to add to the playlist

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** str

**clear**()

Clear the playlists tracks.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**Warning:** This is a desctructive operation and can not be reversed!

**copy**() → spotify.models.playlist.Playlist

Return a shallow copy of the playlist object.

**Returns** **playlist** – The playlist object copy.

**Return type** *Playlist*

**extend**(*tracks: Union[Playlist, Iterable[Union[spotify.models.track.Track, str]]]*)

Extend a playlists tracks with that of another playlist or a list of Track/Track URIs.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**Parameters** **tracks** (*Union["Playlist", List[Union[Track, str]]]*) –

**Tracks to add to the playlist, acceptable values are:**

- A *spotify.Playlist* object
- A list of spotify.Track objects or Track URIs

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** str

**get_all_tracks**() → Tuple[spotify.models.track.PlaylistTrack, ...]
Get all playlist tracks from the playlist.

**Returns** **tracks** – The playlists tracks.

**Return type** Tuple[`PlaylistTrack`]

**get_tracks**(*, *limit: Optional[int] = 20*, *offset: Optional[int] = 0*) → Tuple[spotify.models.track.PlaylistTrack, ...]
Get a fraction of a playlists tracks.

**Parameters**

- **limit** (`Optional[int]`) – The limit on how many tracks to retrieve for this playlist (default is 20).
- **offset** (`Optional[int]`) – The offset from where the api should start from in the tracks.

**Returns** **tracks** – The tracks of the playlist.

**Return type** Tuple[*PlaylistTrack*]

**insert**(*index, obj: Union[spotify.models.track.PlaylistTrack, spotify.models.track.Track]*) → None
Insert an object before the index.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**pop**(*index: int = -1*) → spotify.models.track.PlaylistTrack
Remove and return the track at the specified index.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**Returns** **playlist_track** – The track that was removed.

**Return type** `PlaylistTrack`

**Raises** `IndexError` – If there are no tracks or the index is out of range.

**remove**(*value: Union[spotify.models.track.PlaylistTrack, spotify.models.track.Track]*) → None
Remove the first occurence of the value.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**Raises** `ValueError` – If the value is not present.

**remove_tracks**(**tracks*)
Remove one or more tracks from a user's playlist.

**Parameters** **tracks** (Iterable[Union[`str`, `Track`]]) – Tracks to remove from the playlist

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** `str`

**reorder_tracks**(*start: int*, *insert_before: int*, *length: int = 1*, *, *snapshot_id: Optional[str] = None*) → str
Reorder a track or a group of tracks in a playlist.

**Parameters**

- **`start`** (*`int`*) – The position of the first track to be reordered.
- **`insert_before`** (*`int`*) – The position where the tracks should be inserted.
- **`length`** (*`Optional[int]`*) – The amount of tracks to be reordered. Defaults to 1 if not set.
- **`snapshot_id`** (*`str`*) – The playlist's snapshot ID against which you want to make the changes.

**Returns** **snapshot_id** – The snapshot id of the playlist.

**Return type** str

**`replace_tracks`**(**tracks*) → None

Replace all the tracks in a playlist, overwriting its existing tracks.

This powerful request can be useful for replacing tracks, re-ordering existing tracks, or clearing the playlist.

**Parameters** **`tracks`** (Iterable[Union[`str`, `Track`]]) – Tracks to place in the playlist

**`reverse`**() → None

Reverse the playlist in place.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**`sort`**(**, key: Optional[Callable[[spotify.models.track.PlaylistTrack], bool]] = None, reverse: Optional[bool] = False*) → None

Stable sort the playlist in place.

**Note:** This method will mutate the current playlist object, and the spotify Playlist.

**`uri`**

str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

## Player

**class** `spotify.`**`Player`**(*client*, *user*, *data*)

A Spotify Users current playback.

**`device`**

The device that is currently active.

**Type** *`spotify.Device`*

**`repeat_state`**

"off", "track", "context"

**Type** `str`

**shuffle_state**
If shuffle is on or off.

**Type** bool

**is_playing**
If something is currently playing.

**Type** bool

**enqueue**(*uri: Union[spotify.models.base.URIBase, str], device: Union[spotify.models.common.Device, str, None] = None*)
Add an item to the end of the user's current playback queue.

**Parameters**

- **uri** (Union[spotify.URIBase, str]) – The uri of the item to add to the queue. Must be a track or an episode uri.
- **device_id** (Optional[Union[Device, str]]) – The id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**next**(**, device: Union[spotify.models.common.Device, str, None] = None*)
Skips to next track in the user's queue.

**Parameters** **device** (Optional[SomeDevice]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**pause**(**, device: Union[spotify.models.common.Device, str, None] = None*)
Pause playback on the user's account.

**Parameters** **device** (Optional[SomeDevice]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**play**(**uris, offset: Union[int, str, spotify.models.track.Track, None] = 0, device: Union[spotify.models.common.Device, str, None] = None*)
Start a new context or resume current playback on the user's active device.

The method treats a single argument as a Spotify context, such as a Artist, Album and playlist objects/URI. When called with multiple positional arguments they are interpreted as a array of Spotify Track objects/URIs.

**Parameters**

- ***uris** (*SomeURI*) – When a single argument is passed in that argument is treated as a context (except if it is a track or track uri). Valid contexts are: albums, artists, playlists. Album, Artist and Playlist objects are accepted too. Otherwise when multiple arguments are passed in they, A sequence of Spotify Tracks or Track URIs to play.
- **offset** (Optional[Offset]) – Indicates from where in the context playback should start. Only available when *context* corresponds to an album or playlist object, or when the *uris* parameter is used. when an integer offset is zero based and can't be negative.
- **device** (Optional[SomeDevice]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**previous**(**, device: Union[spotify.models.common.Device, str, None] = None*)
Skips to previous track in the user's queue.

Note that this will ALWAYS skip to the previous track, regardless of the current track's progress. Returning to the start of the current track should be performed using *seek()*

**Parameters** **`device`** (Optional[`SomeDevice`]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**`resume`** (**, device: Union[spotify.models.common.Device, str, None] = None*)

Resume playback on the user's account.

**Parameters** **`device`** (Optional[`SomeDevice`]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**`seek`** (*pos, *, device: Union[spotify.models.common.Device, str, None] = None*)

Seeks to the given position in the user's currently playing track.

**Parameters**

- **`pos`** (*`int`*) – The position in milliseconds to seek to. Must be a positive number. Passing in a position that is greater than the length of the track will cause the player to start playing the next song.
- **`device`** (Optional[`SomeDevice`]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**`set_repeat`** (*state, *, device: Union[spotify.models.common.Device, str, None] = None*)

Set the repeat mode for the user's playback.

**Parameters**

- **`state`** (*`str`*) – Options are repeat-track, repeat-context, and off
- **`device`** (Optional[`SomeDevice`]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**`set_volume`** (*volume: int, *, device: Union[spotify.models.common.Device, str, None] = None*)

Set the volume for the user's current playback device.

**Parameters**

- **`volume`** (*`int`*) – The volume to set. Must be a value from 0 to 100 inclusive.
- **`device`** (Optional[`SomeDevice`]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**`shuffle`** (*state: Optional[bool] = None, *, device: Union[spotify.models.common.Device, str, None] = None*)

shuffle on or off for user's playback.

**Parameters**

- **`state`** (*`Optional[bool]`*) – if *True* then Shuffle user's playback. else if *False* do not shuffle user's playback.
- **`device`** (Optional[`SomeDevice`]) – The Device object or id of the device this command is targeting. If not supplied, the user's currently active device is the target.

**`transfer`** (*device: Union[spotify.models.common.Device, str], ensure_playback: bool = False*)

Transfer playback to a new device and determine if it should start playing.

**Parameters**

- **`device`** (`SomeDevice`) – The device on which playback should be started/transferred.
- **`ensure_playback`** (*`bool`*) – if *True* ensure playback happens on new device. else keep the current playback state.

## Library

**class** spotify.**Library**(*client*, *user*)

A Spotify Users Library.

**user**

The user which this library object belongs to.

**Type** `Spotify.User`

**check_saved_shows**(**shows*) → List[bool]

Check if one or more shows is already saved in the current Spotify user's library.

**Parameters** **ids** (List[:class: *Show*]) – A list of the spotify.Show or unique spotify ids.

**Returns** **bools** – A list of bool results whether the show is saved or not.

**Return type** List[bool]

**contains_albums**(**albums*) → List[bool]

Check if one or more albums is already saved in the current Spotify user's 'Your Music' library.

**Parameters** **albums** (*Union[Album, str]*) – A sequence of artist objects or spotify IDs

**contains_tracks**(**tracks*) → List[bool]

Check if one or more tracks is already saved in the current Spotify user's 'Your Music' library.

**Parameters** **tracks** (*Union[Track, str]*) – A sequence of track objects or spotify IDs

**get_albums**(***, *limit=20*, *offset=0*) → List[spotify.models.album.Album]

Get a list of the albums saved in the current Spotify user's 'Your Music' library.

**Parameters**

- **limit** (*Optional[int]*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.
- **offset** (*Optional[int]*) – The index of the first item to return. Default: 0

**get_all_albums**() → List[spotify.models.album.Album]

Get a list of the albums saved in the current Spotify user's 'Your Music' library.

**Returns** **albums** – The albums.

**Return type** List[*Album*]

**get_all_podcasts**() → List[spotify.models.podcast.Podcast]

Get all of the users saved podcasts, shows from spotify.

**Returns** **playlists** – A list of the users podcasts.

**Return type** List[`Podcast`]

**get_all_tracks**() → List[spotify.models.track.Track]

Get a list of all the songs saved in the current Spotify user's 'Your Music' library.

**Returns** **tracks** – The tracks of the artist.

**Return type** List[`Track`]

**get_tracks**(***, *limit=20*, *offset=0*) → List[spotify.models.track.Track]

Get a list of the songs saved in the current Spotify user's 'Your Music' library.

**Parameters**

- **limit** (*Optional[int]*) – The maximum number of items to return. Default: 20. Minimum: 1. Maximum: 50.

- **offset** (*Optional[int]*) – The index of the first item to return. Default: 0

**remove_albums** (**albums*)

Remove one or more albums from the current user's 'Your Music' library.

**Parameters albums** (*Sequence[Union[Album, str]]*) – A sequence of artist objects or spotify IDs

**remove_saved_shows** (**shows*)

Delete one or more shows from current Spotify user's library.

**Parameters ids** (List[:class: *Show*]) – A list of the spotify.Show or unique spotify ids.

**Returns Result** – An empty dictionary if the request is successful.

**Return type** Dict

**remove_tracks** (**tracks*)

Remove one or more tracks from the current user's 'Your Music' library.

**Parameters tracks** (*Sequence[Union[Track, str]]*) – A sequence of track objects or spotify IDs

**save_albums** (**albums*)

Save one or more albums to the current user's 'Your Music' library.

**Parameters albums** (*Sequence[Union[Album, str]]*) – A sequence of artist objects or spotify IDs

**save_tracks** (**tracks*)

Save one or more tracks to the current user's 'Your Music' library.

**Parameters tracks** (*Sequence[Union[Track, str]]*) – A sequence of track objects or spotify IDs

### PlaylistTrack

**class** spotify.**PlaylistTrack** (*client, data*)

A Track on a Playlist.

Like a regular `Track` but has some additional attributes.

**added_by**

The Spotify user who added the track.

**Type** str

**is_local**

Whether this track is a local file or not.

**Type** bool

**added_at**

The datetime of when the track was added to the playlist.

**Type** datetime.datetime

### Device

**class** spotify.**Device** (*data*)

A Spotify Users device.

**id**
The device ID

**Type** str

**name**
The name of the device.

**Type** int

**type**
A Device type, such as “Computer”, “Smartphone” or “Speaker”.

**Type** str

**volume**
The current volume in percent. This may be null.

**Type** int

**is_active**
if this device is the currently active device.

**Type** bool

**is_restricted**
Whether controlling this device is restricted. At present if this is “true” then no Web API commands will be accepted by this device.

**Type** bool

**is_private_session**
If this device is currently in a private session.

**Type** bool

## Context

**class** `spotify.`**Context**(*data*)
A Spotify Context.

**type**
The object type, e.g. “artist”, “playlist”, “album”.

**Type** str

**href**
A link to the Web API endpoint providing full details of the track.

**Type** str

**external_urls**
External URLs for this context.

**Type** str

**uri**
The Spotify URI for the context.

**Type** str

### Image

**class** spotify.**Image**(*, *height: str*, *width: str*, *url: str*)

An object representing a Spotify image resource.

**height**

The height of the image.

**Type** str

**width**

The width of the image.

**Type** str

**url**

The URL of the image.

**Type** str

### Exceptions

#### SpotifyException

**class** spotify.**SpotifyException**

Base exception class for spotify.py.

#### HTTPException

**class** spotify.**HTTPException**(*response*, *message*)

A generic exception that's thrown when a HTTP operation fails.

#### Forbidden

**class** spotify.**Forbidden**(*response*, *message*)

An exception that's thrown when status code 403 occurs.

#### NotFound

**class** spotify.**NotFound**(*response*, *message*)

An exception that's thrown when status code 404 occurs.

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search

# Index

## A

add_playlist_tracks() (*spotify.HTTPClient method*), 10
add_tracks() (*spotify.Playlist method*), 33
add_tracks() (*spotify.User method*), 28
added_at (*spotify.PlaylistTrack attribute*), 39
added_by (*spotify.PlaylistTrack attribute*), 39
Album (*class in spotify*), 24
album() (*spotify.HTTPClient method*), 10
album_group (*spotify.Album attribute*), 24
album_tracks() (*spotify.HTTPClient method*), 10
album_type (*spotify.Album attribute*), 25
albums() (*spotify.HTTPClient method*), 10
Artist (*class in spotify*), 26
artist() (*spotify.HTTPClient method*), 11
artist_albums() (*spotify.HTTPClient method*), 11
artist_related_artists() (*spotify.HTTPClient method*), 11
artist_top_tracks() (*spotify.HTTPClient method*), 11
artists (*spotify.Album attribute*), 24
artists() (*spotify.HTTPClient method*), 11
audio_features() (*spotify.HTTPClient method*), 11
available_devices() (*spotify.HTTPClient method*), 11

## B

birthdate (*spotify.User attribute*), 28

## C

categories() (*spotify.HTTPClient method*), 11
category() (*spotify.HTTPClient method*), 12
category_playlists() (*spotify.HTTPClient method*), 12
change_playlist_details() (*spotify.HTTPClient method*), 12
check_saved_shows() (*spotify.HTTPClient method*), 12
check_saved_shows() (*spotify.Library method*), 38
clear() (*spotify.Playlist method*), 33
Client (*class in spotify*), 6
client_id (*spotify.Client attribute*), 7
client_id (*spotify.HTTPClient attribute*), 10
client_secret (*spotify.HTTPClient attribute*), 10
close() (*spotify.Client method*), 7
close() (*spotify.HTTPClient method*), 12
collaborative (*spotify.Playlist attribute*), 32
contains_albums() (*spotify.Library method*), 38
contains_tracks() (*spotify.Library method*), 38
Context (*class in spotify*), 40
copy() (*spotify.Playlist method*), 33
copyrights (*spotify.Album attribute*), 25
country (*spotify.User attribute*), 28
create_playlist() (*spotify.HTTPClient method*), 12
create_playlist() (*spotify.User method*), 28
current_player() (*spotify.HTTPClient method*), 13
current_playlists() (*spotify.HTTPClient method*), 13
current_user() (*spotify.HTTPClient method*), 13
currently_playing() (*spotify.HTTPClient method*), 13
currently_playing() (*spotify.User method*), 29

## D

delete_saved_albums() (*spotify.HTTPClient method*), 13
delete_saved_tracks() (*spotify.HTTPClient method*), 13
description (*spotify.Playlist attribute*), 32
Device (*class in spotify*), 39
device (*spotify.Player attribute*), 35
display_name (*spotify.User attribute*), 28

## E

edit_playlist() (*spotify.User method*), 29
email (*spotify.User attribute*), 28
enqueue() (*spotify.Player method*), 36

extend() (*spotify.Playlist method*), 33
external_urls (*spotify.Context attribute*), 40

## F

featured_playlists() (*spotify.HTTPClient method*), 13
follow_artist_or_user() (*spotify.HTTPClient method*), 14
follow_playlist() (*spotify.HTTPClient method*), 14
follow_playlist() (*spotify.User method*), 29
followed_artists() (*spotify.HTTPClient method*), 14
followers (*spotify.Artist attribute*), 26
followers (*spotify.Playlist attribute*), 32
followers (*spotify.User attribute*), 28
following_artists_or_users() (*spotify.HTTPClient method*), 14
following_playlists() (*spotify.HTTPClient method*), 14
Forbidden (*class in spotify*), 41
from_code() (*spotify.User class method*), 29
from_refresh_token() (*spotify.User class method*), 29
from_token() (*spotify.User class method*), 29

## G

genres (*spotify.Album attribute*), 25
genres (*spotify.Artist attribute*), 26
get_album() (*spotify.Client method*), 7
get_albums() (*spotify.Artist method*), 26
get_albums() (*spotify.Client method*), 7
get_albums() (*spotify.Library method*), 38
get_all_albums() (*spotify.Artist method*), 27
get_all_albums() (*spotify.Library method*), 38
get_all_playlists() (*spotify.User method*), 30
get_all_podcasts() (*spotify.Library method*), 38
get_all_tracks() (*spotify.Album method*), 25
get_all_tracks() (*spotify.Library method*), 38
get_all_tracks() (*spotify.Playlist method*), 33
get_artist() (*spotify.Client method*), 7
get_artists() (*spotify.Client method*), 7
get_bearer_info() (*spotify.HTTPClient method*), 14
get_devices() (*spotify.User method*), 30
get_episode() (*spotify.Client method*), 8
get_episode() (*spotify.HTTPClient method*), 14
get_multiple_episodes() (*spotify.HTTPClient method*), 14
get_multiple_shows() (*spotify.Client method*), 8
get_multiple_shows() (*spotify.HTTPClient method*), 15
get_player() (*spotify.User method*), 30
get_playlist() (*spotify.HTTPClient method*), 15
get_playlist_cover_image() (*spotify.HTTPClient method*), 15
get_playlist_tracks() (*spotify.HTTPClient method*), 15
get_playlists() (*spotify.HTTPClient method*), 16
get_playlists() (*spotify.User method*), 30
get_podcasts() (*spotify.User method*), 30
get_saved_shows() (*spotify.HTTPClient method*), 16
get_show() (*spotify.HTTPClient method*), 16
get_shows_episodes() (*spotify.HTTPClient method*), 16
get_track() (*spotify.Client method*), 8
get_tracks() (*spotify.Album method*), 25
get_tracks() (*spotify.Library method*), 38
get_tracks() (*spotify.Playlist method*), 34
get_user() (*spotify.Client method*), 8

## H

height (*spotify.Image attribute*), 41
href (*spotify.Album attribute*), 24
href (*spotify.Artist attribute*), 26
href (*spotify.Context attribute*), 40
href (*spotify.Playlist attribute*), 32
href (*spotify.User attribute*), 27
http (*spotify.Client attribute*), 7
HTTPClient (*class in spotify*), 9
HTTPException (*class in spotify*), 41

## I

id (*spotify.Album attribute*), 24
id (*spotify.Artist attribute*), 26
id (*spotify.Client attribute*), 8
id (*spotify.Device attribute*), 39
id (*spotify.Playlist attribute*), 32
id (*spotify.User attribute*), 27
Image (*class in spotify*), 41
images (*spotify.Artist attribute*), 26
images (*spotify.Playlist attribute*), 32
images (*spotify.User attribute*), 28
insert() (*spotify.Playlist method*), 34
is_active (*spotify.Device attribute*), 40
is_local (*spotify.PlaylistTrack attribute*), 39
is_playing (*spotify.Player attribute*), 36
is_private_session (*spotify.Device attribute*), 40
is_restricted (*spotify.Device attribute*), 40
is_saved_album() (*spotify.HTTPClient method*), 16
is_saved_track() (*spotify.HTTPClient method*), 17

## L

label (*spotify.Album attribute*), 25
Library (*class in spotify*), 38
loop (*spotify.Client attribute*), 7
loop (*spotify.HTTPClient attribute*), 10

M

markets (*spotify.Album attribute*), 25

N

name (*spotify.Album attribute*), 24
name (*spotify.Artist attribute*), 26
name (*spotify.Device attribute*), 40
name (*spotify.Playlist attribute*), 32
new_releases() (*spotify.HTTPClient method*), 17
next() (*spotify.Player method*), 36
NotFound (*class in spotify*), 41

O

oauth2_url() (*spotify.Client method*), 8
owner (*spotify.Playlist attribute*), 32

P

pause() (*spotify.Player method*), 36
pause_playback() (*spotify.HTTPClient method*), 17
play() (*spotify.Player method*), 36
play_playback() (*spotify.HTTPClient method*), 17
playback_queue() (*spotify.HTTPClient method*), 17
Player (*class in spotify*), 35
Playlist (*class in spotify*), 32
PlaylistTrack (*class in spotify*), 39
pop() (*spotify.Playlist method*), 34
popularity (*spotify.Album attribute*), 25
popularity (*spotify.Artist attribute*), 26
previous() (*spotify.Player method*), 36
product (*spotify.User attribute*), 28
public (*spotify.Playlist attribute*), 33

R

recently_played() (*spotify.HTTPClient method*), 17
recently_played() (*spotify.User method*), 30
recommendations() (*spotify.HTTPClient method*), 18
related_artists() (*spotify.Artist method*), 27
release_date (*spotify.Album attribute*), 25
release_date_precision (*spotify.Album attribute*), 25
remove() (*spotify.Playlist method*), 34
remove_albums() (*spotify.Library method*), 39
remove_playlist_tracks() (*spotify.HTTPClient method*), 18
remove_saved_shows() (*spotify.HTTPClient method*), 19
remove_saved_shows() (*spotify.Library method*), 39
remove_tracks() (*spotify.Library method*), 39
remove_tracks() (*spotify.Playlist method*), 34
remove_tracks() (*spotify.User method*), 30
reorder_playlists_tracks() (*spotify.HTTPClient method*), 19
reorder_tracks() (*spotify.Playlist method*), 34
reorder_tracks() (*spotify.User method*), 31
repeat_playback() (*spotify.HTTPClient method*), 19
repeat_state (*spotify.Player attribute*), 35
replace_playlist_tracks() (*spotify.HTTPClient method*), 20
replace_tracks() (*spotify.Playlist method*), 35
replace_tracks() (*spotify.User method*), 31
request() (*spotify.HTTPClient method*), 20
resume() (*spotify.Player method*), 37
reverse() (*spotify.Playlist method*), 35
route() (*spotify.HTTPClient static method*), 20

S

save_albums() (*spotify.HTTPClient method*), 20
save_albums() (*spotify.Library method*), 39
save_shows() (*spotify.HTTPClient method*), 20
save_tracks() (*spotify.HTTPClient method*), 20
save_tracks() (*spotify.Library method*), 39
saved_albums() (*spotify.HTTPClient method*), 20
saved_tracks() (*spotify.HTTPClient method*), 21
search() (*spotify.Client method*), 8
search() (*spotify.HTTPClient method*), 21
seek() (*spotify.Player method*), 37
seek_playback() (*spotify.HTTPClient method*), 22
set_playback_volume() (*spotify.HTTPClient method*), 22
set_repeat() (*spotify.Player method*), 37
set_volume() (*spotify.Player method*), 37
shuffle() (*spotify.Player method*), 37
shuffle_playback() (*spotify.HTTPClient method*), 22
shuffle_state (*spotify.Player attribute*), 35
skip_next() (*spotify.HTTPClient method*), 22
skip_previous() (*spotify.HTTPClient method*), 22
snapshot_id (*spotify.Playlist attribute*), 33
sort() (*spotify.Playlist method*), 35
SpotifyException (*class in spotify*), 41

T

top_artists() (*spotify.User method*), 31
top_artists_or_tracks() (*spotify.HTTPClient method*), 22
top_tracks() (*spotify.Artist method*), 27
top_tracks() (*spotify.User method*), 31
total_albums() (*spotify.Artist method*), 27
track() (*spotify.HTTPClient method*), 23
track_audio_analysis() (*spotify.HTTPClient method*), 23
track_audio_features() (*spotify.HTTPClient method*), 23

`tracks` (*spotify.Playlist attribute*), 33
`tracks()` (*spotify.HTTPClient method*), 23
`transfer()` (*spotify.Player method*), 37
`transfer_player()` (*spotify.HTTPClient method*), 23
`type` (*spotify.Context attribute*), 40
`type` (*spotify.Device attribute*), 40

## U

`unfollow_artists_or_users()` (*spotify.HTTPClient method*), 24
`unfollow_playlist()` (*spotify.HTTPClient method*), 24
`upload_playlist_cover_image()` (*spotify.HTTPClient method*), 24
`uri` (*spotify.Album attribute*), 24
`uri` (*spotify.Artist attribute*), 26
`uri` (*spotify.Context attribute*), 40
`uri` (*spotify.Playlist attribute*), 35
`uri` (*spotify.User attribute*), 27
`url` (*spotify.Artist attribute*), 26
`url` (*spotify.Image attribute*), 41
`url` (*spotify.Playlist attribute*), 32
`url` (*spotify.User attribute*), 27
`User` (*class in spotify*), 27
`user` (*spotify.Library attribute*), 38
`user()` (*spotify.HTTPClient method*), 24
`user_from_token()` (*spotify.Client method*), 9

## V

`volume` (*spotify.Device attribute*), 40

## W

`width` (*spotify.Image attribute*), 41