# pyspotify Documentation

***Release 1.12***

**Doug Winter and contributors**

July 17, 2015

Contents

pyspotify provides a Python interface to Spotify's online music streaming service.

To use this package you will also need libspotify, which is available from Spotify. You will need a Spotify Premium account. You will also need to apply for, and receive an API key from Spotify.

CHAPTER 1

# Project resources

- Documentation
- Source code
- Issue tracker
- CI server
- IRC: `#mopidy` at irc.freenode.net
- Download development snapshot

CHAPTER 2

# Table of contents

## 2.1 Introduction

pyspotify provides a Python interface to Spotify's online music streaming service.

### 2.1.1 Completion status

pyspotify is very usable, and implements most of the libspotify 0.0.8 API. The table below shows what is done and what is left to be done.

| Subsystem | Status |
|---|---|
| Error handling | Complete |
| Session handling | Complete |
| Link subsystem | Complete |
| Track subsystem | Complete |
| Album subsystem | Complete |
| Artist subsystem | Complete |
| Album browsing | Complete |
| Artist browsing | Complete |
| Image handling | Incomplete |
| Search subsystem | Complete |
| Playlist subsystem | Incomplete |
| User handling | Complete |
| Toplist handling | Complete |
| Inbox subsystem | Not available |
| Offline synchronization | Not available |

### 2.1.2 Requirements

To use this package you will also need libspotify, which is available from Spotify.

You will need a Spotify Premium account. You will also need to apply for, and receive an API key from Spotify.

### 2.1.3 Installation

#### Debian package

For Ubuntu and Debian users, *pyspotify* can be found in the `python-spotify` package of the Mopidy APT archive.

#### Arch Linux package

Install the `pyspotify-git` package from the AUR.

#### Using Pip (latest stable release)

The `pip` program for installing Python packages is usually found in the `python-pip` package of your Linux distribution.

To install `pyspotify`, run:

```
sudo pip install -U pyspotify
```

To update an existing installation, simply use the same command.

#### Using Pip (latest development version)

To install the very latest git version of pyspotify:

```
sudo pip install -U pyspotify==dev
```

#### Using setuptools (latest git version)

You will have first to clone the git repository.

Then to install it:

```
sudo python setup.py install
```

### 2.1.4 Trying it out

Included with pyspotify is a simple program, `examples/jukebox.py`. Run this with your credentials and access to an API key, and it will let you browse and play from your playlists, conduct searches and play from Spotify URIs.

## 2.2 Managers

The role of the pyspotify *managers* is to help the developer by abstracting basic operations of the Spotify library. They include callbacks that you can implement if you wish to be notified of particular Spotify events.

The one you certainly want to use is the *`SpotifySessionManager`*, as it handles all login and other basic session operations.

### 2.2.1 Session manager

**class** spotify.manager.**SpotifySessionManager**(*username=None*, *password=None*, *remember_me=False*, *login_blob=''*, *proxy=None*, *proxy_username=None*, *proxy_password=None*)

Client for Spotify. Inherit from this class to have your callbacks called on the appropriate events.

Exceptions raised in your callback handlers will be displayed on the standard error output (stderr).

**When logging in a user, the application can pass one of:**

- *username* + *password*: standard login using a plaintext password
- nothing: logs in the last user which credentials have been stored using *remember_me*.
- *username* + *login_blob*: the blob is encrypted data from *libspotify*, for when a multi-user application wants to use the re-login feature. The blob is obtained from the *credentials_blob_updated()* callback after a successful login to the Spotify AP.

**When behind a proxy, the application can specify:**

- ***proxy*: url to the proxy server that should be used. The format** is `protocol://<host>:port` (where protocol is `http/https/socks4/socks5`)
- *proxy_username*: username to authenticate with the proxy server.
- *proxy_password*: password to authenticate with the proxy server.

**connect**()

Connect to the Spotify API using the given username and password. If `username` is `None`, reconnection of the last user will be attempted.

This method does not return before we disconnect from the Spotify service.

**disconnect**()

Terminate the current Spotify session.

**logged_in**(*session*, *error*)

Callback.

Called when the login completes. You almost certainly want to do something with *session.playlist_container()* if the login succeded.

**Parameters**

- **session** (*spotify.Session*) – the current session.
- **error** (string or `None`) – an error message, `None` if all went well.

**logged_out**(*session*)

Callback.

The user has or has been logged out from Spotify.

**Parameters** **session** (*spotify.Session*) – the current session.

**metadata_updated**(*session*)

Callback.

The current user's metadata has been updated.

**Parameters** **session** (*spotify.Session*) – the current session.

**connection_error** (*session*, *error*)
Callback.

A connection error occured in `libspotify`.

**Parameters**

- **session** (*spotify.Session*) – the current session.
- **error** (string or `None`) – an error message. If `None`, the connection is back.

**message_to_user** (*session*, *message*)
Callback.

An informative message from `libspotify`, destinated to the user.

**Parameters**

- **session** (*spotify.Session*) – the current session.
- **message** (*string*) – a message.

**notify_main_thread** (*session=None*)
Callback.

When this method is called by `libspotify`, one should call *session.process_events()*.

If you use the `SessionManager`'s default loop, the default implementation of this method does the job. Though, if you implement your own loop for handling Spotify events, you'll need to override this method.

> **Warning:** This method is called from an internal thread in libspotify. You should make sure *not* to use the Spotify API from within it, as libspotify isn't thread safe.

**Parameters** **session** (*spotify.Session*) – the current session.

**music_delivery** (*session*, *frames*, *frame_size*, *num_frames*, *sample_type*, *sample_rate*, *channels*)
Callback.

Called whenever new music data arrives from Spotify.

You should override this method *or* *music_delivery_safe()*, not both.

> **Warning:** This method is called from an internal thread in libspotify. You should make sure *not* to use the Spotify API from within it, as libspotify isn't thread safe.

**Parameters**

- **session** (*spotify.Session*) – the current session
- **frames** (`buffer`) – the audio data
- **frame_size** (`int`) – bytes per frame
- **num_frames** (`int`) – number of frames in this delivery
- **sample_type** (`int`) – currently this is always 0 which means 16-bit signed native endian integer samples
- **sample_rate** (`int`) – audio sample rate, in samples per second
- **channels** (`int`) – number of audio channels. Currently 1 or 2

**Returns** number of frames consumed

**Return type** `int`

**music_delivery_safe**(*session*, *frames*, *frame_size*, *num_frames*, *sample_type*, *sample_rate*, *channels*)
This method does the same as *music_delivery()*, except that it's called from the *SpotifySessionManager* loop. You can safely use Spotify APIs from within this method.

You should override this method *or* *music_delivery()*, not both.

**play_token_lost**(*session*)
Callback.

The playback stopped because a track was played from another application, with the same account.

**Parameters** **session** (*spotify.Session*) – the current session.

**log_message**(*session*, *message*)
Callback.

A log message from `libspotify`.

**Parameters**

- **session** (*spotify.Session*) – the current session.
- **message** (*string*) – the message.

**end_of_track**(*session*)
Callback.

Playback has reached the end of the current track.

**Parameters** **session** (*spotify.Session*) – the current session.

**credentials_blob_updated**(*session*, *blob*)
Callback.

Called when storable credentials have been updated, usually called when we have connected to the AP.

**Warning:** This method is called from an internal thread in libspotify. You should make sure *not* to use the Spotify API from within it, as libspotify isn't thread safe.

**Parameters**

- **session** (*spotify.Session*) – the current session.
- **blob** – a string which contains an encrypted token that can be stored safely on disk instead of storing plaintext passwords.

### 2.2.2 Playlist manager

**class** `spotify.manager.`**SpotifyPlaylistManager**
Handles Spotify playlists callbacks. To implement you own callbacks, inherit from this class.

Exceptions raised in your callback handlers will be displayed on the standard error output (stderr).

**watch**(*playlist*, *userdata=None*)
Listen to modifications events on a playlist.

**unwatch**(*playlist*, *userdata=None*)
Stop listening to events on the playlist.

**tracks_added**(*playlist*, *tracks*, *position*, *userdata*)
Callback

Called when tracks are inserted in the playlist.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **tracks** (list of *spotify.Track*) – list of added tracks
- **position** (int) – position in which the tracks where inserted

**tracks_moved**(*playlist*, *tracks*, *new_position*, *userdata*)
Callback

Called when tracks are moved in the playlist.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **tracks** (list of *spotify.Track*) – list of moved tracks
- **new_position** (int) – new position of the tracks

**tracks_removed**(*playlist*, *tracks*, *userdata*)
Callback

Called when tracks are removed from the playlist.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **tracks** (list of *spotify.Track*) – list of removed tracks

**playlist_renamed**(*playlist*, *userdata*)
Callback

Called when a playlist has been renamed. *spotify.Playlist.name()* can be used to find out the new name.

**Parameters** **playlist** (*spotify.Playlist*) – playlist on which the event occured

**playlist_state_changed**(*playlist*, *userdata*)
Callback

Called when state changed for a playlist.

There are three states that trigger this callback:

•Collaboration for this playlist has been turned on or off.

•The playlist started having pending changes, or all pending changes have now been committed.

•The playlist started loading, or finished loading.

**Parameters** **playlist** (*spotify.Playlist*) – playlist on which the event occured

**playlist_update_in_progress**(*playlist*, *done*, *userdata*)
Callback

Called when a playlist is updating or is done updating.

This is called before and after a series of changes are applied to the playlist. It allows e.g. the user interface to defer updating until the entire operation is complete.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **done** (bool) – whether the update is finished

**playlist_metadata_updated**(*playlist*, *userdata*)
Callback

Called when metadata for one or more tracks in a playlist has been updated.

**Parameters** **playlist** (*spotify.Playlist*) – playlist on which the event occured

**track_created_changed**(*playlist*, *position*, *user*, *when*, *userdata*)
Callback

Called when create time and/or creator for a playlist entry changes.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **position** (int) – track's position in the playlist
- **user** (*spotify.User*) – creator of the playlist
- **when** (int) – time in seconds since the UNIX Epoch

**track_message_changed**(*playlist*, *position*, *message*, *userdata*)
Callback

Called when message attribute for a playlist entry changes.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **position** (int) – track's position in the playlist
- **message** (unicode) – new message

**track_seen_changed**(*playlist*, *position*, *seen*, *userdata*)
Callback

Called when seen attribute for a playlist entry changes.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **position** (int) – track's position in the playlist
- **seen** (bool) – new seen attribute

**description_changed**(*playlist*, *description*, *userdata*)
Callback

Called when playlist description has changed.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **description** (unicode) – new description

**subscribers_changed**(*playlist*, *userdata*)
Callback

Called when playlist subscribers changes (count or list of names).

**Parameters** **playlist** (*spotify.Playlist*) – playlist on which the event occured

**image_changed**(*playlist*, *image*, *userdata*)
Callback

Called when playlist image has changed.

**Parameters**

- **playlist** (*spotify.Playlist*) – playlist on which the event occured
- **image** (str) – image id of the new image

### 2.2.3 Container manager

**class** spotify.manager.**SpotifyContainerManager**
Handles Spotify playlist container callbacks. To implement you own callbacks, inherit from this class.

Exceptions raised in your callback handlers will be displayed on the standard error output (stderr).

**watch**(*container*, *userdata=None*)
Listen to modifications events on a container.

**unwatch**(*container*, *userdata=None*)
Stop listenning to events on the container.

**container_loaded**(*container*, *userdata*)
Callback

This callback is called when the container has finished loading. This event happens after a successful login, or after a modification on the container (after the 3 callbacks described below).

**Parameters** **container** (*spotify.PlaylistContainer*) – a playlist container

**playlist_added**(*container*, *playlist*, *position*, *userdata*)
Callback

Called when a new playlist is added to the container.

**Parameters**

- **container** (*spotify.PlaylistContainer*) – a playlist container
- **playlist** (*spotify.Playlist*) – playlist added to the container
- **position** (int) – new position of the playlist

**playlist_moved**(*container*, *playlist*, *position*, *new_position*, *userdata*)
Callback

Called when a playlist is moved in the container.

**Parameters**

- **container** (*spotify.PlaylistContainer*) – a playlist container
- **playlist** (*spotify.Playlist*) – playlist moved
- **position** (int) – old position
- **new_position** (int) –

**playlist_removed**(*container*, *playlist*, *position*, *userdata*)
Callback

Called when a playlist is removed from the container.

**Parameters**

- **container** (*spotify.PlaylistContainer*) – a playlist container
- **playlist** (*spotify.Playlist*) – playlist removed
- **position** (int) – old position

## 2.3 Audio sinks

The *spotify.audiosink* module provides audio sink wrappers for different audio sinks like ALSA, OSS, and PortAudio.

spotify.audiosink.**import_audio_sink**(*audio_sinks=None*)

Try to import each audio sink until one is successfully imported.

The *audio_sinks* parameter specificies what audio sinks to import, in the given order. If *audio_sinks* is not provided, the list spotify.audiosink.AUDIO_SINKS will be used.

**Parameters** **audio_sinks** (*list of two-tuples of (modulename, classname)*) – audio sinks to try to import

**Returns** the first audio sink that was successfully imported

**Return type** class

**Raise** ImportError if no audio sinks can be imported

**class** spotify.audiosink.**BaseAudioSink**(*backend=None*)

*BaseAudioSink* provides the interface which is implemented by all audio sink wrappers in the *spotify.audiosink* module.

The interface is a perfect match for the *spotify.manager.SpotifySessionManager.music_delivery()* method, making it easy to play audio data received from Spotify.

**music_delivery**(*session*, *frames*, *frame_size*, *num_frames*, *sample_type*, *sample_rate*, *channels*)

To use one of the bundled audio controllers in a Spotify client you develop, just call this method every time you get audio data from Spotify, e.g. from your implementation of *spotify.manager.SpotifySessionManager.music_delivery()*.

**Parameters**

- **session** (*spotify.Session*) – the current session
- **frames** (buffer) – the audio data
- **frame_size** (int) – bytes per frame
- **num_frames** (int) – number of frames in this delivery
- **sample_type** (int) – currently this is always 0 which means 16-bit signed native endian integer samples
- **sample_rate** (int) – audio sample rate, in samples per second
- **channels** (int) – number of audio channels. Currently 1 or 2

**Returns** number of frames consumed

**Return type** int

**start**()

Should be called when audio output starts.

This is a hook for the audio sink to do work just before the audio starts.

**stop**()
Should be called when audio output stops.

This is a hook for the audio sink to do work just after the audio stops.

**pause**()
Should be called when audio output is paused.

This is a hook for the audio sink to do work when the audio is paused.

### 2.3.1 Implementations

Implementations of the *BaseAudioSink* interface include:

**class** spotify.audiosink.alsa.**AlsaSink**
Requires a system using ALSA, which includes most Linux systems, and the pyalsaaudio library.

**class** spotify.audiosink.oss.**OssSink**
Requires a system using OSS or with an OSS emulation, typically a Linux or BSD system. Uses the ossaudiodev module from the Python standard library.

**class** spotify.audiosink.portaudio.**PortAudioSink**
Requires a system with the PortAudio library installed and the Python binding pyaudio. The PortAudio library is available for both Linux, Mac OS X, and Windows.

**class** spotify.audiosink.gstreamer.**GstreamerSink**
Requires a system with Gstreamer installed and the Python bindings gst-python. The Gstreamer library is available for both Linux, Mac OS X, and Windows. Though, it isn't always trivial to install Gstreamer.

## 2.4 API Reference

This is the API reference for the pyspotify project. It is structured similarly to the libspotify API documentation. All the classes and submodules of pyspotify are contained in the *spotify* parent module.

### 2.4.1 Error handling

**exception** spotify.**SpotifyError**

This is the exception used by *pyspotify* to transmit Spotify errors to the Python application.

### 2.4.2 Session settings

**class** spotify.**Settings**
This class stores settings used when creating a Spotify session.

**application_key = None**
Your application key (binary string).

**cache_location = 'tmp'**
The location where Spotify will write cache files. This cache include tracks, cached browse results and coverarts. Set to '' to disable cache.

**proxy = None**
Url to the proxy server that should be used. The format is protocol://<host>:port (where protocol is http/https/socks4/socks5).

**proxy_password = None**
Password to authenticate with the proxy server.

**proxy_username = None**
Username to authenticate with proxy server.

**settings_location = 'tmp'**
The location where Spotify will write setting files and per-user cache items. This includes playlists, track metadata, etc. 'settings_location' may be the same path as 'cache_location'. 'settings_location' folder will not be created (unlike 'cache_location'), if you don't want to create the folder yourself, you can set 'settings_location' to 'cache_location'.

**user_agent = 'pyspotify-example'**
"User-Agent" for your application - max 255 characters long. The User-Agent should be a relevant, customer facing identification of your application.

### 2.4.3 Session handling

The session handling is usually done by inheriting the *spotify.manager.SpotifySessionManager* class from the *spotify.manager* module. Then the manager's `connect()` method calls the *Session.create()* and *Session.login()* functions.

#### The `Session` class

**class** spotify.**Session**
A Spotify session object.

**classmethod create**(*manager*, *settings*)
Creates a new Spotify session. Call once per process.

**Parameters**

- **manager** – an object that has the session callbacks as methods
- **settings** – an *Settings* object

**Returns** a *Session* object embedding the newly created Spotify session

**login**(*username*[, *password*, *remember_me*, *blob*])
Logs in the specified user to the Spotify service.

The application must not store any user password in plain text. If password storage is needed, the application must store the encrypted binary blob corresponding to the user and obtained via the *manager.SpotifySessionManager.credentials_blob_updated()* session callback. One of `password` or `blob` must be specified.

**Parameters**

- **username** (*string*) – the user's login to Spotify Premium
- **password** (*string*) – the user's password to Spotify Premium
- **remember_me** (`bool`) – set this flag if you want libspotify to remember this user
- **blob** (`str`) – binary login blob

**relogin**()
Use this method if you want to re-login the last user who set the `remember_me` flag in *Session.login()*

**browse_album**(*album*, *callback*[, *userdata*])
Browse an album, calling the callback when the browser's metadata is loaded.

**Parameters**

- **album** (*Album*) – a Spotify album (does not have to be loaded)
- **callback** – a function with signature : `(AlbumBrowser browser, Object userdata)`
- **userdata** – any object

**Returns** An *AlbumBrowser* object containing the results

Deprecated since version 1.7: Use *AlbumBrowser* instead.

**browse_artist**(*artist*, *callback*[, *userdata*])
Browse an artist, calling the callback when the browser's metadata is loaded.

**Parameters**

- **artist** (*Artist*) – a Spotify artist (does not have to be loaded)
- **callback** – a function with signature : `(ArtistBrowser browser, Object userdata)`
- **userdata** – any object

**Returns** An *ArtistBrowser* object containing the results.

Deprecated since version 1.7: Use *ArtistBrowser* instead.

**display_name**()

**Return type** string

**Returns** the full name for the logged in user.

Raises *SpotifyError* if not logged in.

**flush_caches**()
This will make libspotify write all data that is meant to be stored on disk to the disk immediately. libspotify does this periodically by itself and also on logout. So under normal conditions this should never need to be used.

**image_create**(*id*)

**Parameters** **id** (*string*) – the id of the image to be fetched.

**Returns** an *Image* object.

Create an image of album cover art.

**is_available**(*track*)

**Parameters** **track** (*Track*) – a track

**Return type** `int`

**Returns** Whether the track is available for playback.

**load**(*track*)

**Parameters** **track** (*Track*) – a track

**Raises** *SpotifyError*

Load the specified track on the player.

**logout**()
Logout from the session.

**play**(*play*)

**Parameters** **play** (*int*) – Pause playback if `0`, else play.

Play or pause the currently loaded track

**playlist_container**()

**Return type** *PlaylistContainer*

**Returns** the playlist container for the currently logged in user.

**process_events**()
Make the *libspotify* library process any pending event. This should be called from the *notify_main_thread* session callback.

**search**(*query*, *callback*[, *track_offset=0*, *track_count=32*, *album_offset=0*, *album_count=32*, *artist_offset=0*, *artist_count=32*, *playlist_offset=0*, *playlist_count=32*, *search_type='standard'*, *userdata=None* ])

**Parameters**

- **query** – Query search string
- **callback** – signature `(Results results, Object userdata)`
- **track_offset** – The offset among the tracks of the result
- **track_count** – The number of tracks to ask for
- **album_offset** – The offset among the albums of the result
- **album_count** – The number of albums to ask for
- **artist_offset** – The offset among the artists of the result
- **artist_count** – The number of artists to ask for
- **playlist_offset** – The offset among the playlists of the result
- **playlist_count** – The number of playlists to ask for
- **search_type** – 'standard' or 'suggest'

**Returns** The search results

**Return type** *Results*

Conduct a search, calling the callback when the results are available.

**seek**(*offset*)
Seek to *offset* (in milliseconds) in the currently loaded track.

**set_preferred_bitrate**(*bitrate*)
Set the preferred bitrate for the audio stream.

**0:** 160 kbps

**1:** 320 kbps

**2:** 96 kbps

**starred**()

**Return type** *Playlist* object.

**Returns** the playlist of starred tracks for the logged in user.

**unload**()
Stop the currently playing track and unloads it from the player.

**user_is_loaded**()
Return whether the user is loaded or not, as an *int*.

If the user is not logged in, this method raises a *SpotifyError*.

**username**()
Return a string containing the canonical username for the logged in user.

If the user is not logged in, this method raises a *SpotifyError*.

### 2.4.4 Spotify links (URIs)

#### The Link class

**class** spotify.**Link**
Link objects

**LINK_INVALID**
Link type not valid - default until the library has parsed the link, or when parsing failed.

**LINK_TRACK**

**LINK_ALBUM**

**LINK_ARTIST**

**LINK_SEARCH**

**LINK_PLAYLIST**

**LINK_PROFILE**

**LINK_STARRED**

**LINK_LOCALTRACK**

**LINK_IMAGE**

**__str__**()
Return the link as a string in the Spotify format.

Example: `spotify:track:5st5644IlBmKiiRE73UsoZ`

**type**()
Return the type of the link as an `int`. Check value against the `LINK_*` types.

**as_album**()

**Returns** the link as an *Album* object.

**Return type** *Album*

**as_artist**()

**Returns** the link as an *Artist* object.

**Return type** *Artist*

**as_playlist**()

**Returns** the link as a *Playlist* object.

**Return type** *Playlist*

**as_track**()

**Returns** the link as a *Track* object.

**Return type** *Track*

static **from_album**(*album*)

**Parameters** **album** (*Album*) – an album

**Returns** link to the album

**Return type** *Link*

**Raises** *SpotifyError*

Create a new *Link* object from an *Album* object.

static **from_artist**(*artist*)

**Parameters** **artist** (*Artist*) – an artist

**Returns** link to the artist

**Return type** *Link*

**Raises** *SpotifyError*

Create a new *Link* object from an *Artist* object.

static **from_playlist**(*playlist*)

**Parameters** **playlist** (*Playlist*) – a playlist

**Returns** link to the playlist

**Return type** *Link*

**Raises** *SpotifyError*

Create a new *Link* object from a *Playlist* object.

static **from_search**(*results*)

**Parameters** **results** (*Results*) – a result set

**Returns** link to the result set

**Return type** *Link*

**Raises** *SpotifyError*

Create a new *Link* object from a *Results* object.

static **from_string**(*s*)

**Parameters** **s** (string) – a Spotify URI

**Returns** link to the same resource as the URI string

**Return type** *Link*

**Raises** *SpotifyError*

Create a new *Link* object from a string. Raises an exception if the string is not a valid Spotify URI.

static **from_track**(*track*[, *offset*])

**Parameters**

- **track** (*Track*) – a track

- **offset** (int) – offset in milliseconds from the start of the track

**Returns** link to the result set

**Return type** *Link*

**Raises** *SpotifyError*

Create a new *Link* object from a *Track* object, and optionally a time offset in milliseconds from the start of the track.

### 2.4.5 Track subsystem

#### The Track class

**class** spotify.**Track**

Track objects

**album**()

**Return type** *Album*

**Returns** The album of this track.

**artists**()

**Return type** List of *Artist*

**Returns** The artists who performed on this track.

**availability**()

**Return type** integer

**Returns** The availability status of this track.

**Possible values:**

- 0: Track is not available
- 1: Track is available and can be played
- 2: Track can not be streamed using this account
- 3: Track not available on artist's request

**disc**()

**Return type** int

**Returns** The disc number of this track.

---

**Note:** This function returns valid data only for tracks appearing in a browse artist or browse album result (otherwise returns 0).

---

**duration**()

**Return type** int

**Returns** The duration of this track, in milliseconds.

**error**()

**Return type** int

**Returns** An error code associated with this track. For example if it could not load.

**index**()

**Return type** int

**Returns** The position of this track on its disc.

---

**Note:** This function returns valid data only for tracks appearing in a browse artist or browse album result (otherwise returns 0).

---

**is_autolinked**()

**Return type** int

**Returns** whether this track is linked to another track

If the application wants to play this track, it has to call *playable()* on the track to obtain the linked track.

**is_loaded**()

**Return type** int

**Returns** Load status for this track.

---

**Note:** If the track is not loaded yet, all other functions operating on the track return default values.

---

**is_local**()

**Return type** bool

**Returns** True if track is a local file.

---

**Note:** The track must be loaded or this function will always return False.

---

**name**()

**Return type** String

**Returns** The name of this track.

**playable**()

**Return type** *Track*

**Returns** the playable track for this track

When calling this method on a linked track, returns the linked track. On a non-linked track, returns the track itself.

**popularity**()

**Return type** int

**Returns** The popularity of this track.

**starred**(*session*[, *set*])

**Parameters**

- **session** (*Session*) – The current session.
- **set** (bool) – If given, set the starred status of the track.

**Return type** bool

**Returns** Whether the track is starred or not.

### 2.4.6 Album subsystem

#### The `Album` class

**class** spotify.**Album**
Album objects

**ALBUM**

**SINGLE**

**COMPILATION**

**UNKNOW**

**artist**()

**Return type** *Artist*

**Returns** the artist associated with this album

**cover**()

**Return type** string

**Returns** the id of the cover data associated with this album

**is_available**()

**Return type** int

**Returns** 1 if the album is available, 0 if not.

**is_loaded**()

**Return type** int

**Returns** 1 if this album has been loaded by the client, 0 if not.

**name**()

**Return type** string

**Returns** the name of the album.

**type**()

**Return type** int

**Returns** the type of the album.

You can check the value against one of ALBUM, SINGLE, COMPILATION or UNKNOWN.

**year**()

**Return type** int

**Returns** the year in which the album was released.

### 2.4.7 Artist subsystem

#### The `Artist` class

**class** `spotify.`**`Artist`**
Artist objects.

**`is_loaded`**`()`

**Return type** `int`

**Returns** `1` if this artist has been loaded by the client, `0` if not.

**`name`**`()`

**Return type** string

**Returns** the name of the artist

### 2.4.8 Album browsing

#### The `AlbumBrowser` class

**class** `spotify.`**`AlbumBrowser`**(*album*[, *callback*[, *userdata*]])

**Note:** A sequence of *`Track`* objects.

Browse an album, calling the callback when the browser's metadata is loaded.

**Parameters**

- **`album`** (*`Album`*) – a Spotify album (does not have to be loaded)
- **`callback`** – a function with signature : `(AlbumBrowser browser, Object userdata)`
- **`userdata`** – any object

**`is_loaded`**`()`

**Return type** `int`

**Returns** whether this album browser has finished loading metadata.

### 2.4.9 Artist browsing

#### The `ArtistBrowser` class

**class** `spotify.`**`ArtistBrowser`**(*artist*[, *type*[, *callback*[, *userdata*]]])

**Note:** A sequence of *`Track`* objects.

Browse an artist, calling the callback when the browser's metadata is loaded.

**Parameters**

- **`artist`** (*`Artist`*) – a Spotify artist (does not have to be loaded)

- **type** – this browser's type. One of:
  - `'full'` (default): all data will be fetched (deprecated in pyspotify 1.7 / libspotify 11)
  - `'no_tracks'`: no information about tracks
  - `'no_albums'`: no information about albums (implies `'no_tracks'`)

  The `'no_tracks'` and `'no_albums'` browser types also include a list of top tracks for this artist.
- **callback** – a function with signature : `(ArtistBrowser browser, Object userdata)`
- **userdata** – any object

**is_loaded**()

**Return type** `int`

**Returns** whether this artist browser has finished loading metadata.

**albums**()

**Return type** list of *Album*

**Returns** the list of albums found while browsing

**similar_artists**()

**Return type** list of *Artist*

**Returns** the list of similar artists found while browsing

**tracks**()

**Return type** list of *Track*

**Returns** the list of tracks found while browsing

**tophit_tracks**()

**Return type** list of *Track*

**Returns** the list of top tracks for this artist found while browsing

### 2.4.10 Image subsystem

**Warning:** Spotify images support in **pyspotify** is not complete yet, use at your own risk, or take a look at the code.

#### The `Image` class

**class** `spotify.`**Image**

Image objects

**add_load_callback**()

Add a load callback

**data**()

Get image data

**error**()
Check if image retrieval returned an error code

**format**()
Get image format (currently only JPEG)

**image_id**()
Get image ID

**is_loaded**()
True if this Image has been loaded by the client

**remove_load_callback**()
Remove a load callback

### 2.4.11 Search subsystem

#### The `Results` class

**class** spotify.**Results**
Results corresponding to a search query.

**albums**()

**Return type** list of *Album*

**Returns** albums found by the search.

**artists**()

**Return type** list of *Artist*

**Returns** artists found by the search.

**did_you_mean**()

**Return type** string

**Returns** A query suggestion by Spotify.

**error**()

**Return type** int

**Returns** check if an error happened. 0 means no error.

**is_loaded**()

**Return type** int

**Returns** Whether the results metadata are loaded.

**query**()

**Return type** string

**Returns** the query expression that generated these results.

**total_albums**()

**Return type** int

**Returns** the total number of albums available for this search query.

**Note:** If this value is larger than the interval specified at creation of the search object, more search results are available. To fetch these, create a new search object with a new interval.

**total_artists**()

**Return type** int

**Returns** the total number of artists available for this search query.

**Note:** If this value is larger than the interval specified at creation of the search object, more search results are available. To fetch these, create a new search object with a new interval.

**total_tracks**()

**Return type** int

**Returns** the total number of tracks available for this search query.

**Note:** If this value is larger than the interval specified at creation of the search object, more search results are available. To fetch these, create a new search object with a new interval.

**tracks**()

**Return type** list of *Track*

**Returns** tracks found by the search.

### 2.4.12 Playlists

#### The Playlist class

*Playlist* objects are iterable: they are a list of *Track* objects.

**class** spotify.**Playlist**
Playlist objects.

**add_tracks**(*position*, *tracks*)

**Parameters**

- **position** (int) – where to add the tracks in the playlist
- **tracks** (list of *Track*) – tracks to add to the playlist

**add_tracks_added_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, list of *Track* tracks, int position, Object userdata)
- **userdata** – any object you would like to access in the callback

**add_tracks_moved_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, list of *Track* tracks, int new_position, Object userdata)
- **userdata** – any object you would like to access in the callback

**add_tracks_removed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, list of *Track* tracks, `Object` userdata)
- **userdata** – any object you would like to access in the callback

**add_playlist_renamed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, `Object` userdata)
- **userdata** – any object you would like to access in the callback

**add_playlist_state_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, `Object` userdata)
- **userdata** – any object you would like to access in the callback

Called when state changed for a playlist.

There are three states that trigger this callback:

•Collaboration for this playlist has been turned on or off

•**The playlist started having pending changes, or all pending** changes have now been committed

•The playlist started loading, or finished loading

**add_playlist_update_in_progress_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, `bool` done, `Object` userdata)
- **userdata** – any object you would like to access in the callback

Called when a playlist is updating or is done updating.

This is called before and after a series of changes are applied to the playlist. It allows e.g. the user interface to defer updating until the entire operation is complete.

**add_playlist_metadata_updated_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, `Object` userdata)
- **userdata** – any object you would like to access in the callback

**add_track_created_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, `int` position, *User* user, `int` when, `Object` userdata)
- **userdata** – any object you would like to access in the callback

*user* is the new user information and *when* is a time in seconds since the UNIX Epoch.

**add_track_message_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, int position, unicode message, Object userdata)
- **userdata** – any object you would like to access in the callback

**add_track_seen_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, int position, bool seen, Object userdata)
- **userdata** – any object you would like to access in the callback

**add_description_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, unicode description, Object userdata)
- **userdata** – any object you would like to access in the callback

**add_subscribers_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, Object userdata)
- **userdata** – any object you would like to access in the callback

**add_image_changed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*Playlist* p, str image_id, Object userdata)
- **userdata** – any object you would like to access in the callback

**is_collaborative**()

**Return type** int

**Returns** collaborative status for a playlist.

**Note:** A playlist in collaborative state can be modifed by all users, not only the user owning the list.

**is_loaded**()

**Return type** int

**Returns** whether this playlist has been loaded by the client

**name**()

**Return type** string

**Returns** the name of the playlist.

**num_subscribers**()

**Return type** int

**Returns** The number of subscribers of this playlist

**owner**()

**Return type** *spotify.User*

**Returns** the owner of the playlist

**rename**(*name*)

**Parameters** **name** (unicode) – the new name

**remove_callback**(*callback*[, *userdata*])
Removes the corresponding callback, userdata couple.

**remove_tracks**(*tracks*)

**Parameters** **tracks** (list of int) – A list of track positions to be removed from the playlist.

**subscribers**()

**Return type** list of unicode

**Returns** a list of canonical names of subscribers of this playlist.

---

**Note:** The count returned for this function may be less than those returned by *num_subscribers()*. Spotify does not track each user subscribed to a playlist for playlist with many (>500) subscribers.

---

**track_create_time**(*index*)

**Parameters** **index** (int) – index of the track in the playlist

**Return type** int

**Returns** number of seconds after Unix epoch the track was added to the playlist

**type**()
Returns 'playlist'

**update_subscribers**()
Ask library to update the subscription count for a playlist.

When the subscription info has been fetched from the Spotify backend the *manager.SpotifyPlaylistManager.subscribers_changed()* callback will be invoked. In that callback use *num_subscribers()* and/or *subscribers()* to get information about the subscribers. You can call those two functions anytime you want but the information might not be up to date in such cases

**get_offline_status**()

**Return type** int

**Returns**

- 0 if not offline enabled
- 1 if synchronized to local storage
- 2 if currently downloading
- 3 if queued for download

**set_offline_mode**(*offline*)

**Parameters** **offline** (bool) – If the playlist should be synced for offline playback.

**get_offline_download_completed**()

**Return type** int

**Returns** percentage complete, 0 if not downloading

#### The `PlaylistFolder` class

**class** `spotify.`**`PlaylistFolder`**
An entry in a playlist container that is not a playlist (often folder boundaries).

**`id`**()
if type is `'folder_start'`, returns the id of the folder, else returns 0.

**`is_loaded`**()
returns `True` when the container it belongs to is loaded.

**`name`**()
if type is `'folder_start'`, returns the name of the folder, else returns an empty string.

**`type`**()
returns `'folder_start'`, `'folder_end'` or `'placeholder'`.

### 2.4.13 Playlist containers

#### The `PlaylistContainer` class

The playlist container contains all the playlists attached to a session. It is a list of *`Playlist`* and *`PlaylistFolder`* objects.

**class** `spotify.`**`PlaylistContainer`**

**`add_new_playlist`**(*name*)

**Parameters** **`name`** (`unicode` or ascii `str`) – name of the new playlist

Add a new playlist to the container.

**`add_loaded_callback`**(*callback*[, *userdata*])

**Parameters**

- **`callback`** – signature: (*`PlaylistContainer`* pc, `Object` userdata).
- **`userdata`** – any object you would like to access in the callback.

The callback will be called when all metadata in the playlist container has finished loading.

**`add_playlist_added_callback`**(*callback*[, *userdata*])

**Parameters**

- **`callback`** – signature: (*`PlaylistContainer`* pc, *`Playlist`*: p, `int` position, `Object` userdata).
- **`userdata`** – any object you would like to access in the callback.

The callback will be called when a playlist is added to the playlist container.

**`add_playlist_moved_callback`**(*callback*[, *userdata*])

**Parameters**

- **`callback`** – signature: (*`PlaylistContainer`* pc, *`Playlist`* p, `int` position, `int` new_position, `Object` userdata).
- **`userdata`** – any object you would like to access in the callback.

The callback will be called when a playlist is moved from *position* to *new_position*.

**add_playlist_removed_callback**(*callback*[, *userdata*])

**Parameters**

- **callback** – signature: (*PlaylistContainer* pc, *Playlist* p, int position, Object userdata).
- **userdata** – any object you would like to access in the callback.

The callback will be called when a playlist is removed from the container.

**remove_playlist**(*index*)

**Parameters** **index** (int) – index of the playlist to remove

Removes a playlist from the container.

### 2.4.14 Users

**class** spotify.**User**
User objects

**is_loaded**()

**Return type** bool

**Returns** whether the user is loaded.

**canonical_name**()

**Return type** unicode

**Returns** the canonical name of the user.

**display_name**()

**Return type** unicode

**Returns** the display name of the user, or the canonical name if not loaded.

### 2.4.15 Toplists

**class** spotify.**ToplistBrowser**(*type*, *region*[, *callback*, *userdata*])
A ToplistBrowser object.

The browser is a sequence of *spotify.Album*, *spotify.Artist* or *spotify.Track* depending on the *type* argument given when creating the object.

**Parameters**

- **type** – one of 'albums', 'artists' or 'tracks'.
- **region** – one of:
  - 'all' for global toplists
  - a two letters country code (e.g 'SE', 'FR')
  - 'current' for the current logged in user's toplists
  - a *spotify.User* object to see this user's toplists
- **callback** – a function with signature (ToplistBrowser tb, userdata)
- **userdata** – any object you would like to access in the callback

**is_loaded**()

**Returns** `True` if the object's metadata is loaded.

**error**()

**Returns** None or an error message associated with the error.

### 2.4.16 Inbox

**Warning:** The Spotify inbox is currently not implemented in **pyspotify**.

### 2.4.17 API constants

#### User relation types

spotify.constant.**RELATION_TYPE_UNKNOWN**
Not yet known.

spotify.constant.**RELATION_TYPE_NONE**
No relation.

spotify.constant.**RELATION_TYPE_UNIDIRECTIONAL**
The currently logged in user is following this user.

spotify.constant.**RELATION_TYPE_BIDIRECTIONAL**
Bidirectional friendship established.

## 2.5 Changes

### 2.5.1 v1.12 (2015-01-16)

This version is compatible with *libspotify* version 12.

This is the first release in 18 months, including only minor improvements accumulated during the first eight months. Development focus has for a long time been on pyspotify 2 which is in beta and is fully usable.

**New features**

- Added methods for marking playlists for offline use: *get_offline_status()*, *set_offline_mode()*, and *get_offline_download_completed()*. Contributed by Alexandre Petitjean.

**Other changes**

- For developers: In *pyspotify* 1.7 we split out our mock version of libspotify to an independent project, *libmockspotify*. We've now reverted this, and a copy of the source code of libmockspotify 0.3.1 is included in the *pyspotify* repo. *libmockspotify* is deprecated, and from now on we maintain whatever *libspotify* mocks we need for *pyspotify* development in the *pyspotify* repo.

  To be able to run the tests, you still need to pass `--with-mock` to your `python setup.py ...` command to build pyspotify with mock support. Alternatively, you can use `fab test` to build *pyspotify* and run the tests.

- Jukebox example code cleaned up. It now has better error handling and you can select the audio sink to use. Changed to using argparse (requires Python >= 2.7). Please use the `-?` command line option to get more information.

### 2.5.2 v1.11 (2013-07-01)

This version is compatible with *libspotify* version 12.

**API changes**

- *spotify.audiosink.alsa.AlsaSink* now defaults to using `PCM_NONBLOCK`, which seems to help on audio underruns on Raspberry Pi. Thanks to Stefan Hoffmann.
- *spotify.audiosink.alsa.AlsaSink* now accepts a `period_size` kwarg, which defaults to 8192. Thanks to Stefan Hoffmann.
- *spotify.manager.SpotifyPlaylistManager.watch()* now listen to 10 additional types of playlist modification events, in total 13 types.

**New features**

- Add missing link types:
  - *spotify.Link.LINK_PROFILE*
  - *spotify.Link.LINK_STARRED*
  - *spotify.Link.LINK_LOCALTRACK*
  - *spotify.Link.LINK_IMAGE*
- Add *spotify.PlaylistContainer.remove_playlist()*.

**Code cleanups**

- Fixed formatting and style of entire C code base with exception of mock code.
- Fixed multiple Python reference count bugs found by using gcc-python-plugin and manual inspection.
- Fixed multiple libspotify reference count bugs by forcing all `*_FromSpotify` calls to specify if a reference needs to be added or not. Also audited for corresponding release calls.
- Converted all `malloc`/`free` calls to use the Python allocator. Also audited for missing `free` calls and marked outstanding issues with TODOs (only playlistcontainer callbacks remain as known issues).
- Simplified session callback handling by creating a common callback handler.
- Added `<Object>_SP_<OBJECT>` macros to hide access to the internal `sp_*` pointers.
- Switched to being strict about avoiding `Py_XDECREF` and making sure we check for `NULL`.
- Switched to proper `debug_printf` macro that always ends up in code but optimized out so we never have broken debug printing.
- Switched to using existing helpers inside the `<Object>_str` functions instead of duplicating code.
- Return boolean values in API calls that have `bool` return values.
- Simplify session creation code and add helpers for building session configs.
- Switch to using `PyErr_SetNone` for errors without a description.
- Use `PyArg_Parse*` type format strings for argument handling and refcount handling in most of the callback code.
- Ensure all encoded strings are freed after use.
- Reuse single callback function for "simple" playlist callbacks.

### 2.5.3 v1.10 (2012-12-12)

This version is compatible with *libspotify* version 12.

**API changes**

- Session method *connect* has been renamed to *Session.login()* for consistency.
- Add *spotify.Link.as_playlist()*. (Fixes: #82)

**New features**

- Split session creation and user login. The session must be created only once per process. After that, the application can connect and disconnect a user at will. (Fixes: #73)

  New methods: *Session.create()*, *Session.relogin()*, *Session.logout()*.

  The *Session* object now calls `manager.SpotifySessionManager._manager_logged_out()` when getting a *logged_out* event from *libspotify*, or *manager.SpotifySessionManager.logged_out* if the former is not defined.

### 2.5.4 v1.9.1 (2012-11-23)

This version, like 1.9, is compatible with *libspotify* version 12.

**Bug fixes**

- Fixed memory leak caused by the `from_*` methods on *spotify.Link*. The memory leak was introduced in v1.2 in June 2011, so all users should really update to this version.

### 2.5.5 v1.9 (2012-11-20)

This version, like 1.8, is compatible with *libspotify* version 12.

**New features**

- Added optional parameters `proxy`, `proxy_username`, `proxy_password` to *spotify.manager.SpotifySessionManager*, to allow libspotify to connect to the Spotify service through an authenticated proxy. Thanks to Dvad.
- Added methods *spotify.Track.is_autolinked()* and *spotify.Track.playable()* to support autolinked tracks. (Fixes: #74)

### 2.5.6 v1.8.1 (2012-11-04)

Minor fix to ease Debian package building:

- Replace `Makefile` used as a development convenience with Fabric and `fabfile.py`, so that debhelper doesn't think the project is built using make.

### 2.5.7 v1.8 (2012-11-04)

Updated to work with *libspotify* version 12:

- `sp_album_cover` requires a preferred image size. For now, it's hard coded to `SP_IMAGE_SIZE_NORMAL`. Thanks to olle. (Fixes: #66)

Since the above API change to libspotify isn't reflected in the pyspotify API yet, there are no API changes between v1.7.1 and 1.8.

### 2.5.8 v1.7.1 (2012-09-07)

Maintenance release to fix a login issue experienced by some users.

This version, like 1.7, is compatible with *libspotify* version 11.

**Bug fixes**

- Fix bogus comparision of pointers. This caused `SpotifyError: No credentials stored` for some users. (Fixes: #65)
- Fix wrong return type in the *`Playlist.remove_tracks()`* docs.
- Remove unused include of `pthread.h`, which caused warnings from lintian.

### 2.5.9 v1.7 (2012-04-22)

**API changes**

- This version works with *libspotify* version 11.
- Artist and album browsers are now created directly from the *`ArtistBrowser`* and *`AlbumBrowser`* class constructors. The *`Session.browse_artist()`* and *`Session.browse_album()`* methods still work but have been deprecated. Also, callbacks are optional for the two browsers.
- The audio sink wrappers have been cleaned up and moved to a new *`spotify.audiosink`* module. The interface is the same, but you'll need to update your imports if you previously used either `spotify.alsahelper.AlsaController` (renamed to *`spotify.audiosink.alsa.AlsaSink`*) or `spotify.osshelper.OssController` (renamed to *`spotify.audiosink.oss.OssSink`*).
- An *`ArtistBrowser`* object is now a list of *`Track`*, as it was written in the API documentation.
- `offset` is now optional in *`Link.from_track()`*.
- Remove undocumented/internal method `spotify.manager.SpotifySessionManager.wake()`. *`spotify.manager.SpotifySessionManager.notify_main_thread()`* does the same job. Make sure you haven't accidentally overrided `notify_main_thread()` in your `SpotifySessionManager` subclass.
- Remove undocumented/internal method `spotify.manager.SpotifySessionManager.terminate()`. Use *`spotify.manager.SpotifySessionManager.disconnect()`* instead.

**New features**

- Added method *`spotify.Playlist.owner()`*.
- Added methods *`spotify.Results.total_albums()`* and *`spotify.Results.total_artists()`*.
- Added methods *`spotify.ArtistBrowser.albums()`*, *`spotify.ArtistBrowser.similar_artists()`*, *`spotify.ArtistBrowser.tracks()`* and *`spotify.ArtistBrowser.tophit_tracks()`*.
- Added optional argument `type` for *`spotify.ArtistBrowser`*.
- pyspotify now registers a "null handler" for logging to the `spotify` logger. This means that any pyspotify code is free to log debug log to any logger matching `spotify.*`.

  By default the log statements will be swallowed by the null handler. An application developer using pyspotify may add an additional log handler which listens for log messages to the `spotify` logger, and thus get debug information from pyspotify.

- Multi-user credential retainment using `login_blob` from the *spotify.manager.SpotifySessionManager* and the *spotify.manager.SpotifySessionManager.credentials_blob_updated()* method.
- Added a `search_type` argument for searches.
- Added new method *spotify.Session.flush_caches()*.
- Add new *spotify.manager.SpotifySessionManager.music_delivery_safe()* callback that can safely use the Spotify API without segfaulting. A little overhead is caused by serializing and passing data to the main thread, so if you are not going to use the Spotify API from your callbacks, or you're doing your own synchronization, you can continue to use the non-safe methods with a bit less overhead.
- Bundled audio sink support:
  - A audio sink wrapper for PortAudio, *spotify.audiosink.portaudio.PortAudioSink*, have been contributed by Tommaso Barbugli. PortAudio is available on both Linux, Mac OS X, and Windows.
  - A audio sink wrapper for Gstreamer, *spotify.audiosink.gstreamer.GstreamerSink*, have been contributed by David Buchmann. Gstreamer is available on both Linux, Mac OS X, and Windows.
  - The audio sink selector code originally written by Tommaso Barbugli for the `jukebox.py` example app have been generalized and made available for other applications as *spotify.audiosink.import_audio_sink()*.
- Jukebox example:
  - The jukebox got support for playing entire playlists. Thanks to Bjørn Schjerve.
  - The jukebox now formats duration in minutes and seconds. Thanks to David Buchmann.

**Other changes**

- For developers: *pyspotify* now uses libmockspotify for its mocking needs. The mock module only contains Python bindings to the *libmockspotify* API. To be able to run the tests, you need to pass `--with-mock` to your `python setup.py ...` command to build pyspotify with mock support. Alternatively, you can use `make test` to run the tests.

### 2.5.10 v1.6.1 (2011-12-29)

Maintenance release to fix a segfault for some users of playlist folders.

**Bug fixes**

- Calling *spotify.PlaylistFolder.is_loaded()* would cause a double `free()`, and thus a segfault.

### 2.5.11 v1.6 (2011-11-29)

Updated to work with libspotify 10.1.16.

**API changes**

- `Session.is_available(track)` has been moved to *spotify.Track.availability()*, and returns a detailed availability status of the track.
- `Session.is_local(track)` is now *spotify.Track.is_local()*, and returns a boolean.
- Removed methods: `Session.get_friends`, `User.full_name`, `User.picture`, and `User.relation`, as they was removed from the libspotify API.

**New features**

- Add new method: *spotify.Playlist.track_create_time()*. Contributed by Benjamin Chapus.

### 2.5.12 v1.5 (2011-10-30)

Updated to work with libspotify 9.1.32.

**New features**

- Remember me: when setting the `remember_me` parameter to `True` at first login, it is possible to log in again without specifying the `username` and `password` attributes. Don't forget to logout in order to store the credentials.
- Add new method: *spotify.Playlist.subscribers()*
- Add new method: *spotify.Playlist.num_subscribers()*
- Add new method: *spotify.Playlist.update_subscribers()*
- Playlist folder boundaries are now recognized. Playlist containers contain both *spotify.Playlist* and *spotify.PlaylistFolder* objects. Both classes got a `type()` method, which returns the string `playlist`, `folder_start`, `folder_end`, or `placeholder`.

### 2.5.13 v1.4 (2011-09-24)

pyspotify v1.4 only works with libspotify v0.0.8. As libspotify v9.x has been released, this release of pyspotify will probably be the last release to work with libspotify v0.0.8.

**API changes**

- All callbacks with optional userdata are now called with the `userdata` parameter set to `None`, which means they are called with the same number of parameters every time.
- Messages from the Spotify service (`log` and `user`) have been converted to `unicode` objects.

**New features**

- Exceptions raised in callbacks are written to `stderr`
- *spotify.Session.search()* now accepts Unicode queries
- Add user handling: *spotify.User*
- Add toplist browsing: *spotify.ToplistBrowser*
- Add new method: *spotify.Playlist.rename()*
- Add new method: `spotify.Session.get_friends()`. Contributed by Francisco Jordano.
- Add new method: *spotify.Playlist.add_tracks()*. Contributed by Andreas Franzén.
- Add new method: *spotify.PlaylistContainer.add_new_playlist()*. Contributed by Andreas Franzén.

**Bug fixes**

- *spotify.manager.SpotifySessionManager.log_message()* callback used `str` in place of `unicode`
- *spotify.manager.SpotifySessionManager.message_to_user()* callback used `str` in place of `unicode`
- Argument errors were unchecked in *spotify.Session.search()*
- Fix crash on valid error at image creation. Fixed by Jamie Kirkpatrick.

- Keep compatibility with Python 2.5. Contributed by Jamie Kirkpatrick.
- Callbacks given at artist/album browser creation are now called by pyspotify. Fixed by Jamie Kirkpatrick.
- Fix exception when a `long` was returned from *spotify.manager.SpotifySessionManager.music_delivery()*

### 2.5.14 v1.3 (2011-06-11)

It has only been four days since the v1.2 release, but we would like to get the change from bytestrings to unicode objects released before more projects start using pyspotify, as this change is really backwards incompatible.

- All strings returned by pyspotify has been changed from UTF-8 encoded bytestrings to unicode objects.
- Track autolinking enabled for all playlists.
- Add `spotify.__version__` which exposes the current pyspotify version. The API version of the libspotify used is already available as `spotify.api_version`.

### 2.5.15 v1.2 (2011-06-07)

As of May 2011, Doug Winter transfered the maintenance of pyspotify to the Mopidy project. The Mopidy developers, which depends upon pyspotify, have during the first half of 2011 been maintaining a branch of pyspotify and related Debian packages, and done some unofficial releases. With this change, we hope to get pyspotify up to speed again, and make it a useful library both for Mopidy and other projects.

Lately, Antoine Pierlot-Garcin aka *bok* have been doing lots of work on pyspotify, both on catching up with the features of libspotify, fixing and extending the test suite, writing documentation, and on fixing bugs. A big thanks to him!

- Upgraded to libspotify 0.0.8
- New managers: *SpotifyPlaylistManager* and *SpotifyContainerManager* giving access to all the Playlist{,Container} callbacks
- Artist and Album browsing available. Contributed by Jamie Kirkpatrick.
- Added a method to stop the playback. Contributed by Jamie Kirkpatrick.
- Better error messages when not logged in and accessing user information
- Added support for a playlist of all starred tracks
- Get/Set starred status for a track
- Better memory management

### 2.5.16 v1.1+mopidy20110405 (2011-04-05)

Unofficial release by the Mopidy developers.

- Exposed the track_is_local() check function. Contributed by Jamie Kirkpatrick.
- Fixed incorrect calls to determine track availability/locality. Contributed by Jamie Kirkpatrick.

### 2.5.17 v1.1+mopidy20110331 (2011-03-31)

Unofficial release by the Mopidy developers.

- Pass error messages instead of error codes to session callbacks. Contributed by Antoine Pierlot-Garcin.

- Fixed an issue where all playlists would appar blank when starting up. Contributed by Jamie Kirkpatrick.
- Make new config flags default to 0. Thanks to Jamie Kirkpatrick and Antoine Pierlot-Garcin.

### 2.5.18 v1.1+mopidy20110330 (2011-03-30)

Unofficial release by the Mopidy developers.

- Further updates for libspotify 0.0.7 support. Contributed by Antoine Pierlot-Garcin.

### 2.5.19 v1.1+mopidy20110223 (2011-02-23)

Unofficial release by the Mopidy developers.

- Upgraded to libspotify 0.0.7. Contributed by Antoine Pierlot-Garcin.

### 2.5.20 v1.1+mopidy20110106 (2011-01-06)

Unofficial release by the Mopidy developers.

- Upgraded to libspotify 0.0.6
- Add OSS support for sound output
- Add is_collaborative to playlists
- Add tracks_added playlist callback
- Add removed and moved callbacks for playlists
- Add remove_tracks to playlists
- Add seek support by mapping sp_session_player_seek
- Add support to set preferred bitrate
- Fix a segfault. Thanks to Valentin David.

### 2.5.21 v1.1 (2010-04-25)

Last release by Doug Winter.

- Upgraded to libspotify 0.0.4
- See the git history for changes up to v1.1.

Contributors to pyspotify up until v1.1 includes:

- Doug Winter
- Stein Magnus Jodal
- Thomas Jost
- Ben Firshman
- Johannes Knutsen

## 2.6 Development

Development of pyspotify is coordinated through the IRC channel `#mopidy` at `irc.freenode.net` and through GitHub.

### 2.6.1 Code style

For C code, follow the style of the Python C API, as outlined in **PEP 7**.

For Python code, follow Mopidy's style.

### 2.6.2 Commit guidelines

- We follow the development process described at http://nvie.com/git-model.
- Keep commits small and on topic.
- If a commit looks too big you should be working in a feature branch not a single commit.
- Merge feature branches with `--no-ff` to keep track of the merge.

### 2.6.3 Running tests

To run tests, you need to install the `nose` test runner. On Ubuntu:

```
sudo apt-get install python-nose
```

Using Pip:

```
sudo pip install nose
```

Then you can build pyspotify and run `nosetests`:

```
rm -rf build/
python setup.py build --with-mock
PYTHONPATH=$(echo build/lib.linux-*/) nosetests
```

### 2.6.4 Continuous integration server

pyspotify uses the free service Travis CI for automatically running the test suite when code is pushed to GitHub. This works both for the main pyspotify repo, but also for any forks. This way, any contributions to pyspotify through GitHub will automatically be tested by Travis CI, and the build status will be visible in the GitHub pull request interface, making it easier to evaluate the quality of pull requests.

In addition, we run a Jenkins CI server at http://ci.mopidy.com/ that runs all test on multiple platforms (Ubuntu, OS X, x86, arm) for every commit we push to the `develop` branch in the main pyspotify repo on GitHub. Thus, new code isn't tested by Jenkins before it is merged into the `develop` branch, which is a bit late, but good enough to get broad testing before new code is released.

In addition to running tests, the Jenkins CI server also gathers coverage statistics and uses pylint to check for errors and possible improvements in our code. So, if you're out of work, the code coverage and pylint data at the CI server should give you a place to start.

### 2.6.5 Writing documentation

To write documentation, we use Sphinx. See their site for lots of documentation on how to use Sphinx. To generate HTML or LaTeX from the documentation files, you need some additional dependencies.

You can install them through Debian/Ubuntu package management:

```
sudo apt-get install python-sphinx
```

Then, to generate docs:

```
cd docs/
make        # For help on available targets
make html   # To generate HTML docs
```

The documentation at http://pyspotify.mopidy.com/ is automatically updated when a documentation update is pushed to `mopidy/pyspotify` at GitHub.

### 2.6.6 Creating releases

1. Update changelog and commit it.
2. Merge the release branch (`v1.x/develop` in the example) into `v1.x/master`:

```
git checkout v1.x/master
git merge --no-ff -m "Release v1.12" v1.x/develop
```

3. Tag the release:

```
git tag -a -m "Release v1.12" v1.12
```

4. Push to GitHub:

```
git push
git push --tags
```

5. Build package and upload to PyPI:

```
git clean -fdx
python setup.py sdist upload
```

6. Spread the word.

## 2.7 Authors

Contributors to pyspotify in the order of appearance:

- Doug Winter <doug@isotoma.com>
- Stein Magnus Jodal <stein.magnus@jodal.no>
- Thomas Jost <thomas.jost@gmail.com>
- Ben Firshman <ben@firshman.co.uk>
- Johannes Knutsen <johannes@knutseninfo.no>
- Antoine Pierlot-Garcin <antoine@bokbox.com>
- Jamie Kirkpatrick <jkp@kirkconsulting.co.uk>

- Francisco Jordano <arcturus@ardeenelinfierno.com>
- Andreas Franzén <andreas.franzen@osynlig.se>
- Morten Berg <mortenberg80@gmail.com>
- Benjamin Chapus <xben@free.fr>
- Joel Barciauskas <jbarciauskas@bluestatedigital.com>
- David Buchmann <david.buchmann@gmail.com>
- Bjørn Schjerve <bischjer@gmail.com>
- Tommaso Barbugli <tbarbugli@gmail.com>
- Kristian Klette <klette@klette.us>
- eoinmcc <eoinmcc@gmail.com>
- David C <dav@dav.com>
- Karl Wettin <karl.wettin@gmail.com>
- Stefan Hoffmann <stefan991@gmail.com>
- Andreas Franzen <andreas.franzen@osynlig.se>
- Nick Steel <kingosticks@gmail.com>
- Thomas Adamcik <thomas@adamcik.no>
- Alexandre Petitjean <alpetitjean@gmail.com>
- Jean-Baptiste Mestelan <mestelan@gmail.com>
- Markus Kaiserswerth <github@sensun.org>
- Lasse Bigum <lasse@bigum.org>
- Silvan Jegen <s.jegen@gmail.com>

## 2.8 Licenses

For a list of contributors, see Authors. For details on who have contributed what, please refer to our git repository.

### 2.8.1 Source code license

### 2.8.2 Documentation license



### 2.8.3 libspotify disclaimer

## S

## Symbols



## A



## B



## C

## N



## O



## P



## Q



## R



## S



## T

## U



## W



## Y