python-discid Documentation

Release 1.0.3

Johannes Dewender

October 06, 2013

CONTENTS

python-discid is a Python binding of libdiscid by MusicBrainz.

The main purpose is the calculation of an identifier for audio discs (Disc ID) to use for the MusicBrainz database. Additionally the disc MCN and track ISRCs can be extracted.

This module is released under the GNU Lesser General Public License Version 3 or later (LGPL3+) and the code repository and the bug tracker are at GitHub.

If you are interested in a binding for the MusicBrainz Web Service, you might be interested in python-musicbrainz-ngs.

CHAPTER

ONE

CONTENTS

1.1 Download and Installation

1.1.1 Dependencies

python-discid works with Python 2 >= 2.6, or Python 3 >= 3.1.

The module `discid` cannot be imported without Libdiscid >= 0.2.2 installed. If you want to use it as optional dependency, import the module only when needed or catch the `OSError` when libdiscid is not found.

1.1.2 Package Repositories (Linux)

If you are using Linux, you might find **python-discid** in a repository used by your package manager.

These packages are known:

- Arch Linux: AUR (Arch Python 2, Arch Python 3)
- openSUSE: software.opensuse.org (SuSE Python 2, SuSE Python 3)
- Ubuntu: PPA (musicbrainz-stable and musicbrainz-daily)

Your package manager will also handle the *libdiscid* dependency automatically.

1.1.3 PyPI

The next-best option is to load the Package from pypi with pip:

```
pip install discid
```

You still have to install Libdiscid.

1.1.4 Source Code

The code is available from GitHub as zip and tar.gz.

You can always get the latest code with **git**:

```
git clone https://github.com/JonnyJD/python-discid.git
```

Installation

You can use **python-discid** already when you put the folder `discid` in the same location you run your script from or somewhere in your `PYTHONPATH`.

System-wide installation is done with:

```
python setup.py install
```

You can test your setup (including Libdiscid) with:

```
python setup.py test
```

1.1.5 Libdiscid

If you don't have a package manager that takes care of the *Libdiscid* dependency, you have to download it manually.

You can find several builds and the source at http://musicbrainz.org/doc/libdiscid.

If no build for your platform is available, you have to build from source and install with:

```
cmake .
make
make install
```

If the last step doesn't work for you, you might have to place the files `discid.DLL`, `libdiscid.*.dylib` or `libdiscid.so.*` (depending on your platform) in the same directory as you start your script from or somewhere in your `PATH`.

1.2 Usage

1.2.1 Basic

The basic use case is:

```
# this will load libdiscid
import discid

print("device: %s" % discid.get_default_device())
disc = discid.read()        # reads from default device
print("id: %s" % disc.id)
print("submission url:\n%s" % disc.submission_url)
```

You can also set the device explicitely:

```
device = discid.get_default_device()
disc = discid.read(device)
id = disc.id
```

You can use other devices of course: see `read()`.

When anything goes wrong reading from the device, `DiscError` is raised.

1.2.2 Advanced

`discid` can do more than just provide the MusicBrainz Disc ID. You can get details about the tracks an fetch additional features like the ISRCs and the MCN, which is basically the EAN/UPC of the disc:

```
disc = discid.read(features=["mcn", "isrc"])
print("mcn: %s" % disc.mcn)
for track in disc.tracks:
    print("{num:>2}: {isrc:13}".format(num=track.number, isrc=track.isrc))
```

1.2.3 Without Disc Access

When you just want to generate disc IDs and you have the necessary data laying around, you can use `put()`. You will need the numbers of the first track (should be 1), the number of the last audio track (cut off trailing data tracks), the total number of sectors for these tracks and the offset for every one of the tracks up to the last audio track.

An example for the TOC TqvKjMu7dMliSfmVEBtrL7sBSno-:

```
first = 1
last = 15
sectors = 258725
offsets = [150, 17510, ..., 235590]
disc = discid.put(first, last, sectors, offsets)
print("id: %s" % disc.id)
last_track = disc.tracks[disc.last_track_num - 1]
print("last track length: %s seconds" % last_track.seconds)
```

Note: The example disc has track 16 as a multimedia/data track. The sector count for the disc is the ending sector for track 15! Depending on how you get this number, you might need to substract 11400 (2:32 minutes) from your sector count. Make sure the last track length is correct!

See Also:

Disc ID Calculation for details on which numbers to choose.

1.2.4 Fetching Metadata

You can use python-musicbrainz-ngs to fetch metadata for your disc. The relevant function is `musicbrainzngs.get_releases_by_discid()`:

```
import discid
import musicbrainzngs

musicbrainzngs.set_useragent("python-discid-example", "0.1", "your@mail")

disc = discid.read()
try:
    result = musicbrainzngs.get_releases_by_discid(disc.id,
                                                   includes=["artists"])
except musicbrainzngs.ResponseError:
    print("disc not found or bad response")
else:
    if result.get("disc"):
        print("artist:\t%s" %
              result["disc"]["release-list"][0]["artist-credit-phrase"])
```

```
        print("title:\t%s" % result["disc"]["release-list"][0]["title"])
    elif result.get("cdstub"):
        print("artist:\t" % result["cdstub"]["artist"])
        print("title:\t" % result["cdstub"]["title"])
```

You can fetch much more data. See `musicbrainzngs` for details.

Note: Please submit your disc ID with `Disc.submission_url` when it isn't found at the MusicBrainz server.

1.3 discid API

Python binding of Libdiscid

Libdiscid is a library to calculate MusicBrainz Disc IDs. This module provides a python-like API for that functionality.

The user is expected to create a `Disc` object using `read()` or `put()` and extract the generated information.

Importing this module will open libdiscid at the same time and will raise `OSError` when libdiscid is not found.

1.3.1 Constants

At the module level there are these constants available:

`discid.`**`LIBDISCID_VERSION_STRING`**
: The version string of the loaded libdiscid in the form *libdiscid x.y.z*. For old versions the string is *libdiscid < 0.4.0*.

`discid.`**`FEATURES`**
: The features libdiscid supports for the platform as a list of strings. Some Functions can raise `NotImplementedError` when a feature is not available. Some features might not be implemented in this python module, see `FEATURES_IMPLEMENTED`.

`discid.`**`FEATURES_IMPLEMENTED = ['read', 'mcn', 'isrc']`**
: The features implemented in this python module as a list of strings. Some might not be available for your platform, see `FEATURES`.

1.3.2 Functions

These functions are used to create a `Disc` object.

`discid.`**`read`**(*device=None*, *features=*[ ])
: Reads the TOC from the device given as string and returns a `Disc` object.

  That string can be either of: `str`, `unicode` or `bytes`. However, it should in no case contain non-ASCII characters. If no device is given, a default, also given by `get_default_device()` is used.

  You can optionally add a subset of the features in `FEATURES` or the whole list to read more than just the TOC. In contrast to libdiscid, `read()` won't read any of the additional features by default.

  A `DiscError` exception is raised when the reading fails, and `NotImplementedError` when libdiscid doesn't support reading discs on the current platform.

`discid.`**`put`**(*first*, *last*, *disc_sectors*, *track_offsets*)
: Creates a TOC based on the information given and returns a `Disc` object.

Takes the *first* track and *last* **audio** track as `int`. *disc_sectors* is the end of the last audio track, normally the total sector count of the disc. *track_offsets* is a list of all audio track offsets.

Depending on how you get the total sector count, you might have to substract 11400 (2:32 min.) for discs with data tracks.

A `TOCError` exception is raised when illegal parameters are provided.

See Also:

Disc ID Calculation

You can get the device that is used as a default with

`discid.`**`get_default_device`**`()`
The default device to use for `read()` on this platform given as a `unicode` or `str` object.

1.3.3 Disc object

class `discid.`**`Disc`**
The class of the object returned by `read()` or `put()`.

`id`
This is the MusicBrainz Disc ID, a `unicode` or `str` object.

`freedb_id`
This is the FreeDB Disc ID (without category), a `unicode` or `str` object.

`submission_url`
Disc ID / TOC Submission URL for MusicBrainz

With this url you can submit the current TOC as a new MusicBrainz Disc ID. This is a `unicode` or `str` object.

`first_track_num`
Number of the first track

`last_track_num`
Number of the last **audio** track

`sectors`
Total length in sectors

`length`
This is an alias for `sectors`

`seconds`
Total length in seconds

`mcn`
This is the Media Catalogue Number (MCN/UPC/EAN)

It is set after the *"mcn"* feature was requested on a read and supported by the platform or `None`. If set, this is a `unicode` or `str` object.

`tracks`
A list of `Track` objects for this Disc.

1.3.4 Track object

class `discid.`**`Track`**(*disc*, *number*)
Track objects are part of the `Disc` class.

`number`
The track number

`offset`
The track offset

`sectors`
The track length in sectors

`length`
This is an alias for `sectors`

`seconds`
Track length in seconds

`isrc`
The International Standard Recording Code

This will be *None* when the *"isrc"* feature was not requested or not supported, otherwise this is a `unicode` or `str` object.

1.3.5 Exceptions

The discid module includes a custom exception to handle specific problems:

exception `discid.`**`DiscError`**
Bases: `exceptions.IOError`

`read()` will raise this exception when an error occured.

exception `discid.`**`TOCError`**
Bases: `exceptions.Exception`

`put()` will raise this exception when illegal paramaters are provided.

1.4 Changelog

1.4.1 Changes in 1.0.3 (2013-10-06):

- fix: #37 test_emptyness: Assertion disc->success failed

1.4.2 Changes in 1.0.2 (2013-06-27):

- revert code to version 1.0.0 (see #35)
- fix: #35 deprecation warning for DEFAULT_DEVICE shows always
- renamed a api documentation page, a redirect was created

1.4.3 Changes in 1.0.1 (2013-06-26):

- fix: #34 bring back `DEFAULT_DEVICE` as deprecated

1.4.4 Changes in 1.0.0 (2013-06-25):

- #30 `DEFAULT_DEVICE` is now `get_default_device()`
- #32 `Disc.submission_url` doesn't point to a redirect now
- fix: seconds are now rounded the same as on MB server (0.5->up)

1.4.5 Changes in 0.5.0 (2013-04-27):

- feature: #10 add `Disc.mcn` and `Track.isrc`
- feature: add `LIBDISCID_VERSION_STRING`
- feature: #28 add `Disc.seconds`, `Track.seconds` and aliases `Disc.length` and `Track.sectors`
- #22 move `read()` and `put()` to module level
- #25 provide a package *discid* rather than a module
- #29 changed parameters for `put()` to include extra *sectors* and add `TOCError`
- rename `DiscId` to `Disc`
- fix: #27 move track attributes to `Track`
- fix: #24 only have "real" tracks in the list(s) (0 not special)
- fix: #19 only read the requested features from disc (sparse)
- fix: #26 remove `DiscId.webservice_url` (deprecated)
- fix: detect the version of libdiscid 0.3.0 also in lib64 installations

1.4.6 Changes in 0.4.0 (2013-04-09):

- feature: added `FEATURES_IMPLEMENTED`, `DiscId.track_lengths`, `DiscId.webservice_url` and `DiscId.freedb_id`
- feature #18: disc access test suite
- fix #21: uninformative error on Windows

1.4.7 Changes in 0.3.0 (2013-03-11):

- feature #20: add `FEATURES` list
- feature: `DiscId.put()`, `DiscId.track_offsets`, `DiscId.sectors`, `DiscId.first_track_num`, `DiscId.last_track_num`
- fix #17: test fails on Mac OS X for default_device
- fix #16: prefer libdiscid in current directory
- fix #15: import can now raise `OSError`

- fix #14: find libdiscid in current folder (Linux/Unix)

1.4.8 Changes in 0.2.1 (2013-01-30):

- fix #9: test fails on Python 3.2 because of unicode literals

1.4.9 Changes in 0.2.0 (2013-01-30):

- API change from `DiscId.get_id()` to `DiscId.id`
- added `DEFAULT_DEVICE` as a module constant
- added `DiscId.submission_url`
- added an actual documentation and links to linux packages
- add tests and continuous integration configuration
- add changelog

1.4.10 Changes in 0.1.0 (2013-01-12):

- initial version with `DiscId.read()` and `DiscId.get_id()`

CHAPTER

TWO

RELATED TOOLS

There are other other bindings of libdiscid available. Please check Libdiscid.

In particular there is another Python binding named python-libdiscid. The main difference is, that *python-libdiscid* is released under the Expat license and uses Cython. This means that it needs to be compiled against libdiscid. *Python-discid* doesn't need compilation, as it uses `ctypes`.

If you want to use the disc ID created by *python-discid* to query MusicBrainz for metatdata, then you should use python-musicbrainz-ngs. See *Fetching Metadata* for using *discid* and *musicbrainzngs* together.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*