musicbrainzngs Documentation

Release 0.4

Alastair Porter et. al

October 08, 2013

CONTENTS

musicbrainzngs implements Python bindings of the MusicBrainz Web Service (WS/2, NGS). With this library you can retrieve all kinds of music metadata from the MusicBrainz database.

musicbrainzngs is released under a simplified BSD style license.

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 Package manager

If you want the latest stable version of musicbrainzngs, the first place to check is your systems package manager. Being a relatively new library, you might not be able to find it packaged by your distribution and need to use one of the alternate installation methods.

1.1.2 PyPI

Musicbrainzngs is available on the Python Package Index. This makes installing it with pip as easy as:

```
pip install musicbrainzngs
```

1.1.3 Git

If you want the latest code or even feel like contributing, the code is available on GitHub.

You can easily clone the code with git:

```
git clone git://github.com/alastair/python-musicbrainz-ngs.git
```

Now you can start hacking on the code or install it system-wide:

```
python setup.py install
```

1.2 Usage

In general you need to set a useragent for your application, start searches to get to know corresponding MusicBrainz IDs and then retrieve information about these entities.

The data is returned in form of a `dict`.

If you also want to submit data, then you must authenticate as a MusicBrainz user.

This part of the documentation will give you usage examples. For an overview of available functions you can have a look at the *API*.

1.2.1 Identification

To access the MusicBrainz webservice through this library, you need to identify your application by setting the useragent header made in HTTP requests to one that is unique to your application.

To ease this, the convenience function `musicbrainzngs.set_useragent()` is provided which automatically sets the useragent based on information about the application name, version and contact information to the format recommended by MusicBrainz.

If a request is made without setting the useragent beforehand, a `musicbrainzngs.UsageError` will be raised.

1.2.2 Authentication

Certain calls to the webservice require user authentication prior to the call itself. The affected functions state this requirement in their documentation. The user and password used for authentication are the same as for the MusicBrainz website itself and can be set with the `musicbrainzngs.auth()` method. After calling this function, the credentials will be saved and automaticall used by all functions requiring them.

If a method requiring authentication is called without authenticating, a `musicbrainzngs.UsageError` will be raised.

If the credentials provided are wrong and the server returns a status code of 401, a `musicbrainzngs.AuthenticationError` will be raised.

1.2.3 Getting data

You can get MusicBrainz entities as a `dict` when retrieving them with some form of identifier. An example using `musicbrainzngs.get_artist_by_id()`:

```
artist_id = "c5c2ea1c-4bde-4f4d-bd0b-47b200bf99d6"
try:
    musicbrainzngs.get_artist_by_id(artist_id)
except WebServiceError as exc:
    print("Something went wrong with the request: %s" % exc)
else:
    artist = result["artist"]
    print("name:\t\t%s" % artist["name"])
    print("sort name:\t%s" % artist["sort-name"])
```

You can get more information about entities connected to the artist with adding *includes* and you filter releases and release_groups:

```
result = musicbrainzngs.get_artist_by_id(artist_id,
              includes=["release-groups"], release_type=["album", "ep"])
for release_group in result["artist"]["release-group-list"]:
    print("{title} ({type})".format(title=release_group["title"],
                                    type=release_group["type"]))
```

Tip: Compilations are also of primary type "album". You have to filter these out manually if you don't want them.

Note: You can only get at most 25 release groups using this method. If you want to fetch all release groups you will have to browse.

1.2.4 Searching

When you don't know the MusicBrainz IDs yet, you have to start a search. Using `musicbrainzngs.search_artist()`:

```
result = musicbrainzngs.search_artists(artist="xx", type="group",
                                       country="GB")
for artist in result['artist-list']:
    print(u"{id}: {name}".format(id=artist['id'], name=artist["name"]))
```

Tip: Musicbrainzngs returns unicode strings. It's up to you to make sure Python (2) doesn't try to convert these to ascii again. In the example we force a unicode literal for print. Python 3 works without fixes like these.

You can also use the query without specifying the search fields:

```
musicbrainzngs.search_release_groups("the clash london calling")
```

The query and the search fields can also be used at the same time.

1.2.5 Browsing

When you want to fetch a list of entities greater than 25, you have to use one of the browse functions. Not only can you specify a *limit* as high as 100, but you can also specify an *offset* to get the complete list in multiple requests.

An example would be using `musicbrainzngs.browse_release_groups()` to get all releases for a label:

```
label = "71247f6b-fd24-4a56-89a2-23512f006f0c"
limit = 100
offset = 0
releases = []
page = 0
while True:
    page += 1
    print("fetching page number %d.." % page)
    result = musicbrainzngs.browse_releases(label=label, includes=["labels"],
                        release_type=["album"], limit=limit, offset=offset)
    page_releases = result['release-list']
    releases += page_releases
    offset += limit
    if len(page_releases) < limit:
        break
print("")
for release in releases:
    for label_info in release['label-info-list']:
        catnum = label_info.get('catalog-number')
        if label_info['label']['id'] == label and catnum:
            print("{catnum:>17}: {date:10} {title}".format(catnum=catnum,
                        date=release['date'], title=release['title']))
print("\n%d releases on  %d pages" % (len(releases), page))
```

Tip: You should always try to filter in the query, when possible, rather than fetching everything and filtering afterwards. This will make your application faster since web service requests are throttled. In the example we filter by *release_type*.

1.2.6 Submitting

You can also submit data using musicbrainzngs. Please use `musicbrainzngs.set_hostname()` to set the host to test.musicbrainz.org when testing the submission part of your application.

Authentication is necessary to submit any data to MusicBrainz.

An example using `musicbrainzngs.submit_barcodes()` looks like this:

```
musicbrainzngs.set_hostname("test.musicbrainz.org")
musicbrainzngs.auth("test", "mb")

barcodes = {
    "174a5513-73d1-3c9d-a316-3c1c179e35f8": "5099749534728",
    "838952af-600d-3f51-84d5-941d15880400": "602517737280"
}
musicbrainzngs.submit_barcodes(barcodes)
```

See *Submitting* in the API for other possibilites.

1.3 API

This is a shallow python binding of the MusicBrainz web service so you should read Development/XML Web Service/Version 2 to understand how that web service works in general.

All requests that fetch data return the data in the form of a `dict`. Attributes and elements both map to keys in the dict. List entities are of type `list`.

This part will give an overview of available functions. Have a look at *Usage* for examples on how to use them.

1.3.1 General

`musicbrainzngs.`**`auth`**(*u*, *p*)
Set the username and password to be used in subsequent queries to the MusicBrainz XML API that require authentication.

`musicbrainzngs.`**`set_rate_limit`**(*limit_or_interval=1.0*, *new_requests=1*)
Sets the rate limiting behavior of the module. Must be invoked before the first Web service call. If the *limit_or_interval* parameter is set to False then rate limiting will be disabled. If it is a number then only a set number of requests (*new_requests*) will be made per given interval (*limit_or_interval*).

`musicbrainzngs.`**`set_useragent`**(*app*, *version*, *contact=None*)
Set the User-Agent to be used for requests to the MusicBrainz webservice. This must be set before requests are made.

`musicbrainzngs.`**`set_hostname`**(*new_hostname*)
Set the base hostname for MusicBrainz webservice requests. Defaults to 'musicbrainz.org'.

1.3.2 Getting Data

All of these functions will fetch a MusicBrainz entity or a list of entities as a dict. You can specify a list of *includes* to get more data and you can filter on *release_status* and *release_type*. See `musicbrainz.VALID_RELEASE_STATUSES` and `musicbrainz.VALID_RELEASE_TYPES`. The valid includes are listed for each function.

musicbrainzngs.**get_artist_by_id**(*id*, *includes*=[], *release_status*=[], *release_type*=[])
Get the artist with the MusicBrainz *id* as a dict with an 'artist' key.

Available includes: recordings, releases, release-groups, works, various-artists, discids, media, aliases, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation

musicbrainzngs.**get_label_by_id**(*id*, *includes*=[], *release_status*=[], *release_type*=[])
Get the label with the MusicBrainz *id* as a dict with a 'label' key.

Available includes: releases, discids, media, aliases, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation

musicbrainzngs.**get_recording_by_id**(*id*, *includes*=[], *release_status*=[], *release_type*=[])
Get the recording with the MusicBrainz *id* as a dict with a 'recording' key.

Available includes: artists, releases, discids, media, artist-credits, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation, aliases

musicbrainzngs.**get_recordings_by_echoprint**(*echoprint*, *includes*=[], *release_status*=[], *release_type*=[])
Search for recordings with an echoprint. The result is a dict with an 'echoprint' key, which again includes a 'recording-list'.

The preferred fingerprint method is AcoustID.

Available includes: artists, releases, discids, media, artist-credits, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation, aliases

musicbrainzngs.**get_recordings_by_puid**(*puid*, *includes*=[], *release_status*=[], *release_type*=[])
Search for recordings with a PUID. The result is a dict with a 'puid' key, which again includes a 'recording-list'.

The preferred fingerprint method is AcoustID.

Available includes: artists, releases, discids, media, artist-credits, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation, aliases

musicbrainzngs.**get_recordings_by_isrc**(*isrc*, *includes*=[], *release_status*=[], *release_type*=[])
Search for recordings with an ISRC. The result is a dict with an 'isrc' key, which again includes a 'recording-list'.

Available includes: artists, releases, discids, media, artist-credits, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation, aliases

musicbrainzngs.**get_release_group_by_id**(*id*, *includes*=[], *release_status*=[], *release_type*=[])
Get the release group with the MusicBrainz *id* as a dict with a 'release-group' key.

Available includes: artists, releases, discids, media, artist-credits, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation, aliases

musicbrainzngs.**get_release_by_id**(*id*, *includes*=[], *release_status*=[], *release_type*=[])
Get the release with the MusicBrainz *id* as a dict with a 'release' key.

Available includes: artists, labels, recordings, release-groups, media, artist-credits, discids, puids, echoprints, isrcs, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, recording-level-rels, work-level-rels, annotation, aliases

musicbrainzngs.**get_releases_by_discid**(*id*, *includes*=[], *release_status*=[], *release_type*=[])
Search for releases with a Disc ID.

The result is a dict with either a 'disc' or a 'cdstub' key. A 'disc' has a 'release-list' and a 'cdstub' key has direct 'artist' and 'title' keys.

Available includes: artists, labels, recordings, release-groups, media, artist-credits, discids, puids, echoprints, isrcs, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, recording-level-rels, work-level-rels, annotation, aliases

musicbrainzngs.**get_work_by_id**(*id*, *includes*=[])
Get the work with the MusicBrainz *id* as a dict with a 'work' key.

Available includes: artists, aliases, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation

musicbrainzngs.**get_works_by_iswc**(*iswc*, *includes*=[])
Search for works with an ISWC. The result is a dict with a'work-list'.

Available includes: artists, aliases, tags, user-tags, ratings, user-ratings, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels, annotation

musicbrainzngs.**get_collections**()
List the collections for the currently authenticated user as a dict with a 'collection-list' key.

musicbrainzngs.**get_releases_in_collection**(*collection*)
List the releases in a collection. Returns a dict with a 'collection' key, which again has a 'release-list'.

musicbrainzngs.musicbrainz.**VALID_RELEASE_TYPES = ['nat', 'album', 'single', 'ep', 'compilation', 'soundtrack', 's**
These can be used to filter whenever releases are includes or browsed

musicbrainzngs.musicbrainz.**VALID_RELEASE_STATUSES = ['official', 'promotion', 'bootleg', 'pseudo-release']**
These can be used to filter whenever releases or release-groups are involved

1.3.3 Searching

For all of these search functions you can use any of the allowed search fields as parameter names. The documentation of what these fields do is on Development/XML Web Service/Version 2/Search.

You can also set the *query* parameter to any lucene query you like. When you use any of the search fields as parameters, special characters are escaped in the *query*.

By default the elements are concatenated with spaces in between, so lucene essentially does a fuzzy search. That search might include results that don't match the complete query, though these will be ranked lower than the ones that do. If you want all query elements to match for all results, you have to set *strict=True*.

By default the web service returns 25 results per request and you can set a *limit* of up to 100. You have to use the *offset* parameter to set how many results you have already seen so the web service doesn't give you the same results again.

musicbrainzngs.**search_annotations**(*query=''*, *limit=None*, *offset=None*, *strict=False*, ***fields*)
Search for annotations and return a dict with an 'annotation-list' key.

Available search fields: entity, name, text, type

musicbrainzngs.**search_artists**(*query=''*, *limit=None*, *offset=None*, *strict=False*, ***fields*)
Search for artists and return a dict with an 'artist-list' key.

Available search fields: arid, artist, artistaccent, alias, begin, comment, country, end, ended, gender, ipi, sortname, tag, type

musicbrainzngs.**search_labels**(*query=''*, *limit=None*, *offset=None*, *strict=False*, ***fields*)
Search for labels and return a dict with a 'label-list' key.

Available search fields: alias, begin, code, comment, country, end, ended, ipi, label, labelaccent, laid, sortname, type, tag

musicbrainzngs.**search_recordings**(*query=''*, *limit=None*, *offset=None*, *strict=False*, ***fields*)
Search for recordings and return a dict with a 'recording-list' key.

Available search fields: arid, artist, artistname, creditname, comment, country, date, dur, format, isrc, number, position, primarytype, puid, qdur, recording, recordingaccent, reid, release, rgid, rid, secondarytype, status, tnum, tracks, tracksrelease, tag, type

musicbrainzngs.**search_release_groups**(*query=''*, *limit=None*, *offset=None*, *strict=False*, ***fields*)
Search for release groups and return a dict with a 'release-group-list' key.

Available search fields: arid, artist, artistname, comment, creditname, primarytype, rgid, releasegroup, releasegroupaccent, releases, release, reid, secondarytype, status, tag, type

musicbrainzngs.**search_releases**(*query=''*, *limit=None*, *offset=None*, *strict=False*, ***fields*)
Search for recordings and return a dict with a 'recording-list' key.

Available search fields: arid, artist, artistname, asin, barcode, creditname, catno, comment, country, creditname, date, discids, discidsmedium, format, laid, label, lang, mediums, primarytype, puid, reid, release, releaseaccent, rgid, script, secondarytype, status, tag, tracks, tracksmedium, type

1.3.4 Browsing

You can browse entitities of a certain type linked to one specific entity. That is you can browse all recordings by an artist, for example.

These functions can be used to to include more than the maximum of 25 linked entities returned by the functions in Getting Data. You can set a *limit* as high as 100. The default is still 25. Similar to the functions in Searching, you have to specify an *offset* to see the results you haven't seen yet.

You have to provide exactly one MusicBrainz ID to these functions.

musicbrainzngs.**browse_artists**(*recording=None*, *release=None*, *release_group=None*, *includes=[]*, *limit=None*, *offset=None*)
Get all artists linked to a recording, a release or a release group. You need to give one MusicBrainz ID.

Available includes: aliases, tags, ratings, user-tags, user-ratings

musicbrainzngs.**browse_labels**(*release=None*, *includes=[]*, *limit=None*, *offset=None*)
Get all labels linked to a relase. You need to give a MusicBrainz ID.

Available includes: aliases, tags, ratings, user-tags, user-ratings

musicbrainzngs.**browse_recordings**(*artist=None*, *release=None*, *includes=[]*, *limit=None*, *offset=None*)
Get all recordings linked to an artist or a release. You need to give one MusicBrainz ID.

Available includes: artist-credits, tags, ratings, user-tags, user-ratings

musicbrainzngs.**browse_release_groups**(*artist=None*, *release=None*, *release_type=[]*, *includes=[]*, *limit=None*, *offset=None*)
Get all release groups linked to an artist or a release. You need to give one MusicBrainz ID.

You can filter by `musicbrainz.VALID_RELEASE_TYPES`.

Available includes: artist-credits, tags, ratings, user-tags, user-ratings

musicbrainzngs.**browse_releases**(*artist=None*, *label=None*, *recording=None*, *release_group=None*, *release_status=[]*, *release_type=[]*, *includes=[]*, *limit=None*, *offset=None*)
Get all releases linked to an artist, a label, a recording or a release group. You need to give one MusicBrainz ID.

You can filter by `musicbrainz.VALID_RELEASE_TYPES` or `musicbrainz.VALID_RELEASE_STATUSES`.

Available includes: artist-credits, labels, recordings, release-groups, media, discids, artist-rels, label-rels, recording-rels, release-rels, release-group-rels, url-rels, work-rels

1.3.5 Submitting

These are the only functions that write to the MusicBrainz database. They take one or more dicts with multiple entities as keys, which take certain values or a list of values.

You have to use `auth()` before using any of these functions.

`musicbrainzngs.`**`submit_barcodes`**(*release_barcode*)
Submits a set of {release_id1: barcode, ...}

`musicbrainzngs.`**`submit_puids`**(*recording_puids*)
Submit PUIDs. Submits a set of {recording_id1: [puid1, ...], ...} or {recording_id1: puid, ...}.

`musicbrainzngs.`**`submit_echoprints`**(*recording_echoprints*)
Submit echoprints. Submits a set of {recording_id1: [echoprint1, ...], ...} or {recording_id1: echoprint, ...}.

`musicbrainzngs.`**`submit_isrcs`**(*recording_isrcs*)
Submit ISRCs. Submits a set of {recording-id1: [isrc1, ...], ...} or {recording_id1: isrc, ...}.

`musicbrainzngs.`**`submit_tags`**(*artist_tags={}*, *recording_tags={}*)
Submit user tags. Artist or recording parameters are of the form: {entity_id1: [tag1, ...], ...}

`musicbrainzngs.`**`submit_ratings`**(*artist_ratings={}*, *recording_ratings={}*)
Submit user ratings. Artist or recording parameters are of the form: {entity_id1: rating, ...}

`musicbrainzngs.`**`add_releases_to_collection`**(*collection*, *releases=[]*)
Add releases to a collection. Collection and releases should be identified by their MBIDs

`musicbrainzngs.`**`remove_releases_from_collection`**(*collection*, *releases=[]*)
Remove releases from a collection. Collection and releases should be identified by their MBIDs

1.3.6 Exceptions

These are the main exceptions that are raised by functions in musicbrainzngs. You might want to catch some of these at an appropriate point in your code.

Some of these might have subclasses that are not listed here.

class `musicbrainzngs.`**`MusicBrainzError`**
Base class for all exceptions related to MusicBrainz.

class `musicbrainzngs.`**`UsageError`**
Bases: `musicbrainzngs.musicbrainz.MusicBrainzError`

Error related to misuse of the module API.

class `musicbrainzngs.`**`WebServiceError`**(*message=None*, *cause=None*)
Bases: `musicbrainzngs.musicbrainz.MusicBrainzError`

Error related to MusicBrainz API requests.

class `musicbrainzngs.`**`AuthenticationError`**(*message=None*, *cause=None*)
Bases: `musicbrainzngs.musicbrainz.WebServiceError`

Received a HTTP 401 response while accessing a protected resource.

class musicbrainzngs.**NetworkError**(*message=None*, *cause=None*)
Bases: musicbrainzngs.musicbrainz.WebServiceError

Problem communicating with the MB server.

class musicbrainzngs.**ResponseError**(*message=None*, *cause=None*)
Bases: musicbrainzngs.musicbrainz.WebServiceError

Bad response sent by the MB server.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *search*

PYTHON MODULE INDEX

m

musicbrainzngs, ??