ListenBrainz Documentation

Release 0.1.0

MetaBrainz Foundation

Jun 22, 2018

Contents

This documentation is for:

- Developers *using our API* to submit and fetch listens
- System administrators managing a ListenBrainz installation
- Contributors to the ListenBrainz project

Review the *JSON* documentation if you plan to work with the ListenBrainz API. Most of the complexity comes from reading or constructing ListenBrainz JSON documents.

Contents:

CHAPTER 1

Set up a development environment

To contribute to the ListenBrainz project, you need a development environment. With your development environment, you can test your changes before submitting a patch back to the project. This guide helps you set up a development environment and run ListenBrainz locally on your workstation. By the end of this guide, you will have…

- Install system dependencies
- Register a MusicBrainz application
- Initialize development databases
- Run ListenBrainz locally on your workstation

Install dependencies

The `listenbrainz-server` is shipped in Docker containers. This helps create your development environment and later deploy the application. Therefore, to work on the project, you need to install Docker and use containers for building the project. Containers save you from installing all of this on your own workstation.

See the different installation instructions for your distribution below.

CentOS / RHEL

```
sudo yum install epel-release
sudo yum install docker docker-compose
```

Debian / Debian-based systems

```
sudo apt-get update && sudo apt-get install docker docker-compose
```

Fedora

```
sudo dnf install docker docker-compose
```

openSUSE

```
sudo zypper install docker docker-compose
```

Ubuntu / Ubuntu-based systems

```
sudo apt-get update && sudo apt-get install docker docker-compose
```

Register a MusicBrainz application

Next, you need to register your application and get a OAuth token from MusicBrainz. Using the OAuth token lets you sign into your development environment with your MusicBrainz account. Then, you can import your plays from somewhere else.

To register, visit the MusicBrainz applications page. There, look for the option to register your application. Fill out the form with these three options.

- **Name**: (any name you want and will recognize, e.g. `listenbrainz-server-devel`)
- **Type**: `Web Application`
- **Callback URL**: `http://localhost/login/musicbrainz/post`

After entering this information, you'll have a OAuth client ID and OAuth client secret. You'll use these for configuring ListenBrainz.

Update config.py

With your new client ID and secret, update the ListenBrainz configuration file. If this is your first time configuring ListenBrainz, copy the sample to a live configuration.

```
cp listenbrainz/config.py.sample listenbrainz/config.py
```

Next, open the file with your favorite text editor and look for this section.

```
# MusicBrainz OAuth
MUSICBRAINZ_CLIENT_ID = "CLIENT_ID"
MUSICBRAINZ_CLIENT_SECRET = "CLIENT_SECRET"
```

Update the strings with your client ID and secret. After doing this, your ListenBrainz development environment is able to authenticate and log in from your MusicBrainz login.

Also, in order for the Last.FM import to work, you should also update your Last.FM API key in this file. Look for the following section in the file.

```
# Lastfm API
LASTFM_API_URL = "https://ws.audioscrobbler.com/2.0/"
LASTFM_API_KEY = "USE_LASTFM_API_KEY"
```

Update the Last.FM API key with your key. After doing this, your ListenBrainz development environment is able to import your listens from Last.FM.

In case you don't have a Last.FM API key, you can get it from Last.FM API page.

You also need to update the `API_URL` field value to `http://localhost`.

Initialize ListenBrainz containers

Next, run the `develop.sh` script in the root of the repository. Using `docker-compose`, it creates multiple Docker containers for the different services and parts of the ListenBrainz server. This script starts Redis, PostgreSQL, InfluxDB, and web server containers. This also makes it easy to stop them all later.

The first time you run it, it downloads and creates the containers. But it's not finished yet.

```
./develop.sh
```

Initialize ListenBrainz databases

Your development environment needs some specific databases to work. Before proceeding, run these three commands to initialize the databases.

```
docker-compose -f docker/docker-compose.yml -p listenbrainz run --rm web python3␣
↪manage.py init_db --create-db
docker-compose -f docker/docker-compose.yml -p listenbrainz run --rm web python3␣
↪manage.py init_msb_db --create-db
docker-compose -f docker/docker-compose.yml -p listenbrainz run --rm web python3␣
↪manage.py init_influx
```

Your development environment is now ready. Now, let's actually see ListenBrainz load locally!

Run the magic script

Now that the databases are initialized, always start your development environment by executing the `develop.sh` script. Now, it will work as expected.

```
./develop.sh
```

You will see the containers eventually run again. Leave the script running to see your development environment in the browser. Later, shut it down by pressing CTRL^C. Once everything is running, visit your new site from your browser!

```
http://localhost
```

Now, you are all set to begin making changes and seeing them in real-time inside of your development environment!

Test your changes with unit tests

Unit tests are an important part of ListenBrainz. It helps make it easier for developers to test changes and help prevent easily avoidable mistakes later on. Before commiting new code or making a pull request, run the unit tests on your code.

```
./test.sh
```

This builds and runs the containers needed for the tests. Each container does not use volumes that link to data outside of the containers, so it does not interfere with production databases.

Also, run the **integration tests** for ListenBrainz.

```
./integration-test.sh
```

When the tests complete, you will see if your changes are valid or not. These tests are a helpful way to validate new changes without a lot of work.

CHAPTER 2

ListenBrainz API

The ListenBrainz server supports the following end-points for submitting and fetching listens. All endpoints have this root URL for our current production site [1].

- **API Root URL**: `https://api.listenbrainz.org`
- **Web Root URL**: `https://listenbrainz.org`

Note: All ListenBrainz services are only available on **HTTPS**!

Reference

API Calls

`POST /1/submit-listens`

Submit listens to the server. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header!

Listens should be submitted for tracks when the user has listened to half the track or 4 minutes of the track, whichever is lower. If the user hasn't listened to 4 minutes or half the track, it doesn't fully count as a listen and should not be submitted.

For complete details on the format of the JSON to be POSTed to this endpoint, see *JSON Documentation*.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – listen(s) accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

`GET /1/user/`(*user_name*)**`/listens`**

Get listens for user `user_name`. The format for the JSON returned is defined in our *JSON Documentation*.

[1] The beta endpoints (i.e. `beta.listenbrainz.org`) were deprecated in Fall 2017. If you were using this endpoint, please use the current, production endpoints instead.

If none of the optional arguments are given, this endpoint will return the `DEFAULT_ITEMS_PER_GET` most recent listens. The optional `max_ts` and `min_ts` UNIX epoch timestamps control at which point in time to start returning listens. You may specify max_ts or min_ts, but not both in one call. Listens are always returned in descending timestamp order.

Parameters

- **`max_ts`** – If you specify a `max_ts` timestamp, listens with listened_at less than (but not including) this value will be returned.
- **`min_ts`** – If you specify a `min_ts` timestamp, listens with listened_at greater than (but not including) this value will be returned.
- **`count`** – Optional, number of listens to return. Default: `DEFAULT_ITEMS_PER_GET` . Max: `MAX_ITEMS_PER_GET`

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

`GET /1/latest-import`

Get the timestamp of the newest listen submitted by a user in previous imports to ListenBrainz.

In order to get the timestamp for a user, make a GET request to this endpoint. The data returned will be JSON of the following format:

```
{
    'musicbrainz_id': the MusicBrainz ID of the user,
    'latest_import': the timestamp of the newest listen submitted in previous␣
↪imports.
                     Defaults to 0
}
```

Parameters

- **`user_name`** – the MusicBrainz ID of the user whose data is needed

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

`POST /1/latest-import`

Update the timestamp of the newest listen submitted by a user in an import to ListenBrainz.

In order to update the timestamp of a user, you'll have to provide a user token in the Authorization Header. User tokens can be found on https://listenbrainz.org/profile/.

The JSON that needs to be posted must contain a field named *ts* in the root with a valid unix timestamp. Example:

```
{
  'ts': 0
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – latest import timestamp updated
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Rate limiting

The ListenBrainz API is rate limited via the use of rate limiting headers that are sent as part of the HTTP response headers. Each call will include the following headers:

- **X-RateLimit-Limit**: Number of requests allowed in given time window
- **X-RateLimit-Remaining**: Number of requests remaining in current time window
- **X-RateLimit-Reset-In**: Number of seconds when current time window expires (*recommended*: this header is resilient against clients with incorrect clocks)
- **X-RateLimit-Reset**: UNIX epoch number of seconds (without timezone) when current time window expires [2]

Rate limiting is automatic and the client must use these headers to determine the rate to make API calls. If the client exceeds the number of requests allowed, the server will respond with error code `429: Too Many Requests`. Requests that provide the *Authorization* header with a valid user token may receive higher rate limits than those without valid user tokens.

Timestamps

All timestamps used in ListenBrainz are UNIX epoch timestamps in UTC. When submitting timestamps to us, please ensure that you have no timezone adjustments on your timestamps.

Constants

Constants that are releavant to using the API:

`listenbrainz.webserver.views.api_tools.`**MAX_LISTEN_SIZE = 10240**
Maximum overall listen size in bytes, to prevent egregious spamming.

`listenbrainz.webserver.views.api_tools.`**MAX_ITEMS_PER_GET = 100**
The maximum number of listens returned in a single GET request.

`listenbrainz.webserver.views.api_tools.`**DEFAULT_ITEMS_PER_GET = 25**
The default number of listens returned in a single GET request.

`listenbrainz.webserver.views.api_tools.`**MAX_TAGS_PER_LISTEN = 50**
The maximum number of tags per listen.

`listenbrainz.webserver.views.api_tools.`**MAX_TAG_SIZE = 64**
The maximum length of a tag

[2] Provided for compatibility with other APIs, but we still recommend using `X-RateLimit-Reset-In` wherever possible

CHAPTER 3

API Usage Examples

Note: These examples are written in Python version **3.6.3** and use requests version **2.18.4**.

Prerequisites

All the examples assume you have a development version of the ListenBrainz server set up on `localhost`. Remember to set `DEBUG` to `True` in the config. When in production, you can replace `localhost` with `api.listenbrainz.org` to use the real API. In order to use either one, you'll need a token. You can find it under `ROOT/profile/` when signed in, with `ROOT` being either `localhost` for the dev version or `listenbrainz.org` for the real API.

Caution: You should use the token from the API you're using. In production, change the token to one from `listenbrainz.org`.

Examples

Submitting Listens

See *JSON Documentation* for details on the format of the Track dictionaries.

If everything goes well, the json response should be `{"status": "ok"}`, and you should see a recent listen of "Never Gonna Give You Up" when you visit `ROOT/user/{your-user-name}`.

```
from time import time
import requests

ROOT = '127.0.0.1'

def submit_listen(listen_type, payload, token):
    """Submits listens for the track(s) in payload.

    Args:
        listen_type (str): either of 'single', 'import' or 'playing_now'
        payload: A list of Track dictionaries.
        token: the auth token of the user you're submitting listens for
```

```
    Returns:
         The json response if there's an OK status.

    Raises:
         An HTTPError if there's a failure.
         A ValueError is the JSON in the response is invalid.
    """

    response = requests.post(
        url="http://{0}/1/submit-listens".format(ROOT),
        json={
            "listen_type": listen_type,
            "payload": payload,
        },
        headers={
            "Authorization": "Token {0}".format(token)
        }
    )

    response.raise_for_status()

    return response.json()


if __name__ == "__main__":
    EXAMPLE_PAYLOAD = [
        {
            # An example track.
            "listened_at": int(time()),
            "track_metadata": {
                "additional_info": {
                    "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
                    "artist_mbids": [
                        "db92a151-1ac2-438b-bc43-b82e149ddd50"
                    ],
                    "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
                    "tags": ["you", "just", "got", "semi", "rick", "rolled"]
                },
                "artist_name": "Rick Astley",
                "track_name": "Never Gonna Give You Up",
                "release_name": "Whenever you need somebody"
            }
        }
    ]

    # Input token from the user and call submit listen
    token = input('Please enter your auth token: ')
    json_response = submit_listen(listen_type='single', payload=EXAMPLE_PAYLOAD,␣
↪token=token)

    print("Response was: {0}".format(json_response))
    print("Check your listens - there should be a Never Gonna Give You Up track,␣
↪played recently.")
```

Getting Listen History

See *JSON Documentation* for details on the format of the Track dictionaries.

If there's nothing in the listen history of your user, you can run `submit_listens` before this.

If there is some listen history, you should see a list of tracks like this:

```
import requests

ROOT = '127.0.0.1'
# The following token must be valid, but it doesn't have to be the token of the user␣
↪you're
# trying to get the listen history of.
TOKEN = 'YOUR_TOKEN_HERE'
AUTH_HEADER = {
    "Authorization": "Token {0}".format(TOKEN)
}


def get_listens(username, min_ts=None, max_ts=None, count=None):
    """Gets the listen history of a given user.

    Args:
        username: User to get listen history of.
        min_ts: History before this timestamp will not be returned.
                DO NOT USE WITH max_ts.
        max_ts: History after this timestamp will not be returned.
                DO NOT USE WITH min_ts.
        count: How many listens to return. If not specified,
               uses a default from the server.

    Returns:
        A list of listen info dictionaries if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON in the response is invalid.
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="http://{0}/1/user/{1}/listens".format(ROOT, username),
        params={
            "min_ts": min_ts,
            "max_ts": max_ts,
            "count": count,
        },
        # Note that an authorization header isn't compulsary for requests to get␣
↪listens
        # BUT requests with authorization headers are given relaxed rate limits by␣
↪ListenBrainz
        headers=AUTH_HEADER,
    )

    response.raise_for_status()

    return response.json()['payload']['listens']

if __name__ == "__main__":
```

```
    username = input('Please input the MusicBrainz ID of the user: ')
    listens = get_listens(username)

    for track in listens:
        print("Track: {0}, listened at {1}".format(track["track_metadata"]["track_name
↪"],
                                                   track["listened_at"]))
```

```
Track: Never Gonna Give You Up, listened at 1512040365
Track: Never Gonna Give You Up, listened at 1511977429
Track: Never Gonna Give You Up, listened at 1511968583
Track: Never Gonna Give You Up, listened at 1443521965
Track: Never Gonna Give You Up, listened at 42042042
```

Latest Import

Set and get the timestamp of the latest import into ListenBrainz.

Setting

```
from time import time
import requests

ROOT = '127.0.0.1'

def set_latest_import(timestamp, token):
    """Sets the time of the latest import.

    Args:
        timestamp: Unix epoch to set latest import to.
        token: the auth token of the user you're setting latest_import of

    Returns:
        The JSON response if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON response is invalid.
    """
    response = requests.post(
        url="http://{0}/1/latest-import".format(ROOT),
        json={
            "ts": timestamp
        },
        headers={
            "Authorization": "Token {0}".format(token),
        }
    )

    response.raise_for_status()

    return response.json()

if __name__ == "__main__":
```

```
    ts = int(time())
    token = input('Please enter your auth token: ')
    json_response = set_latest_import(ts, token)

    print("Response was: {0}".format(json_response))
    print("Set latest import time to {0}.".format(ts))
```

Getting

If your user has never imported before and the latest import has never been set by a script, then the server will return `0` by default. Run `set_latest_import` before this if you don't want to actually import any data.

```
import requests

ROOT = '127.0.0.1'
# The token can be any valid token.
TOKEN = 'YOUR_TOKEN_HERE'
AUTH_HEADER = {
    "Authorization": "Token {0}".format(TOKEN)
}

def get_latest_import(username):
    """Gets the latest import timestamp of a given user.

    Args:
        username: User to get latest import time of.

    Returns:
        A Unix timestamp if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON in the response is invalid.
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="http://{0}/1/latest-import".format(ROOT),
        params={
            "user_name": username,
        },
        headers=AUTH_HEADER,
    )

    response.raise_for_status()
    return response.json()["latest_import"]


if __name__ == "__main__":
    username = input('Please input the MusicBrainz ID of the user: ')
    timestamp = get_latest_import(username)

    print("User {0} last imported on {1}".format(USERNAME, timestamp))
```

You should see output like this:

```
User naiveaiguy last imported on 30 11 2017 at 12:23
```

CHAPTER 4

JSON Documentation

Note: Do not submit copyrighted information in these fields!

Submission JSON

To submit a listen via our API (see: *ListenBrainz API*), `POST` a JSON document to the `submit-listens` endpoint. Submit one of three types JSON documents:

- `single`: Submit single listen
  - Indicates user just finished listening to track
  - Only a single track may be specified in `payload`
- `playing_now`: Submit `playing_now` notification
  - Indicates that user just began listening to track
  - `payload` contains *only* one track
  - Submitting `playing_now` documents is optional
  - Timestamp must be omitted from a `playing_now` submission.
- `import`: Submit previously saved listens
  - `payload` may contain more than one listen, but complete document may not exceed `MAX_LISTEN_SIZE` bytes in size

The `listen_type` element defines different types of submissions. The element is placed at the top-most level of the JSON document. The only other required element is the `payload` element. This provides an array of listens – the payload may be one or mote listens (as designated by `listen_type`):

```
{
  "listen_type": "single",
  "payload": [
      --- listen data here ---
  ]
}
```

A sample listen payload may look like:

```
{
  "listened_at": 1443521965,
  "track_metadata": {
    "additional_info": {
      "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
      "artist_mbids": [
        "db92a151-1ac2-438b-bc43-b82e149ddd50"
      ],
      "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
      "tags": [ "you", "just", "got", "rick rolled!"]
    },
    "artist_name": "Rick Astley",
    "track_name": "Never Gonna Give You Up",
    "release_name": "Whenever you need somebody"
  }
}
```

A complete submit listen JSON document may look like:

```
{
  "listen_type": "single",
  "payload": [
    {
      "listened_at": 1443521965,
      "track_metadata": {
        "additional_info": {
          "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
          "artist_mbids": [
            "db92a151-1ac2-438b-bc43-b82e149ddd50"
          ],
          "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
          "tags": [ "you", "just", "got", "rick rolled!"]
        },
        "artist_name": "Rick Astley",
        "track_name": "Never Gonna Give You Up",
        "release_name": "Whenever you need somebody"
      }
    }
  ]
}
```

Fetching listen JSON

The JSON documents returned from our API look like the following:

```
{
  "payload": {
    "count": 25,
    "user_id": "-- the MusicBrainz ID of the user --",
    "listens": [
      "-- listen data here ---"
    ]
  }
}
```

The number of listens in the document are returned by the top-level `count` element. The `user_id` element contains the MusicBrainz ID of the user whose listens are being returned. The other element is the `listens` element. This is a list which contains the listen JSON elements (described above).

Payload JSON details

A minimal payload must include the `listened_at`, `track_metadata/artist_name` and `track_metadata/track_name` elements:

```
{
  "listened_at": 1443521965,
  "track_metadata": {
    "artist_name": "Rick Astley",
    "track_name": "Never Gonna Give You Up",
  }
}
```

`artist_name` and `track_name` elements must be simple strings, while the `listened_at` element must be an integer.

Add additional metadata you may have for a track to the `additional_info` element. Any additional information allows us to better correlate your listen data to existing MusicBrainz-based data. If you have MusicBrainz IDs availble, submit them!

The following optional elements may also be included in the `track_metadata` element:

element	description
`release_name`	the name of the release this recording was played from.

The following optional elements may also be included in the `additional_info` element. If you do not have the data for any of the following fields, omit the key entirely:

element	description
`artist_mbids`	A list of MusicBrainz Artist IDs, one or more Artist IDs may be included here. If you have a complete MusicBrainz artist credit that contains multiple Artist IDs, include them all in this list.
`release_group_mbid`	A MusicBrainz Release Group ID of the release group this recording was played from.
`release_mbid`	A MusicBrainz Release ID of the release this recording was played from.
`recording_mbid`	A MusicBrainz Recording ID of the recording that was played.
`track_mbid`	A MusicBrainz Track ID associated with the recording that was played.
`work_mbids`	A list of MusicBrainz Work IDs that may be associated with this recording.
`tracknumber`	The tracknumber of the recording. This first recording on a release is tracknumber 1.
`isrc`	The ISRC code associated with the recording.
`spotify_id`	The Spotify track URL associated with this recording. e.g.: http://open.spotify.com/track/1rrgWMXGCGHru5bIRxGFV0
`tags`	A list of user defined tags to be associated with this recording. These tags are similar to last.fm tags. For example, you have apply tags such as `punk`, `see-live`, `smelly`. You may submit up to `MAX_TAGS_PER_LISTEN` tags and each tag may be up to `MAX_TAG_SIZE` characters large.

At this point, we are not scrubbing any superflous elements that may be submitted via the `additional_info` element. We're open to see how people will make use of these unspecified fields and may decide to formally specify or scrub elements in the future.

CHAPTER 5

Last.FM Compatible API for ListenBrainz

There are two versions of the Last.FM API used by clients to submit data to Last.FM.

1. The latest Last.FM API
2. The AudioScrobbler API v1.2

ListenBrainz can understand requests sent to both these APIs and use their data to import listens submitted by clients like VLC and Spotify. Existing Last.FM clients can be pointed to the ListenBrainz proxy URL and they should submit listens to ListenBrainz instead of Last.FM.

Note: This information is also present on the ListenBrainz website.

AudioScrobbler API v1.2

Clients supporting the old version of the AudioScrobbler API (such as VLC and Spotify) can be configured to work with ListenBrainz by making the client point to `http://proxy.listenbrainz.org` and using your MusicBrainz ID as username and the LB Authorization Token as password.

If the software you are using doesn't support changing where the client submits info (like Spotify), you can edit your `/etc/hosts` file as follows:

```
138.201.169.196    post.audioscrobbler.com
138.201.169.196    post2.audioscrobbler.com
```

Last.FM API

These instructions are for setting up usage of the Last.FM API for Audacious client on Ubuntu. These steps can be modified for other clients as well.

For development

1. Install dependencies from here, then clone the repo and install audacious.
2. Before installing audacious-plugins, edit the file *audacious-plugins/src/scrobbler2/scrobbler.h* to update the following setting on line L28. This is required only because the local server does not have https support.:

```
`SCROBBLER_URL` to "http://ws.audioscrobbler.com/2.0/".
```

3. Compile and install the plugins from the instructions given here.
4. Edit the `/etc/hosts` file and add the following entry:

```
127.0.0.1 ws.audioscrobbler.com
```

5. Flush dns and restart network manager using:

```
$ sudo /etc/init.d/dns-clean start
$ sudo /etc/init.d/networking restart
```

6. Register an application on MusicBrainz with the following Callback URL `http://<HOSTURL>/login/musicbrainz/post` and update the received MusicBrainz Client ID and Client Secret in config.py of ListenBrainz. `HOSTURL` should be as per the settings of the server. Example: `localhost`
7. In Audacious, go to File > Settings > Plugins > Scrobbler2.0 and enable it. Now open its settings and then authenticate.
8. **When you get a URL from your application which look like this `http://last.fm/api/auth/?api_key=as3..234`**

    - If you are running a local server, then `HOSTURL` should be similar to "localhost:8080".
    - If you are not running the server, then `HOSTURL` should be "api.listenbrainz.org".

For users

1. Repeat all the above steps, except for steps 2 and 6.
2. For Step 8, choose the 2nd option for `HOSTURL`.

CHAPTER 6

ListenBrainz Data Dumps

ListenBrainz provides data dumps that you can import into your own server or use for other purposes. These data dumps are created regularly once a month. Each dump contains a number of different files. Depending on your use cases, you may or may not require all of them.

File Descriptions

A ListenBrainz data dump consists of two archives:

1. `listenbrainz-public-dump.tar.xz`
2. `listenbrainz-listens-dump.tar.xz`

listenbrainz-public-dump.tar.xz

This file contains information about ListenBrainz users and statistics derived from listens submitted to ListenBrainz calculated using Google BigQuery about users, artists, recordings etc.

listenbrainz-listens-dump.tar.xz

This is the core ListenBrainz data dump. This file contains all the listens submitted to ListenBrainz by its users.

Structure of the listens dump

The ListenBrainz listens dump consists of a number of files containing listens in JSON format, one document per line. Each user's listens are listed in one file in chronological order, with the latest listen first. The exact location of each user's listens is listed in the `index.json` file which is a JSON document containing a file name, an offset and size (in bytes) to uniquely identify the location and size of each user's listens.

The format of the `index.json` file is as follows:

```
{
    'user1': {
        'file_name': "file which contains user1's listens",
        'offset': "the byte at which user1's listens begin in the file",
        'size': "the size (in bytes) of the user's listens"
```

```
    }
}
```

Hence, if you wanted to extract a particular user's listens, you would look up that user in the `index.json` file, find the filename and offset from there, open the file and seek to that byte and read the bytes specified by the `index.json` files. Each line in the part of the file we read is a listen submitted for that particular user.

Here is some example code to explain the mentioned way of parsing the listens dump:

```
import json
import os

def read_user_listens(username):
    with open('index.json') as f:
        index = json.load(f)

        # get the filename, offset and size for user
        # from the index
        file_name = index[username]['file_name']
        offset = index[username]['offset']
        size = index[username]['size']

        # directory structure of the form "listens/%s/%s/%s.listens" % (uuid[0],␣
↪uuid[0:2], uuid)
        file_path = os.path.join('listens', file_name[0], file_name[0:2], '%s.listens
↪' % file_name)
        with open(file_path) as listen_file:
            listen_file.seek(offset)
            listens = listen_file.read(size)
            return map(json.loads, listens.split('\n'))


if __name__ == '__main__':
    username = input('Enter the name of the user: ')
    for listen in read_user_listens(username):
        print(json.dumps(listen, indent=4))
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

/1

```
GET /1/latest-import,8
GET /1/user/(user_name)/listens,7
POST /1/latest-import,8
POST /1/submit-listens,7
```

D

M