
ListenBrainz Documentation

Release 0.1.0

MetaBrainz Foundation

Nov 15, 2017

1	Set up a development environment	3
1.1	Install dependencies	3
1.2	Register a MusicBrainz application	4
1.3	Initialize ListenBrainz containers	4
1.4	Initialize ListenBrainz databases	5
1.5	Run the magic script	5
1.6	Test your changes with unit tests	5
2	ListenBrainz API	7
2.1	Reference	7
3	JSON Documentation	11
3.1	Submission JSON	11
3.2	Fetching listen JSON	12
3.3	Payload JSON details	13
4	Indices and tables	15
	HTTP Routing Table	17

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:

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.

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 committing 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 if your changes are valid or not. These tests are a helpful way to validate new changes without a lot of work.

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/user/import>) 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*

POST /1/latest-import

Get and update the timestamp of the newest listen submitted in previous imports to ListenBrainz.

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.

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*

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/user/import>.

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

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.

GET /1/latest-import

Get and update the timestamp of the newest listen submitted 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*

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/user/import>.

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

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.

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

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

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*

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 expires²

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.

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

Constants

Constants that are relevant 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

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 more 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:

```
{
  "count": 25,
  "payload": [
    "-- listen data here ---"
  ]
}
```

The number of listens in the document are returned by the top-level `count` element. The other element is the `payload` element. This 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 available, submit them!

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

element	description
<code>release_name</code>	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
<code>artist_mbids</code>	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.
<code>release_group</code>	A MusicBrainz Release Group ID of the release group this recording was played from.
<code>release_mbid</code>	A MusicBrainz Release ID of the release this recording was played from.
<code>recording_mbid</code>	A MusicBrainz Recording ID of the recording that was played.
<code>track_mbid</code>	A MusicBrainz Track ID associated with the recording that was played.
<code>work_mbids</code>	A list of MusicBrainz Work IDs that may be associated with this recording.
<code>tracknumber</code>	The tracknumber of the recording. This first recording on a release is tracknumber 1.
<code>isrc</code>	The ISRC code associated with the recording.
<code>spotify_id</code>	The Spotify track URL associated with this recording. e.g.: http://open.spotify.com/track/1rrgWMXGCGHru5bIRxGFV0
<code>tags</code>	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 <code>MAX_TAGS_PER_LISTEN</code> tags and each tag may be up to <code>MAX_TAG_SIZE</code> characters large.

At this point, we are not scrubbing any superfluous 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.

Indices and tables

- `genindex`
- `modindex`
- `search`

/1

GET /1/latest-import, 8
GET /1/user/(user_name)/listens, 9
POST /1/latest-import, 7
POST /1/submit-listens, 7

D

DEFAULT_ITEMS_PER_GET (in module listen-brainz.websserver.views.api_tools), 10

M

MAX_ITEMS_PER_GET (in module listen-brainz.websserver.views.api_tools), 10

MAX_LISTEN_SIZE (in module listen-brainz.websserver.views.api_tools), 10

MAX_TAG_SIZE (in module listen-brainz.websserver.views.api_tools), 10

MAX_TAGS_PER_LISTEN (in module listen-brainz.websserver.views.api_tools), 10