MusicBrainz API

The API discussed here is an interface to the MusicBrainz Database. It is aimed at developers of media players, CD rippers, taggers, and other applications requiring music metadata. The API's architecture follows the REST design principles. Interaction with the API is done using HTTP and all content is served in a simple but flexible format, in either XML or JSON. XML is the default format; to get a JSON response, you can either set the Accept header to `"application/json"` or add `fmt=json` to the query string (if both are set, `fmt=` takes precedence).

Contents

General FAQ

What can I do with the MusicBrainz API?
You can look up information about a particular MusicBrainz entity ("give me info about The Beatles"), browse the data to find entities connected to a particular entity ("show me all releases by The Beatles"), or search for entities matching a specific query ("show me all artists matching the query 'Beatles' so I can find the one I want and ask for more data").

Who can use the MusicBrainz API? Is it free?
Non-commercial use of this web service is free; please see our commercial plans or contact us if you would like to use this service commercially.

Do I need an API key?
Currently, no. But you must have a meaningful user-agent string.

Do I need to provide authentication?
Data submission, as well as requests that involve user information, require digest authentication.

Which formats can I get the data in?
The API was originally written to return XML, but nowadays it can also return JSON.

Is there any significant difference between the XML and JSON APIs?
For requesting data, the XML and JSON API are effectively equivalent. The XML API is the only one that allows submitting data to MusicBrainz (but keep in mind only ratings, tags, barcodes and ISRCs can be submitted via the API at all; for most data additions you should use the website instead).

Is there a limit to the number of requests I can make per second?
Yes. See our rate limiting rules.

This seems very complicated, can I see some examples?
Yes, we have an example page showcasing some queries and showing the returned format you can expect for each.

Are there language bindings for the API?
Yes, in many different languages. See our list of external libraries.

What should I do if I encounter unexpected behaviour not covered in these docs?
You can ask question in IRC or in the forums.

Check to see if a ticket has been filed in the bug tracker, and if not consider writing one.

What else should I know before I start using the API?
It'd probably be helpful to know:

- How MusicBrainz is structured
- What relationships are available

So you're on version 2 of the API then? What happened to version 1?
The version 1 of the API was designed with the data structure of the original (pre-2011) version of the MusicBrainz database in mind. It was deprecated in 2011 when we changed to our current data schema, and after running (without further updates) for several years to avoid breaking any tools using it, it was finally taken down in 2019.

Do you ever make breaking changes?
We try to avoid that, but sometimes we might need to do so. In those cases, they will be announced on our blog, so consider following that!

Application rate limiting and identification

All users of the API must ensure that each of their client applications never make more than ONE call per second. Making more than one call per second drives up the load on the servers and prevents others from using the MusicBrainz API. If you impact the server by making more than one call per second, your IP address may be blocked preventing all further access to MusicBrainz. Also, it is important that your application sets a proper User-Agent string in its HTTP request headers. For more details on both of these requirements, please see our rate limiting page.

Introduction

The API root URL is `https://musicbrainz.org/ws/2/`.

We have 13 resources on our API which represent core entities in our database:

```
area, artist, event, genre, instrument, label, place, recording, release, release-group, series, work, url
```

We also provide an API interface for the following non-core resources:

```
rating, tag, collection
```

And we allow you to perform lookups based on other unique identifiers with these resources:

```
discid, isrc, iswc
```

On each entity resource, you can perform three different GET requests:

```
lookup:   /<ENTITY_TYPE>/<MBID>?inc=<INC>
browse:   /<RESULT_ENTITY_TYPE>?<BROWSING_ENTITY_TYPE>=<MBID>&limit=<LIMIT>&offset=<OFFSET>&inc=<INC>
search:   /<ENTITY_TYPE>?query=<QUERY>&limit=<LIMIT>&offset=<OFFSET>
```

... except that browse and search are not implemented for genre entities at this time.

Note: Keep in mind only the search request is available without an MBID (or, in specific cases, a disc ID, ISRC or ISWC). If all you have is the name of an artist or album, for example, you'll need to make a search and pick the right result to get its MBID; only then will you able to use it in a lookup or browse request.

On the genre resource, we support an "all" sub-resource to fetch all genres, paginated, in alphabetical order:

```
all:      /genre/all?limit=<LIMIT>&offset=<OFFSET>
```

The `/genre/all` resource, in addition to supporting XML and JSON, can output all genre *names* as text by specifying `fmt=txt` or setting the Accept header to `"text/plain"`. The genre names are returned in alphabetical order and separated by newlines. (`limit` and `offset` are not supported for the `txt` format.)

Of these first three types of requests:

- Lookups, non-MBID lookups and browse requests are documented in following sections, and you can find examples on the dedicated examples page.
- Searches are more complex and are documented on the search documentation page.

Relax NG Schema

The file musicbrainz_mmd-2.0.rng is a Relax NG Schema for the XML version of this API. It can also be used to validate submissions you're trying to make through it.

Search

Searches are documented on the search documentation page.

Lookups

You can perform a lookup of an entity when you have the MBID for that entity:

```
lookup:   /<ENTITY_TYPE>/<MBID>?inc=<INC>
```

Note that unless you have provided an MBID in exactly the format listed, you are not performing a lookup request. If your URL includes something like artist=<MBID>, then please see the Browse section. If it includes query=<QUERY>, please see the Search page.

Subqueries

The inc= parameter allows you to request more information to be included about the entity. Any of the entities directly linked to the entity can be included.

```
/ws/2/area
/ws/2/artist            recordings, releases, release-groups, works
/ws/2/collection        user-collections (includes private collections, requires authentication)
/ws/2/event
/ws/2/genre
/ws/2/instrument
/ws/2/label             releases
/ws/2/place
/ws/2/recording         artists, releases, release-groups, isrcs, url-rels
/ws/2/release           artists, collections, labels, recordings, release-groups
/ws/2/release-group     artists, releases
```

```
/ws/2/series
/ws/2/work
/ws/2/url
```

In addition, Relationships are available for all entity types except genres via inc parameters.

To include more than one subquery in a single request, separate the arguments to `inc=` with a + (plus sign), like `inc=recordings+labels`.

All lookups which include release-groups allow a type= argument to filter the release-groups by a specific type. All lookups which include releases also allow the type= argument, and a status= argument is allowed.

Note that the number of linked entities returned is always limited to 25. If you need the remaining results, you will have to perform a browse request.

Linked entities are always ordered alphabetically by gid.

Note: In the XML API, when including `recordings` with a `release` entity, `tracks` listed in `media` have no `title` if that doesn't differ from recording's title, to reduce the size of the response.

inc= arguments which affect subqueries

Some additional inc= parameters are supported to specify how much of the data about the linked entities should be included:

```
- discids           include discids for all media in the releases
- media             include media for all releases, this includes the # of tracks on each medium and its forma
- isrcs             include isrcs for all recordings
- artist-credits    include artists credits for all releases and recordings
- various-artists   include only those releases where the artist appears on one of the tracks,
                    but not in the artist credit for the release itself (this is only valid on a
                    /ws/2/artist?inc=releases request).
```

Misc inc= arguments

```
- aliases                  include artist, label, area or work aliases; treat these as a set, as they are not
- annotation               include annotation
- tags, ratings            include tags and/or ratings for the entity
- user-tags, user-ratings  same as above, but only return the tags and/or ratings submitted by the specified
- genres, user-genres      include genres (tags in the genres list ): either all or the ones submitted by th
```

Requests with user-tags, user-genres and user-ratings require authentication. You can authenticate using HTTP Digest, use the same username and password used to access the main https://musicbrainz.org website.

The method to request genres mirrors that of tags: you can use inc=genres to get all the genres everyone has proposed for the entity, or inc=user-genres to get all the genres you have proposed yourself (or both!). For example, to get the genres for the release group for Nine Inch Nails' Year Zero you'd want **https://musicbrainz.org/ws/2/release-group/3bd76d40-7f0e-36b7-9348-91a33afee20e?inc=genres+user-genres** for the XML API and **https://musicbrainz.org/ws/2/release-group/3bd76d40-7f0e-36b7-9348-91a33afee20e?inc=genres+user-genres&fmt=json** for the JSON API.

Since genres are tags, all the genres are also served with inc=tags with all the other tags. As such, you can always use the tag endpoint if you would rather filter the tags by your own genre list rather than follow the MusicBrainz one, or if you want to also get other non-genre tags (maybe you want moods, or maybe you're really interested in finding artists who perform hip hop music and were murdered – we won't stop you!).

Relationships

You can request relationships with the appropriate includes:

```
- area-rels
- artist-rels
- event-rels
- genre-rels
- instrument-rels
- label-rels
- place-rels
- recording-rels
- release-rels
- release-group-rels
- series-rels
- url-rels
- work-rels
```

These will load relationships between the requested entity and the specific entity type. For example, if you request "work-rels" when looking up an artist, you'll get all the relationships between this artist and any works, and if you request "artist-rels" you'll get the relationships between this artist and any other artists. As such, keep in mind requesting "artist-rels" for an artist, "release-rels" for a release, etc. **will not** load all the relationships for the entity, just the ones to *other entities of the same type*.

In a release request, you might also be interested on relationships for *the recordings linked to the release*, or *the release group linked to the release*, or even for *the works linked to those recordings that are linked to the release* (for example, to find out who played guitar on a specific track, who wrote the lyrics for the song being performed, or whether the release group is part of a series). Similarly, for a recording request, you might want to get the relationships for any linked works. There are three additional includes for this:

```
- recording-level-rels
- release-group-level-rels (for releases only)
- work-level-rels
```

Keep in mind these just act as switches. If you request *work-level-rels* for a recording, you will still need to request *work-rels* (to get the relationship from the recording to the work in the first place) and any other relationship types you want to see (for example, *artist-rels* if you want to see work-artist relationships).

With relationships included, entities will have `<relation-list>` nodes for each target entity type (XML) or a `relations` object containing all relationships (JSON). You can see some examples the examples page.

Any attributes on a relationship will be on `<attribute-list>` nodes (XML) or in the `attributes` array (JSON). Relationship attributes always have a type ID, and some may have an associated value. Those can be found as attributes of the `<attribute>` element (XML) or by using the attribute name as a key for the `attribute-values` and `attribute-ids` elements (JSON). Sometimes the relationship attribute may also have a 'credited-as' name indicated by the user (for example, "guitar" could be credited as "Fender Stratocaster" or "violin" as "1st violin"). In an XML response this is yet another attribute on the XML `<attribute>` element element, while on a JSON response you'll need to look at the `attribute-credits` element.

Note that requesting "genre-rels" does not indicate the genres for a specific entity. For that, use "genres".

Non-MBID Lookups

Instead of MBIDs, you can also perform lookups using several other unique identifiers. However, because clashes sometimes occur, each of these lookups return a list of entities (there is no limit, all linked entities will be returned, paging is not supported).

discid

```
lookup: /discid/<discid>?inc=<INC>&toc=<TOC>
```

A `discid` lookup returns a list of associated releases, the 'inc=' arguments supported are identical to a lookup request for a release.

If there are no matching releases in MusicBrainz, but a matching CD stub exists, it will be returned. This is the default behaviour. If you do *not* want to see CD stubs, pass 'cdstubs=no.' CD stubs are contained within a <cdstub> element, and otherwise have the same form as a release. Note that CD stubs do not have artist credits, just artists.

If you provide the "toc" query parameter, and if the provided disc ID is not known by MusicBrainz, a fuzzy lookup will done to find matching MusicBrainz releases. Note that if CD stubs are found this will not happen. If you do want TOC fuzzy lookup, but not CD stub searching, specify "cdstubs=no". For example:

```
/ws/2/discid/I5l9cCSFccLKFEKS.7wqSZAorPU-?toc=1+12+267257+150+22767+41887+58317+72102+91375+104652+115380+132
```

Will look for the disc id first, and if it fails, will try to find tracklists that are within a similar distance to the one provided.

It's also possible to perform a fuzzy TOC search without a discid. Passing "-" (or any invalid placeholder) as the discid will cause it to be ignored if a valid TOC is present:

```
/ws/2/discid/-?toc=1+12+267257+150+22767+41887+58317+72102+91375+104652+115380+132165+143932+159870+174597
```

By default, fuzzy TOC searches only return mediums whose format is set to "CD." If you want to search all mediums regardless of format, add 'media-format=all' to the query:

```
/ws/2/discid/-?toc=1+12+267257+150+22767+41887+58317+72102+91375+104652+115380+132165+143932+159870+174597&me
```

The TOC consists of the following:

- First track (always 1)
- total number of tracks
- sector offset of the leadout (end of the disc)

- a list of sector offsets for each track, beginning with track 1 (generally 150 sectors)

isrc

```
lookup: /isrc/<isrc>?inc=<INC>
```

An `isrc` lookup returns a list of recordings, the 'inc=' arguments supported are identical to a lookup request for a recording.

iswc

```
lookup: /iswc/<iswc>?inc=<INC>
```

An `iswc` lookup returns a list of works, the 'inc=' arguments supported are identical to a lookup request for a work.

url (by text)

```
lookup: /ws/2/url?resource=<URL>[&resource=<URL>]...
```

The URL endpoint's 'resource' parameter is for providing a URL directly, rather than a URL MBID (for example, https://musicbrainz.org/ws/2/url?resource=http://www.madonna.com/ versus https://musicbrainz.org/ws/2/url/b663423b-9b54-4067-9674-fffaecf68851). This URL will need to be appropriately URL-escaped for inclusion as a query parameter; this means that URLs that include url-escaped parameters, or query parameters of their own, will need to be escaped a second time.

If the requested 'resource' does not exist, this endpoint will return 404 not found.

The 'resource' parameter can be specified multiple times (up to 100) in a single query:

```
/ws/2/url?resource=http://www.madonna.com/&resource=https://www.ladygaga.com/
```

The response will contain a url-list in this case, rather than a single top-level url, and any 'resource' that is not found will be skipped.

Browse

Browse requests are a direct lookup of all the entities directly linked to another entity ("directly linked" here meaning it does not include entities linked by a relationship). For example, you may want to see all releases on the label ubiktune:

```
/ws/2/release?label=47e718e1-7ee4-460c-b1cc-1192a841c6e5
```

Note that browse requests are not searches: in order to browse all the releases on the ubiktune label you will need to know the MBID of ubiktune.

The order of the results depends on what linked entity you are browsing by (however it will always be consistent). If you need to sort the entities, you will have to fetch all entities (see "Paging" below) and sort them yourself.

Linked entities

The following list shows which linked entities you can use in a browse request:

```
/ws/2/area              collection
/ws/2/artist            area, collection, recording, release, release-group, work
/ws/2/collection        area, artist, editor, event, label, place, recording, release, release-group, work
/ws/2/event             area, artist, collection, place
/ws/2/genre             collection
/ws/2/instrument        collection
/ws/2/label             area, collection, release
/ws/2/place             area, collection
/ws/2/recording         artist, collection, release, work
/ws/2/release           area, artist, collection, label, track, track_artist, recording, release-group
/ws/2/release-group     artist, collection, release
/ws/2/series            collection
/ws/2/work              artist, collection
```

As a special case, release also allows track_artist, which is intended to allow you to browse various artist appearances for an artist. It will return any release where the artist appears in the artist credit for a track, but NOT in the artist credit for the entire release (as those would already have been returned in a request with artist=<MBID>).

Release-groups can be filtered on type, and releases can be filtered on type and/or status. For example, if you want all the live bootleg releases by Metallica:

```
/ws/2/release?artist=65f4f0c5-ef9e-490c-aee3-909e7ae6b2ab&status=bootleg&type=live
```

Or all albums and EPs by Autechre:

```
/ws/2/release-group?artist=410c9baf-5469-44f6-9852-826524b80c61&type=album|ep
```

Paging

Browse requests are the only requests which support paging: any browse request supports an 'offset=' argument to get more results. Browse requests also support 'limit=': the default limit is 25, and you can increase that up to 100.

Special note for releases: To ensure requests can complete without timing out, we limit the number of releases returned such that the entire list contains no more than 500 tracks. (However, at least one full release is always returned, even if it has more than 500 tracks; we don't return "partial" releases.) This means that you may not get 100 releases per page if you set `limit=100`; in fact the number will vary per page depending on the size of the releases. In order to page through the results properly, increment `offset` by the number of releases you get from each response, rather than the (larger, fixed) `limit` size.

inc=

Just like with normal lookup requests, the server can be instructed to include more data about the entity using an 'inc=' argument. Supported values for inc= are:

```
/ws/2/area              aliases
/ws/2/artist            aliases
/ws/2/event             aliases
/ws/2/instrument        aliases
/ws/2/label             aliases
/ws/2/place             aliases
/ws/2/recording         artist-credits, isrcs
/ws/2/release           artist-credits, labels, recordings, release-groups, media, discids, isrcs (with record
/ws/2/release-group     artist-credits
/ws/2/series            aliases
/ws/2/work              aliases
/ws/2/area              aliases
/ws/2/url               (only relationship includes)
```

In addition to the inc= values listed above, all entities support:

```
annotation, tags, user-tags, genres, user-genres
```

All entities except area, place, release, and series support:

```
ratings, user-ratings
```

In addition, Relationships are available for all entity types via inc parameters, as with lookup requests.

Release (Group) Type and Status

Any query which includes release groups in the results can be filtered to only include release groups of a certain type. Any query which includes releases in the results can be filtered to only include releases of a certain type and/or status. Valid values are:

```
status     official, promotion, bootleg, pseudo-release, withdrawn, cancelled.
type       album, single, ep, broadcast, other (primary types) / audio drama, audiobook, compilation, demo, dj
```

See the release status documentation and the release group type documentation for info on what these values mean.

Additionally, browsing release groups via artist supports a special filter to show the same release groups as in the default website overview (excluding ones that contain only releases of status promotional, bootleg or pseudo-release). Valid values are:

```
release-group-status     website-default, all
```

Submitting data

You can use the API to submit certain kinds of data. Currently tags (including genres), ratings and ISRCs can be entered through the API.

Authentication

All POST requests require authentication. You should authenticate using HTTP Digest, using the same username and password you use to access the main https://musicbrainz.org website. The realm is "musicbrainz.org".

POST requests should always include a 'client' parameter in the URL (not the body). The value of 'client' should be the ID of the client software submitting data. This has to be the application's name and version number, not that of a client

library (client libraries should use HTTP's User-Agent header). The recommended format is "application-version", where version does not contain a - character.

User data

You can submit tags (including genres) and ratings through the **XML** API using POST requests. As described above, the client software needs to identify itself using the 'client=' parameter. In the following examples I will use 'example.app-0.4.7' as the client identifier; this is obviously a fictitious client.

tags

To submit tags (including genres), perform a POST request to the /ws/2/tag url, like this:

```
/ws/2/tag?client=example.app-0.4.7
```

The body of your request should be an XML formatted list of entities with <user-tag> elements.

An example request is reproduced below:

```
<metadata xmlns="http://musicbrainz.org/ns/mmd-2.0#">
    <artist-list>
        <artist id="a16d1433-ba89-4f72-a47b-a370add0bb56">
            <user-tag-list>
                <user-tag><name>female</name></user-tag>
                <user-tag><name>korean</name></user-tag>
                <user-tag><name>jpop</name></user-tag>
            </user-tag-list>
        </artist>
    </artist-list>
    <recording-list>
        <recording id="047ea202-b98d-46ae-97f7-0180a20ee5cf">
            <user-tag-list>
                <user-tag><name>noise</name></user-tag>
            </user-tag-list>
        </recording>
    </recording-list>
</metadata>
```

Because you're sending XML in the body of your POST request, make sure to also set the Content-Type to "application/xml; charset=utf-8".

Our tag functionality includes the ability to upvote and downvote tags (including genres). This terminology can be confusing. Whenever you tag something, you are in fact "upvoting" it (which will add 1 to the vote count for the tag). Downvoting is the inverse operation, and will subtract 1 from the tag's vote count. Tags that you downvote will be hidden from the UI for you (and if their total vote count drops to 0 or below, they'll be hidden for everyone). The "user-tag" elements can include a "vote" attribute that specifies what action you want to take:

```
<user-tag vote="upvote"><name>noise</name></user-tag>
<user-tag vote="downvote"><name>pop</name></user-tag>
<user-tag vote="withdraw"><name>rock</name></user-tag>
```

The "withdraw" vote will remove any upvote or downvote that you previously added (as if you had never voted).

If you do not supply any "vote" attributes in your request (as in the example above), then the list of tags you submit will be treated as upvotes and will completely replace all existing upvoted tags you have on that entity. (So, tags that are not included in the request will be withdrawn, if they were previously upvoted. Downvoted tags are left in place.) This is a legacy behavior that we maintain from before we had tag voting. Including any "vote" attribute in the request will cause it to only apply those votes that you specified.

ratings

To submit ratings, perform a POST request to the /ws/2/rating url, like this:

```
/ws/2/rating?client=example.app-0.4.7
```

The body of your request should be an XML formatted list of entities with <user-rating> elements.

An example request is reproduced below:

```
<metadata xmlns="http://musicbrainz.org/ns/mmd-2.0#">
    <artist-list>
        <artist id="455641ea-fff4-49f6-8fb4-49f961d8f1ad">
            <user-rating>100</user-rating>
        </artist>
    </artist-list>
    <recording-list>
```

```
        <recording id="c410a773-c6eb-4bc0-9df8-042fe6645c63">
            <user-rating>20</user-rating>
        </recording>
    </recording-list>
</metadata>
```

collections

To add or remove releases (for example) from your collection, perform a PUT or DELETE request to /ws/2/collection/<gid>/releases, respectively:

```
PUT /ws/2/collection/f4784850-3844-11e0-9e42-0800200c9a66/releases/455641ea-fff4-49f6-8fb4-49f961d8f1ad;c410
DELETE /ws/2/collection/f4784850-3844-11e0-9e42-0800200c9a66/releases/455641ea-fff4-49f6-8fb4-49f961d8f1ad;?
```

Other types of entities supported by collections can be submitted, too; just substitute "releases" in the URI with one of: areas, artists, events, labels, places, recordings, release-groups, or works, depending on the type of collection.

You may submit up to ~400 entities in a single request, separated by a semicolon (;), as the PUT example above shows. You are restricted to a maximum URI length of 16kb at the moment (which roughly equates to 400 gids).

To get the description of a collection, perform a lookup request with the collection MBID:

```
GET /ws/2/collection/4a0a2cd0-3b20-4093-bd99-92788045845e
```

To get the description and the summarized contents of a collection, perform a lookup request with the collection MBID and the appropriate entity subquery:

```
GET /ws/2/collection/4a0a2cd0-3b20-4093-bd99-92788045845e/areas
GET /ws/2/collection/f4784850-3844-11e0-9e42-0800200c9a66/releases
...
```

To get the contents of a collection, perform a browse request on the appropriate entity endpoint, using the collection MBID as a parameter:

```
GET /ws/2/area?collection=4a0a2cd0-3b20-4093-bd99-92788045845e
GET /ws/2/release?collection=f4784850-3844-11e0-9e42-0800200c9a66
...
```

To get a list of collections for a given user (including the number of entities in each collection), you can browse the collection endpoint by editor name:

```
GET /ws/2/collection?editor=rob
```

This will only return collections that rob has made public. If you wish to see private collections as an authenticated user, do:

```
GET /ws/2/collection?editor=rob&inc=user-collections
```

Barcode submission

Barcodes may be associated with releases by issuing an XML POST request to:

```
/ws/2/release/?client=example.app-0.4.7
```

The body of the request must be an XML document with a list of <releases>s in a <release-list>, and a single barcode in a <barcode> element for each release. For example:

```
<metadata xmlns="http://musicbrainz.org/ns/mmd-2.0#">
    <release-list>
        <release id="047ea202-b98d-46ae-97f7-0180a20ee5cf">
            <barcode>4050538793819</barcode>
        </release>
    </release-list>
</metadata>
```

Only GTIN (EAN/UPC) codes are accepted. Codes that have an incorrect check sum or a 2/5-digit add-on are refused. These should be manually added as annotation through the release editor instead.

Upon issuing this request MusicBrainz will create a single edit in the edit queue for applying these changes. These changes will *not* be automatically applied, though they will be applied if either no one votes against your changes, or once your changes expire.

ISRC submission

ISRCs may be associated with recordings by issuing an XML POST request to:

```
/ws/2/recording/?client=example.app-0.4.7
```

The body of the request must be an XML document with a list of <recording>s in a <recording-list>, and a list of <ISRC>s in a <isrc-list> to be associated with the recordings. For example:

```
<metadata xmlns="http://musicbrainz.org/ns/mmd-2.0#">
  <recording-list>
    <recording id="b9991644-7275-44db-bc43-fff6c6b4ce69">
      <isrc-list count="1">
        <isrc id="JPB600601201" />
      </isrc-list>
    </recording>
    <recording id="75c961c9-6e00-4861-9c9d-e6ca90d57342">
      <isrc-list count="1">
        <isrc id="JPB600523201" />
      </isrc-list>
    </recording>
  </recording-list>
</metadata>
```

Libraries

It can be accessed with our C/C++ library, libmusicbrainz.

Third party libraries:

- C#/Mono/.NET:
  - avatar29A/MusicBrainz
  - MetaBrainz.MusicBrainz (NuGet Package)
- Common Lisp - cl-musicbrainz
- Go
  - github.com/michiwend/gomusicbrainz
  - go.uploadedlobster.com/musicbrainzws2
- Haskell:
  - ClintAdams/MusicBrainz
  - ocharles/musicbrainz-data
- Java - musicbrainzws2-java
- JavaScript/Node.js:
  - musicbrainz-api
  - node-musicbrainz
- Objective-C - libmusicbrainz-objc
- Perl - WebService::MusicBrainz
- PHP:
  - lachlan-00/MusicBrainz, a fork of mikealmond/MusicBrainz
  - mikealmond/MusicBrainz, a fork of phpbrainz
  - PBXg33k/MusicBrainz
  - stephan-strate/php-music-brainz-api
- Python - python-musicbrainzngs
- Ruby:
  - dwo/musicbrainz-ruby
  - magnolia-fan/musicbrainz
- Rust - musicbrainz_rs: Crate (GitHub)