Deliverable D4.13

Release of tool for the automatic semantic description of music pieces

Grant agreement nr	688382
Project full title	Audio Commons: An Ecosystem for Creative Reuse of Audio Content
Project acronym	AudioCommons
Project duration	36 Months (February 2016 - January 2019)
Work package	WP4
Due date	31 January 2019 (M36)
Submission date	31 January 2019 (M36)
Report availability	Public (X), Confidential ( )
Deliverable type	Report ( ), Demonstrator ( ), Other (X)
Task leader	QMUL
Authors	Johan Pauwels
Document status	Draft ( ), Final (X)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Table of contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Executive Summary

As part of the Audio Commons Ecosystem, a number of tools have been developed for the automatic analysis of audio content without the need for human intervention. These tools are designed for extracting i) musical audio properties for music pieces and music samples, and ii) non-musical audio properties for any kind of sounds. Two prototypes of each of these tools have been released and evaluated during the course of the project.

This deliverable accompanies the release of the tool for automatic annotation of music pieces. This tool takes the form of a web service that can be easily integrated with the rest of the Audio Commons Ecosystem (ACE). It consists of two major parts: 1) an enriching API that provides supplemental audio-based descriptors, computed automatically when queried by id, 2) a search API that retrieves music pieces corresponding to specified descriptor values. In this document, we first describe how to use the APIs, and then give instructions on how to install a copy of the web service on a server. This web service also serves as an example to show how to integrate other potential similar analysis services into the ACE.

Through these APIs, the content in the AC Ecosystem becomes more discoverable, thereby fulfilling one of the main goals set out at the beginning of this project. We conclude with some guidelines for integration of the presented analysis service with the Semantic Mediator, developed in WP2 and other potential improvements.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

1 Introduction and Background

This tool represents the culmination of the work performed for task T4.3 “RTD on automatic methods for the semantic annotation of music pieces” and its associated evaluation in task T4.5. As part of the larger WP4, T4.3 and T4.5 focussed on the development of tools for the semantic annotation on musical pieces, in contrast to the tools for musical samples developed in T4.2 and T4.3 and for non-music audio in WP5.

On the road towards achieving this goal, two prototypes were developed and subsequently refined into this final release. Deliverable D4.3 described the implementation of several automatic annotation tools, followed by an evaluation of these tools in Deliverable D4.5. The main issue highlighted in this deliverable were that the tools were fragmented and weren’t easy to use. To address this point, the second prototype, described in Deliverable D4.8, unified these tools into a web interface that provides a single consistent (programming) interface that is easier to use for developers in the Audio Commons Ecosystem. To illustrate the benefits and potential applications of music descriptors available through a web interface, a music discovery and recommendation application aimed at music learners was created and presented in Deliverable D4.11. The searching functionality developed for this app was then generalised and extended into the search API that is part of the final release. In turn, this API will make similarly novel, but more powerful discovery applications possible that will improve the visibility of the music pieces within the Audio Commons Ecosystem.

The final tool, documented in this Deliverable, integrates parts of the work from T4.2 and is intended to be used in combination with the Semantic Mediator developed in WP2. In this document, Section 2 illustrates its usage alongside the Mediator, using an instance installed on QMUL servers. Section 3 then describes how to install another instance of the webservice. Section 4 presents some considerations about further integration with the Mediator, and conclusions are presented in Section 5.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 688382

2 Usage

The AC Analysis Service provides a web-accessible API that consists of two major parts:

1. an enrichment API that provides supplemental audio-based descriptors, computed automatically when queried by id; and
2. a search API that retrieves music pieces corresponding to specified descriptor values.

In this section, we will describe how to use both of them, using the live instance that is currently running on QMUL's server.

2.1 The Enrichment API

In order to get additional audio-based descriptors for music pieces obtained through the AC Mediator, the enrichment API can be called at `https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis` with the unique identifiers obtained through the Semantic Mediator. This can be done through a command-line tool like cURL or through any programming language. In this report, we will use Python to illustrate the workings of the API.

Suppose you have queried the AC Mediator with a search term and have stored the response:

```
import requests
response = requests.get('http://m2.audiocommons.org/api/audioclips/search',
                        params = {'pattern': 'bach',
                                  'limit': 3})
```

From the Mediator, you get a list of results aggregated from the different content providers, each with a unique identifier (hereafter called AC id) of the form `content-provider:provider-id`, where `content-provider` currently is one of:

- jamendo-tracks
- freesound-sounds
- europeana-res

The results list already contains various metadata, formatted according to the AC Ontology developed in WP2, but if you want to request complementary audio-based descriptors, you need their AC ids to pass on to the Enrichment API.

In Python, you can extract the AC ids from the Mediator response as follows:

```
ac_ids = [item['content']['@id'] for provider in response.json()['results']
                                 for item in provider['members']]
```

And then use them to call Enrichment API with the following syntax:

```
https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/<descriptor>?id=<ac-id>
```

where you fill in the exact descriptor you want to see returned.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

The first time a descriptor is requested for a particular music piece, it gets calculated on the server and stored in a caching database such that subsequent API calls will be much faster.

The following descriptors are currently supported:

- chords (with confidence measure)
- tempo
- beats
- global-key (with confidence measure)
- keys
- tuning
- Instruments

So you can get the tempo for the `ac_ids` returned by the Mediator as follows:

```
tempos = []
for ac_id in ac_ids:
      tempo_response = requests.get('http://audio-analysis.eecs.qmul.ac.uk/
                                     function/ac-analysis/tempo’,
                                    params={'id': ac_id})
      tempo_response.raise_for_status()
      tempos.append(tempo_response.json())
```

By default the format of the API return is plain JSON, but Linked Data can also be returned in the JSON-LD format by requesting the appropriate Content-Type `application/ld+json`.

The call corresponding to the one above would then be:

```
tempo_response = requests.get('http://audio-analysis.eecs.qmul.ac.uk/
                               function/ac-analysis/tempo',
                              params={'id': ac_id},
                              headers={'Content-Type': 'application/ld+json'})
```

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 688382

2.2 The Search API

2.2.1 The Link between the Enrichment and the Search API

With the Search API, the contents of the Audio Commons Ecosystem can be explored not just by textual queries made through the Mediator, but also by searching by musical content. The searching functionality relies on the caching database of descriptors that is created while using the Enrichment API. This means that the number of music pieces that are discoverable through the Search API automatically grows as the usage of the Enrichment API increases (including the addition of more content providers). Since it also means that music in the AC Ecosystem is not discoverable until it is indexed by the Enrichment API, a way to asynchronously call the Enrichment API has also been added to facilitate large scale indexation of AC content. Every endpoint in the Enrichment API has an asynchronous counterpart, obtained by replacing `function` in the URL with `async-function`, which needs to be accessed as a POST method. A call to get a music piece pre-analysed for tempo would look like the following:

```
requests.post('http://audio-analysis.eecs.qmul.ac.uk/
               async-function/ac-analysis/tempo',
              params={'id': ac_id})
```

Calling these asynchronous endpoints won't return the descriptor results (although it is possible to pass on a callback URL using the header `X-Callback-Url`). Instead, it will be queued such that many of these calls can be given concurrently without overloading the server. Whenever a call reaches the beginning of the queue, the same calculations as for the synchronous call will be performed (if the descriptors are not already calculated) and the result stored in the caching database for later usage and searching.

2.2.2 General Usage of the Search API

To perform a search, the endpoint at `https://audio-analysis.eecs.qmul.ac.uk/function/ac-search` needs to be called. Currently, searching by the following descriptors is supported:

- tempo
- tuning
- chords
- global-key

Most of the content of the Jamendo Licensing catalogue has been pre-analysed for these descriptors, using the mechanism described above.

Since a search can return multiple results, a way to page through the results is needed. Therefore the `ac-search` endpoint supports paging as follows:

```
https://audio-analysis.eecs.qmul.ac.uk/function/ac-search[/<number-of-results>[/
<offset>]]
```

which defaults to `/ac-search/1/0` if nothing is specified.

By default, searching will be done over all content providers in the AC Ecosystem (that have been pre-analysed using the enrichment API), but the searching scope can be limited to one or more

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

specific provider(s) by passing their names as a query string of the form `?providers=<provider1>[,<provider2]`, which needs to be appended to the end point as:

```
https://audio-analysis.eecs.qmul.ac.uk/function/ac-search[/<number-of-results>[/
<offset>]][?providers=<provider1>[,<provider2]]
```

For the remainder of this text, we will only use the `?<query-string>` shorthand for brevity, where the presence of the question mark implies that the endpoint needs to be prepended and that multiple query strings can be combined by separating them with and ampersand, like `?<query-string1>&<query-string2>`.

The various descriptors that can be used to search are passed as query strings as well. Each has their own specific syntax of the form `?<descriptor>=<parameters>` , which will be discussed next.

2.2.3 Searching by Tempo or Tuning

Since both tempo (in beats per minute) and tuning (frequency in Hz) are represented as single numbers, the syntax for searching them is identical. A number of ways exist to search tempo and tuning: by threshold, by interval or by tolerance.

Music pieces smaller than or greater than (or equal) than a certain threshold value can be found by passing query strings of the form `?(tempo|tuning)=(<|>|<=|>=)<value>`. For instance, music pieces slower than or equal to 90 beats per minute can be found as:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={'tempo': '<=90'})
```

If you want to find music pieces that fall within an interval, the required query string needs to be of the form `?(tempo|tuning)=<min>-<max>`, where the lower bound is inclusive and the upper bound exclusive. Finding music pieces with a tuning in the interval between 437.5 and 442.5 therefore requires the following API call to be made:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={'tuning': '437.5-442.5'})
```

Finally it is also possible to search tempo or tuning by a maximum allowed deviation (expressed as a percentage) with respect to a target value. The query string then takes a form of `?(tempo|tuning)=<target-value>+-<tolerance>%`. For example, music pieces with a tempo within 5% of 120 BPM can then be found as:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={'tempo': '120+-5%'})
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

2.2.4 Search by Chords

The chord vocabulary that can be used to search music pieces consists of the combination of twelve possible roots and five possible chord types:

```
root ::= A|Bb|B|C|Db|D|Eb|E|F|Gb|G|Ab
type ::= maj|min|7|maj7|min7
chord ::= <root><type>
```

Multiple of these chords can be specified by separating them with a hyphen `-` and passing them on as parameters to the search string as `?chords=<chord-name>[-<chord-name>]`. The query will then return music pieces that only contain those chords. For instance, pieces that only contain `Cmaj`, `Fmaj` and `Gmaj` chords are found by calling:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={‘chords’: 'Cmaj-Fmaj-Gmaj'})
```

The order in which the chords are specified does not affect the result, so `?chords=Cmaj-Fmaj-Gmaj` returns the same as `?chords=Gmaj-Cmaj-Fmaj`.

A more advanced way of searching for chords is achieved by adding the minimum time that the given chords need to be played during the piece. This coverage takes a percentage between 0 and 100 and is passed along as `?chords=<chords-list>,<coverage>%`. The simple search mentioned above equates to setting this parameter to 100%. Finding pieces that contain Amin and Emaj chords at least 60% of their duration can then be done by calling:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={‘chords’: 'Amin-Emaj,80%'})
```

2.2.5 Search by Global Key

Music pieces can also be searched according to their global key, which needs to have the following syntax:

```
tonic ::= A|A#|B|C|C#|D|D#|E|F|F#|G|G#
scale ::= major|minor
key ::= <tonic><scale>
```

A key of this form can simply be passed into the query string as `?global-key=<key>`, for instance to search for pieces in F major as follows:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={‘global-key’: 'Fmajor'})
```

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 688382

It is also possible to specify only a tonic or a scale, for instance to find all music pieces in a minor key:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={‘global-key’: 'minor'})
```

or with tonic G:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={‘global-key’: 'G'})
```

2.2.6 Combining Search Queries

Combining search queries is straightforward: it suffices to chain multiple descriptor parameters in the search query string as `?<descriptor1>=<param1>&<descriptor2>=<param2>`. For instance, searching for pieces in E minor with a tempo between 120 and 140 BPM is done as follows:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={'global-key': 'Eminor', 'tempo': '120-140'})
```

It is also possible to pass the name of a descriptor without parameter to the search string, `?<descriptor1>=<param1>&<descriptor2>`. A descriptor without parameter will not influence the search results, but will cause the result to also include the given descriptor. Doing so avoids to need to make additional calls to the enrichment API with the ids of the results afterwards. So if you want to get the chords of all music pieces with a tempo between 120 and 140 BPM, you can call:

```
requests.get('http://audio-analysis.eecs.qmul.ac.uk/function/ac-search',
params={'tempo': '120-140', 'chords': ''})
```

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 688382

3 Installation

3.1 Prerequisites

The AC Analysis Service is based on the OpenFAAS framework (https://www.openfaas.com/), which itself relies heavily on the Docker (https://www.docker.com) ecosystem. Its installation therefore requires a Docker installation with a container orchestration system such as Kubernetes or the built-in Docker Swarm. Follow the instructions on the OpenFAAS website to install the system. For the remainder of this guide, we also assume that the command line interface has been installed.

The descriptors calculated by the AC Analysis Service are stored in a MongoDB database, therefore an instance needs to be available for the service to work. This can be a locally hosted instance or a cloud database provided by a service such as MongoDB Atlas.

Finally, the code for service needs to be checked out from the git repository at https://github.com/AudioCommons/faas-ac-analysis.git.

3.2 Configuration

Before starting the service, a number of configuration steps need to be executed to adapt to the specific environment. First of all, the file `env-sample.yml` needs to be copied to `env.yml` and the value for the key `MONGODB_CONNECTION` inside the file needs to be adapted to your specific instance. This is a full connection string URI as described in the MongoDB manual, including username and password, of the form:

```
mongodb[+srv]://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:port
tN]]][/[authDB][?options]]
```

The ac-analysis function requires a configuration file to be set up at `ac-analysis/config.py`. This file has to contain a variable named `providers` variable that contains a list of content providers that are supported in this instance. The reasoning is that multiple alternative deployment scenarios are possible, for instance instead of having a centralised instance performing computations like the live instance at QMUL, an instance could be hosted by each of the content providers. In that case, they would each limit the list of providers to themselves. The advantage would then be that the audio to be analysed doesn't need to be transported over the internet. Note that the caching database in that scenario still needs to be a single centralised one, now web-accessible instead of just local, otherwise it wouldn't be possible to search over multiple content providers. Furthermore, the configuration file needs to contain a Python function `audio_uri(provider_id, provider)` that returns a URI to the audio file corresponding to `provider:provider_id`. This function needs to be able to process all providers in the previously defined `providers` variable.

Included in the repository is a relatively complicated example file `ac-analysis/config_cached_audio.py` aimed at centralised installations where the audio content is not available locally. It implements an audio caching mechanism where an audio file gets retrieved from a content provider and then stored locally such that the calculation of multiple descriptors doesn't require the audio to be transferred over the network multiple times. For this, it requires an object storage server, which can either be a locally hosted Minio instance or an S3-compatible cloud service. A simpler example, which does not cache audio but retrieves the necessary files directly from the content providers every time they're needed is available as `ac-analysis/config_direct_audio.py`.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Distributed setups where the audio is available locally will have a simpler configuration. The `audio_uri` function then simply needs to construct the URI to a local file based on its id, although the files need to be accessible over the local network because the filesystems in Docker containers are isolated. Passing local paths to files will therefore not work (and is not scalable anyway).

3.3 Deployment

When OpenFAAS has been installed, the code repository has been cloned and configured with the appropriate instance of a MongoDB, deploying the AC analysis service is straightforward.

Navigate to the `faas-ac-analysis` folder, and then - like any OpenFAAS function -- build the Docker images with `faas build` and deploy them with `faas deploy`. The analysis and search functions are then callable on the OpenFaas gateway (by default `http://127.0.0.1:8080`) through the paths `/function/ac-analysis` and `/function/ac-search` (see above).

3.4 Further Steps

To tighten the security of the AC Analysis Service installation, it is advised to install a reverse proxy such as Kong or Traefik in front of it. This makes it possible to limit access only to whitelisted endpoints (`/ac-analysis` and `/ac-search` in our case), enforce rate limiting and add HTTPS support. Furthermore, Cross-Origin Resource Sharing (CORS), authentication and load-balancing can be handled this way too.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

4 Integration of the Enrichment API with the AC Mediator

4.1 Registration Endpoints

Instead of passing the results of a textual search on the Mediator through to the enrichment API by the client, as shown above, it would be theoretically possible to integrate the results of the latter directly into the former. This would save the the client from making an additional API call and therefore lead to a better user experience. A possible scenario would be to pass an additional parameter `?descriptors=<comma-separated-descriptor-names>` to the Mediator, for example as:

```
requests.get('https://m2.audiocommons.org/api/audioclips/search',
             params={'pattern': 'bach', 'descriptors': 'tempo,global-key'},
             headers={'accept': 'application/json'})
```

Whenever the `descriptors` parameter is present in the call, the Mediator should call the enrichment API in the background, merge its result on the server and then return the aggregate.

If multiple analysis services are deployed in the Audio Commons Ecosystem, the Mediator needs to keep track of which service can provide what descriptor for what content provider. To aid in the registration of an analysis service with the Mediator, two additional endpoints are provided to advertise its capabilities:

- `/ac-analysis/providers`
- `/ac-analysis/descriptors`

4.2 Dealing with potential API blocking

One potential drawback of integrating the enrichment API into the Mediator is that when a descriptor is not already cached in the database, it needs to be calculated on demand. This calculation can potentially take a non-negligible amount of time, which gets compounded by the fact that the Mediator typically returns multiple results at once. This time waiting for the calculation to complete will block the response and will make the API appear unresponsive. This is one of the main reasons why the enrichment API is currently not integrated into the Mediator.

There are three possible ways to handle this potential API blocking:

- Documenting behaviour: The simplest solution requires no action, just warn the user that blocking is possible and a side-effect of the descriptor calculation. Since calling the enrichment API through the Mediator would be opt-in, it is unlikely that a potentially blocking call would be made accidentally.
- Pre-caching policy: It would be possible to avoid waiting for the calculation of descriptors by precalculating all of them as a matter of policy. The asynchronous calling of the enrichment API is made for this purpose, to queue the analysis and return straightaway, without blocking. If this API is called as part of a new music piece entering the AC Ecosystem (which means content providers need to monitored or integrate this call into their system themselves), the descriptors will be ready for use nearly immediately. The drawback is that this process requires a lot of storage. On the other hand, a music piece needs to be indexed before it can show up in the search API, which is another argument in favour of this policy.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

- Technological solution: Technically the most complex solution would be for the Mediator to not call the enrichment API directly, but through a publish-subscribe mechanism such as SEPA such that directly available descriptors can be returned immediately together with other metadata without waiting, while the response can be updated with descriptors that require computation time as soon as the latter are available.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

5 Conclusion

In this report, we described a web-based tool for the automatic semantic description of music pieces. We discuss its usage and its installation on a web server. It consists of two parts, an enriching API that allows to get extra audio-content-based information for music pieces obtained through a textual query to the AC Mediator and a searching API that allows to discover music pieces in the AC Ecosystem by their musical content.

Through these APIs, the content in the AC Ecosystem becomes more discoverable, thereby fulfilling one of the main goals set out at the beginning of this project. It allows users to build novel applications around the musical content of the pieces. The Jam with Jamendo app presented in D4.8 is one example of such an app, but the potential is broader since that one only makes use of a single descriptor whereas multiple descriptors can be combined in the final search API.

Since all music pieces pass through the same processing chain to calculate the same descriptor, one advantage is that it trivially unifies music pieces from different content providers. Scaling the system to new content providers is therefore straightforward. Unfortunately, scaling to new content providers is not as easy for the Mediator, on which the analysis service relies for id allocation and complementary metadata, because the Mediator needs to integrate different sources of metadata, with different scopes and interpretations.

Potential future improvements include the incorporation of more and better performing audio descriptor algorithms. Although some of the descriptor algorithms developed during task T4.2 have already been integrated in this tool, notably for tempo, tuning and global-key calculation, especially the mood recognition of the Essentia tool would be a prime candidate for inclusion. This would complete the wishlist of most popular descriptors to search audio by as identified in the survey reported in D2.1. Specifically for the search API, it would be good to extend it with more powerful options to influence the sorting of the results and add additional filtering, for example to restrict the search to a single content provider.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

Appendix 1: API reference

A1.1 Enrichment API Endpoints

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/chords?id=<ac-id>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/tempo?id=<ac-id>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/beats?id=<ac-id>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/global-key?id=<ac-id>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/keys?id=<ac-id>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/tuning?id=<ac-id>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-analysis/instruments?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/chords?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/tempo?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/beats?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/global-key?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/keys?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/tuning?id=<ac-id>

POST https://audio-analysis.eecs.qmul.ac.uk/async-function/ac-analysis/instruments?id=<ac-id>

A1.2 Search API Endpoints and Query Strings

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-search

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-search/<num-results>

GET https://audio-analysis.eecs.qmul.ac.uk/function/ac-search/<num-results>/<offset>

- ?providers=<provider>
- ?providers=<provider1>,<provider2>
- ?tempo=
- ?tempo=<<value>
- ?tempo=<=<value>
- ?tempo=><value>
- ?tuning=>=<value>
- ?tempo=<min>-<max>
- ?tempo=<value>+-<tolerance>%
- ?tuning=
- ?tuning=<<value>
- ?tuning=<=<value>
- ?tuning=><value>
- ?tuning=>=<value>

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

- ?tuning=<min>-<max>
- ?tuning=<value>+-<tolerance>%
- ?chords=
- ?chords=<chords-list>
- ?chords=<chords-list>,<coverage>%
- ?global-key=
- ?global-key=<key>
- ?global-key=<tonic>
- ?global-key=<scale>

```
root ::= A|Bb|B|C|Db|D|Eb|E|F|Gb|G|Ab
type ::= maj|min|7|maj7|min7
chord ::= <root><type>
chord-list ::= <chord>[“-”<chord-list>]
```

```
tonic ::= A|A#|B|C|C#|D|D#|E|F|F#|G|G#
scale ::= major|minor
key ::= <tonic><scale>
```

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N° 688382

Appendix 2: Example Output of the Enrichment API

A2.1 Chords descriptor for AC id jamendo-tracks:214

Time in seconds


```
{
   "chordSequence" : [
      {
         "start" : 0,
         "end" : 6.25,
         "label" : "A7"
      },
      {
         "start" : 6.25,
         "end" : 11.95,
         "label" : "Amin7"
      },
      {
         "start" : 11.95,
         "end" : 13.95,
         "label" : "Amaj"
      },
...
      {
         "start" : 121.35,
         "end" : 122.85,
         "label" : "Emin7"
      },
      {
         "start" : 122.85,
         "end" : 124.5,
         "label" : "Amaj7"
      }
   ],
   "duration" : 124.5,
   "confidence" : 0.581059390048154,
   "_id" : "jamendo-tracks:214"
}
```


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

A2.2 Tempo descriptor for AC id jamendo-tracks:214

Tempo in BPM


```
{
   "_id" : "jamendo-tracks:214",
   "tempo" : 120.218292236
}
```


A2.3 Beats descriptor for AC id jamendo-tracks:214

Beat locations in seconds


```
{
   "_id" : "jamendo-tracks:214",
   "beats" : [
      0.476009070873,
      0.96362811327,
      1.46285712719,
      1.9620860815,
      2.47292518616,
      2.97215414047,
      3.47138309479,
...
      122.078910828,
      122.473648071,
      122.891609192,
      123.309570312,
      123.727523804
   ]
}
```


A2.4 Global key descriptor for AC id jamendo-tracks:214


```
{
   "confidence" : 0.613963484764,
   "_id" : "jamendo-tracks:214",
   "key" : "D minor"
}
```


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

A2.5 Keys descriptor for AC id jamendo-tracks:214

Time in seconds


```
{
    "keys" : [
        {
             "time" : 0,
             "label" : "A major"
        },
        {
             "time" : 2.229115646,
             "label" : "E major"
        },
        {
             "time" : 5.94430839,
             "label" : "A major"
        },
...
        {
             "time" : 101.05324263,
             "label" : "Bb major"
        },
        {
             "time" : 102.539319728,
             "label" : "F major"
        },
        {
             "time" : 112.198820862,
             "label" : "Eb major"
        }
    ],
    "_id" : "jamendo-tracks:214",
}
```


A2.6 Tuning descriptor for AC id jamendo-tracks:214

Tuning frequency in Hz


```
{
   "_id" : "jamendo-tracks:214",
   "tuning" : 437.718536377
}
```


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382

A2.7 Instruments descriptor for AC id jamendo-tracks:214

Probability of presence for 24 different instruments


```
{
   "instruments" : {
      "Electric Bass" : 0.00043958,
      "Acoustic Bass" : 0.00464457,
      "Synthetic Bass" : 0.0298652,
      "Male Voice" : 0.0062637,
      "Conga" : 0.0521555,
      "Electronic Beats" : 0.00365136,
      "Shaker" : 0.000224567,
      "Harpsichord" : 0.0192729,
      "French Horn" : 0.00129627,
      "Synthesizer" : 0.0512298,
      "Flute" : 0.000220157,
      "Viola" : 0.000718654,
      "Harp" : 0.213958,
      "Acoustic Guitar" : 0.56241,
      "Violin" : 0.000309386,
      "Cello" : 0.0017522,
      "Choir" : 0.00928809,
      "Clarinet" : 0.000115898,
      "Organ" : 0.00356039,
      "Electric Guitar" : 9.62033e-05,
      "Drum Kit" : 0.0348383,
      "Female Voice" : 0.00129625,
      "Electric Piano" : 0.000892381,
      "Piano" : 0.00150107
   },
   "_id" : "jamendo-tracks:214"
}
```


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 688382