Avalon Music Server Documentation

Release 0.6.0

TSH Labs

Feb 16, 2018

Contents

1 Contents **3**

1.1 Requirements . . . 3
1.2 Installation . . . 4
1.3 CLI Tools . . . 7
1.4 WSGI Application . . . 8
1.5 API . . . 11
1.6 UUID Generation . . . 21
1.7 Developers . . . 22
1.8 Maintainers . . . 24
1.9 Change Log . . . 25

2 Indices and tables **31**

The Avalon Music Server is a Python WSGI application and several CLI scripts that, together, scan metadata from a music collection, store it in a database, and expose it as a JSON web service. It is available under the MIT license.

The Avalon Music Server is able to read metadata from ogg, flac, and mp3 files. Clients can then query the server for information about songs, albums, artists, and genres in the collection.

Features include:

- Support for Mp3, Vorbis (Ogg), or Flac audio files
- Support for multiple database backends
- Simple JSON interface including fast prefix matching
- Unicode output support
- Python 2.6 – 3.4

To install it simply run

```
$ pip install avalonms
```

Then, to scan your music collection

```
$ avalon-scan ~/Music
```

Then, start the application using a WSGI server like Gunicorn

```
$ gunicorn --preload avalon.app.wsgi:application
```

The documentation linked below will go into more detail about how to configure and run the Avalon Music Server in a production environment, how to interact with it using the JSON web service, and how to set up an environment to develop it.

CHAPTER 1

Contents

1.1 Requirements

If you follow the instructions in *Installation* (using pip) most of these requirements should be installed for you.

Python requirements:

- Python 2 >= 2.6 or Python 3 >= 3.3
- Argparse >= 1.2.1 (Or >= Python 2.7)
- Flask >= 0.10.1
- Mutagen >= 1.25.1
- SimpleJSON >= 3.5.2
- SQLAlchemy >= 0.9.4

In addition to the libraries above, you'll need a WSGI compatible server to run the Avalon Music Server. Gunicorn or uWSGI are both excellent choices. The rest of the documentation will assume you are using Gunicorn since that is what the reference install of the Avalon Music Server uses.

By default, the Avalon Music Server uses a SQLite database to store music meta data. If you wish to use another database type supported by SQLAlchemy (e.g. MySQL, PostgreSQL) you'll need to install an appropriate library for it.

For example, to install a PostgreSQL driver.

```
$ pip install psycopg2
```

Or a MySQL driver.

```
$ pip install mysql-python
```

1.2 Installation

Two possible ways to install the Avalon Music Server are described below. One is very simple and designed to get you up and running as quickly as possible. The other is more involved and designed to run the Avalon Music Server in a production environment.

All of these instructions are based on a machine running Debian Linux, but they should be applicable to any UNIX-like operating system (with a few modifications).

1.2.1 Quick Start Installation

This section will describe an extremely simple installation of the Avalon Music Server. If you just want to play around with the Avalon Music Server and don't have plans to run it in production, this guide is for you.

First, we'll install the virtualenv tool.

```
$ apt-get install python-virtualenv
```

Next, create a virtual environment that we'll install the Avalon Music Server into.

```
$ virtualenv ~/avalon
```

"Activate" the virtual environment.

```
$ source ~/avalon/bin/activate
```

Install the Avalon Music Server and Gunicorn.

```
$ pip install avalonms gunicorn
```

Scan your music collection and build a database with the meta data from it.

```
$ avalon-scan ~/path/to/music
```

Start the server.

```
$ gunicorn --preload avalon.app.wsgi:application
```

Then, in a different terminal, test it out!

```
$ curl http://localhost:8000/avalon/heartbeat
```

You should see the result `OKOKOK` if the server started correctly. To stop the server, just hit `CTRL-C` in the terminal that it is running in.

1.2.2 Production Installation

This section will describe one potential way to install, configure, and run the Avalon Music Server in production.

This particular installation uses Gunicorn, Supervisord, and Nginx. We'll set it up so that it can be deployed to using the included Fabric files in the future. We'll also make use of the bundled Gunicorn and Supervisor configurations.

Installing Dependencies

First, we'll install the virtualenv tool, Supervisor and Nginx using the package manager.

```
$ apt-get install python-virtualenv supervisor nginx
```

Setting Up The Environment

Next, we'll set up the environment on our server:

Create the group that will own the deployed code.

```
$ sudo groupadd dev
```

Add our user to it so that we can perform deploys without using sudo.

```
$ sudo usermod -g dev `whoami`
```

Create the directories that the server will be deployed into.

```
$ sudo mkdir -p /var/www/avalon/releases
```

Set the ownership and permissions of the directories.

```
$ sudo chown -R root:dev /var/www/avalon
$ sudo chmod -R u+rw,g+rw,o+r /var/www/avalon
$ sudo chmod g+s /var/www/avalon /var/www/avalon/releases
```

Add a new unprivileged user that the Avalon Music Server will run as.

```
$ sudo useradd --shell /bin/false --home /var/www/avalon --user-group avalon
```

Create a virtual environment that we'll install the Avalon Music Server into.

```
$ virtualenv /var/www/avalon/releases/20140717214022
```

Set the "current" symlink to the virtual environment we just created. This is the path that we'll we pointing our Supervisor and Gunicorn configurations at.

```
$ ln -s /var/www/avalon/releases/20140717214022 /var/www/avalon/current
```

Installing from PyPI

Now, let's install the Avalon Music Server, Gunicorn, and a Sentry client into the virtual environment we just created.

```
$ /var/www/avalon/current/bin/pip install avalonms gunicorn raven
```

The Avalon Music Server has an embedded default configuration file. In addition to that, we'll create our own copy of that configuration that we can customize.

```
$ /var/www/avalon/current/bin/avalon-echo-config > /var/www/avalon/local-settings.py
```

Avalon WSGI Application

We won't configure the Avalon WSGI application here, as part of installation. For more information about the available configuration settings for it, see the *WSGI Application* section.

Gunicorn

The installed Avalon Music Server comes with a simple Gunicorn configuration file that is available at `/var/www/avalon/current/share/avalonms/avalon-gunicorn.py` (or `ext/avalon-gunicorn.py` in the codebase). This file configures Gunicorn to:

- Bind the server to only the local interface, port `8000`.
- Spawn three worker processes that will handle requests.
- Use preload mode so that the workers will be able to take advantage of copy-on-write optimizations done by the operating system to save RAM.

Supervisor

The installed Avalon Music server also comes with a simple Supervisord configuration file. This file runs the Avalon Music Server as an unprivileged user, uses the Gunicorn HTTP WSGI server, restarts it if it crashes, and pipes all output to a log file. This is available at `/var/www/avalon/current/share/avalonms/avalon-supervisor-gunicorn.conf` (or `ext/avalon-supervisor-gunicorn.conf` in the codebase).

When you installed Supervisor earlier (if you're on Debian) it created a directory that configurations can be placed into: `/etc/supervisor/conf.d`. Copy the bundled Supervisor configuration file into this directory and set the owner and permissions appropriately.

```
$ sudo cp /var/www/avalon/current/share/avalonms/avalon-supervisor-gunicorn.conf /etc/
↪supervisor/conf.d/
$ sudo chown root:root /etc/supervisor/conf.d/avalon-supervisor-gunicorn.conf
$ sudo chmod 644 /etc/supervisor/conf.d/avalon-supervisor-gunicorn.conf
```

Nginx

Though Gunicorn can run as an HTTP server, you should use a dedicated web server in front of it as a reverse proxy if you plan on exposing it on the public Internet. If so, Nginx is a solid, lightweight, easy to configure choice. In the instructions below, replace `api.example.com` with the domain that you wish to run the Avalon Music Server at.

When you installed Nginx earlier it created a directory that server configurations can be placed into: `/etc/nginx/sites-available/` (if you're on Debian). If you're not on Debian the directory may be in a different location such as `/etc/nginx/conf.d` or you may have a single configuration file: `/etc/nginx/nginx.conf`.

If you have a directory for configurations, create a new file named `api_example_com.conf` with the contents below. If you only have a single configuration file, add the contents below inside the `http` section.

```
upstream avalon {
         server localhost:8000;
}

server {
   listen 80;
   server_name api.example.com;
```

```
    location /avalon {
            proxy_pass http://avalon;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
    }
}
```

If you're on Debian, enable the configuration like so.

```
$ sudo ln -s /etc/nginx/sites-available/api_example_com.conf /etc/nginx/sites-enabled/
```

Start the Server

Now that everything is configured, let's try starting Nginx and Supervisor (which will, in turn, start the Avalon Music Server) and testing it out.

```
$ sudo service supervisor start
$ sudo service nginx start
$ curl http://api.example.com/avalon/heartbeat
```

If everything was installed correctly, the `curl` command should return the string `OKOKOK`.

1.3 CLI Tools

There are several CLI tools that are used as part of the Avalon Music Server besides the actual server part.

Each of these tools will be detailed below.

1.3.1 avalon-echo-config

Print the contents of the default configuration for the Avalon Music Server WSGI application to `STDOUT`.

Useful for creating a configuration file for the server that can be customized as described below.

Synopsis

```
$ avalon-echo-config [options]
```

Options

`-h --help` Prints how to invoke the command and supported options and exits.

`-V --version` Prints the installed version of the Avalon Music Server and exits.

Examples

```
$ avalon-echo-config > /var/www/avalon/local-settings.py
```

1.3.2 avalon-scan

Scan a music collection for meta data and insert it into a database, making sure to create the database schema if it does not already exist.

Database connection information is loaded from the default configuration file and optionally a configuration override file (whose location is specified by the `AVALON_CONFIG` environmental variable). This can also be overridden using the `--database` option.

Synopsis

```
$ avalon-scan [options] {music collection path}
```

Options

`-h --help` Prints how to invoke the command and supported options and exits.

`-V --version` Prints the installed version of the Avalon Music Server and exits.

`-d <URL> --database <URL>` Database URL connection string for the database to write music collection meta data to. If not specified the value from the default configuration file and configuration file override will be used. The URL must be one supported by SQLAlchemy.

`-q --quiet` Be less verbose, only emit ERROR level messages to the console.

Examples

Use the database type and location specified by the default configuration file (usually SQLite and `/tmp/avalon.sqlite`) and scan the music collection in the directory 'music'.

```
$ avalon-scan ~/music
```

Use the database type and location specified by a custom configuration file and scan the music collection in the 'media' directory.

```
$ AVALON_CONFIG=/home/user/avalon/local-settings.py avalon-scan /home/media
```

Use a PostgreSQL database type and connect to a remote database server and scan the music collection in the directory 'music'.

```
$ avalon-scan --database 'postgresql+psycopg2://user:password@server/database' ~/music
```

Use a SQLite database type in a non-default location and scan the music collection in the directory '/home/files/music'.

```
$ avalon-scan --database 'sqlite:////var/db/avalon.sqlite' /home/files/music
```

1.4 WSGI Application

The Avalon WSGI application is meant to be run with a Python WSGI server such as Gunicorn.

The application will...

- Load music collection meta data from a database (as specified by the configuration files described below).

- Build structures that can be used to search and query a music collection.
- Begin serving HTTP requests with a JSON *API*.

1.4.1 Running

The main entry point for the Avalon Music Server WSGI application is the module `avalon.app.wsgi` – the WSGI callable is the attribute `application` within the module. An example of how to use this module and callable with Gunicorn (with three worker processes) is below.

```
$ gunicorn --preload --workers 3 avalon.app.wsgi:application
```

Note that we're using the `--preload` mode which will save us memory when using multiple worker processes.

1.4.2 Configuration

The Avalon WSGI application uses an embedded default configuration file. Settings in that file (described below) can be overridden with a custom configuration file generated as below (assuming the Avalon Music Server has already been installed).

```
$ avalon-echo-config > /var/www/avalon/local-settings.py
```

The file at `/var/www/avalon/local-settings.py` will be an exact copy of the default configuration file. You can change the settings in this new copy and they will override the default settings. Any settings you do not change (or settings removed from the file) will use their default values.

After you have customized this file, you need to tell the Avalon WSGI application to use this file. This is done by setting the value of the `AVALON_CONFIG` environmental variable to the path of this configuration file. An example (once again, using Gunicorn) is below.

```
$ gunicorn --env AVALON_CONFIG=/var/www/avalon/local-settings.py \
    --preload --workers 3 avalon.app.wsgi:application
```

1.4.3 Settings

The following configuration settings are available to customize the behavior of the Avalon WSGI application. The table below describes the settings and how they are used.

Note: Note that some settings available in the configuration are not meant to be changed by end users and are hence omitted below.

`DATABASE_URL`	URL that describes the type of database to connect to and the credentials for connecting to it. The URL mus
`LOG_DATE_FORMAT`	Date format for timestamps in logging messages. The supported tokens for this setting are described in the P
`LOG_FORMAT`	Format for messages logged directly by the Avalon Music Server. See the Python logging documentation for
`LOG_LEVEL`	How verbose should logging done by the Avalon WSGI application be? By default, all messages `INFO` and
`LOG_PATH`	Where should messages be logged to? By default all messages are logged to the `STDERR` stream (the consol
`REQUEST_PATH`	Base path to use for handling requests to the WSGI application. For example, with a value of '/avalon' the h
`SENTRY_DSN`	URL that describes how to log errors to a centralized 3rd party error-logging service, Sentry. This functional
`STATSD_HOST`	Hostname to write Statsd timers and counters to if there is a client installed. The expected client will discard
`STATSD_PORT`	Port to write Statsd timers and counters to. Port 8125 is the port that the Etsy Statsd implementation runs on
`STATSD_PREFIX`	Prefix all metrics emitted with this string. Useful to make sure metrics from the Avalon Music Server don't p

1.4.4 Architecture

Database

The Avalon Music Server CLI tool `avalon-scan` writes music metadata to a database when it scans a music collection. The WSGI application and reads the meta back when it starts.

In each case, when connecting to a database for the first time, the CLI script and the WSGI application will attempt to create the required database schema if it does not already exist.

Provided that you attempt to scan your music collection before running the WSGI application, the scanning portion must have read/write access to the database and the WSGI application must have read access. Otherwise, if you are running the WSGI application, connecting to a database before inserting anything into it via scanning, the WSGI application will attempt create the required schema and will require read/write access.

Workers

The Avalon WSGI application is, for the most part, CPU bound and immutable after start up. Therefore it is a good fit for multiprocess workers and (if your Python implementation doesn't have a Global-Interpreter-Lock) threaded workers.

Logging

By default, the Avalon WSGI application sends logging messages to `STDERR`. This means that if you want to send these messages to a file or a Syslog, you have to configure the logging of the WSGI HTTP server that you are using to run it (or the process manager that runs the WSGI HTTP server).

The Avalon WSGI application can also be configured to send log messages directly to a log file. In this case, the file must be writable by the user that the application is being run as.

Sentry

Sentry is a centralized, 3rd-party, error-logging service. It is available as a paid, hosted, service. However, both the client and server are Free Software and can be run by anyone.

The Avalon WSGI application will optionally log unhandled exceptions to a Sentry instance provided these things are true (otherwise logging to Sentry will not be used).

1. The Sentry client is installed and can be imported.
2. There is a `SENTRY_DSN` configuration setting available and correctly configured.

To install the client run the following command from within the virtualenv that the Avalon WSGI application is installed in.

```
$ pip install raven
```

Statsd

Statsd is a daemon that listens for metrics sent over UDP and periodically pushes them to Graphite.

The Avalon WSGI application will optionally record the execution time of each endpoint if the Statsd client is installed. The Statsd service to send metrics to can be configured with the `STATSD_HOST` and `STATSD_PORT` configuration settings.

To install the client run the following command from within the virtualenv that the Avalon WSGI application is installed in.

```
$ pip install statsd
```

1.4.5 Deployment

If you followed the steps in *Installation* you should be able to use the bundled Fabric deploy scripts to manage your Avalon WSGI application installation.

Note that the Fabric deploy scripts will also install the Gunicorn HTTP server and a client for the Sentry service (however, Sentry won't be used unless you have explicitly configured it).

Some assumptions made by the Fabric deploy scripts:

- You have already created and set the permissions of the directory that will be getting deployed to (as described in installation).
- You have SSH access to the server you are deploying to.
- You have the ability to `sudo` on the server you are deploying to.

If all these things are true, you should be able to deploy a new version of the Avalon WSGI application with a few simple steps.

First, make sure the build environment is clean and then generate packages to install.

```
$ fab clean build.released
```

Next, upload the generated packages, and install them.

```
$ fab -H api.example.com deploy.install
```

Restart the Avalon WSGI application if it's already running.

```
$ fab -H api.example.com deploy.restart
```

That's it! The Avalon WSGI application should now be running on your server.

1.5 API

The Avalon Music Server handles requests on the specified interface and port at the path `/avalon`.

Endpoints return information about a music collection in JSON format based on path and/or query string parameters. Endpoints will return data as JSON for sucessful and error requests.

1.5.1 Heartbeat Endpoint

The `heartbeat` endpoint returns the plain text string `OKOKOK` if the server has completed starting and loading collection data. Requests to the heartbeat will hang until the server has completed starting. The heartbeat does NOT indicate if the server is healthy and serving requests correctly, only that it has completed start up. The idea being, that this is used durning a rolling deploy process where there are multiple nodes behind a load balancer.

Path and method

```
GET /avalon/heartbeat
```

Note: This path may be different depending on your `REQUEST_PATH` configuration setting.

Parameters

- The `heartbeat` endpoint doesn't support any parameters.

Possible error responses

- The `heartbeat` endpoint doesn't return a JSON response object.

Success output format

```
OKOKOK
```

Error output format

```
NONONO
```

Example request

- http://localhost:8000/avalon/heartbeat

1.5.2 Songs endpoint

The `songs` endpoint returns data for individual songs. The results returned can be limited and filtered based on query string parameters.

Path and method

```
GET /avalon/songs
```

Note: This path may be different depending on your `REQUEST_PATH` configuration setting.

Filtering Parameters

Name	Required?	Type	Multiple?	Description
`album`	No	`string`	No	Select only songs belonging to this album, exact match, not case sensitive.
`album_id`	No	`string`	No	Select only songs belonging to this album by UUID. The UUID is expected to be formatted using hexadecimal digits or hexadecimal digits with hyphens. If the UUID is not formatted correctly error code `101` (invalid parameter type) will be returned.
`artist`	No	`string`	No	Select only songs by this artist, exact match, not case sensitive.
`artist_id`	No	`string`	No	Select only songs by this artist by UUID. The UUID is expected to be formatted using hexadecimal digits or hexadecimal digits with hyphens. If the UUID is not formatted correctly error code `101` (invalid parameter type) will be returned.
`genre`	No	`string`	No	Select only songs belonging to this genre, exact match, not case sensitive.
`genre_id`	No	`string`	No	Select only songs belonging to this genre by UUID. The UUID is expected to be formatted using hexadecimal digits or hexadecimal digits with hyphens. If the UUID is not formatted correctly error code `101` (invalid parameter type) will be returned.
`query`	No	`string`	No	Select only songs whose album, artist, genre, or name contains `query`. The match is not case sensitive and unicode characters will be normalized if possible before being compared (in the `query` and fields being compared). The `query` is compared using prefix matching against each portion of the album, artist, genre, or song name (delimited by whitespace).

Other Parameters

Name	Required?	Type	Multiple?	Description
`limit`	No	`integer`	No	If there are more than `limit` results, only `limit` will be returned. This must be a positive `integer`. If the `offset` parameter is present, the `limit` will be applied after the `offset`. If the `limit` is not an integer error code `101` (invalid parameter type) will be returned. If the `limit` is not positive error code `102` (invalid parameter value) will be returned.
`offset`	No	`integer`	No	Skip the first `offset` entries returned as part of a result set. This must be a positive `integer`. This parameter does not have any effect if the `limit` parameter is not also present. If the `offset` is not an integer error code `101` (invalid parameter type) will be returned.
`order`	No	`string`	No	Name of the field to use for ordering the result set. Any valid field of the members of the result set may be used. If the `order` is not a valid field error code `100` (invalid parameter name) will be returned.
`direction`	No	`string`	No	Direction to sort the results in. Valid values are `asc` or `desc`. This parameter does not have any effect if the `order` parameter is not also present. If the `direction` is not `asc` or `desc` error code `102` (invalid parameter value) will be returned.

Example requests

- http://localhost:8000/avalon/songs?artist=NOFX
- http://localhost:8000/avalon/songs?artist_id=b048612e-1207-59f4-bbeb-ba0bc9a48cd1
- http://localhost:8000/avalon/songs?query=Live&artist=The+Bouncing+Souls

- http://localhost:8000/avalon/songs?album_id=2d24515c-a459-552a-b022-e85d1621425a
- http://localhost:8000/avalon/songs?album_id=2d24515ca459552ab022e85d1621425a
- http://localhost:8000/avalon/songs?genre=Ska
- http://localhost:8000/avalon/songs?genre_id=8794d7b7-fff3-50bb-b1f1-438659e05fe5
- http://localhost:8000/avalon/songs?query=anywhere

Possible error responses

Code	Message key	HTTP code	Description
100	avalon.service.error.invalid_input_name	400	An error that indicates that the name of a field specified is not a valid field.
101	avalon.service.error.invalid_input_type	400	An error that indicates the type of a parameter is not valid for that particular parameter.
102	avalon.service.error.invalid_input_value	400	An error that indicates the value of a parameter is not valid for that particular parameter.

Success output format


```
{
  "warnings": [],
  "success": [
    {
      "year": 2004,
      "track": 8,
      "name": "She's a Rebel",
      "album": "American Idiot",
      "album_id": "9ff51c16-2a7a-581c-b0b6-e28f5004139f",
      "artist": "Green Day",
      "artist_id": "b048612e-1207-59f4-bbeb-ba0bc9a48cd1",
      "genre": "Punk",
      "genre_id": "8794d7b7-fff3-50bb-b1f1-438659e05fe5",
      "id": "176cdea2-eb07-59ea-a809-2c6e23198cc8",
      "length": 120
    },
    {
      "year": 2002,
      "track": 11,
      "name": "Rotting",
      "album": "Shenanigans",
      "album_id": "9ddfbc73-6519-5ddf-a493-116cf3add9e1",
      "artist": "Green Day",
      "artist_id": "b048612e-1207-59f4-bbeb-ba0bc9a48cd1",
      "genre": "Punk",
      "genre_id": "8794d7b7-fff3-50bb-b1f1-438659e05fe5",
      "id": "840d20d8-58c6-50f6-b031-2a5a1b7c6f91",
      "length": 171
    }
  ],
  "errors": []
}
```

Error output format


```
{
  "warnings": [],
  "success": null,
  "errors": [
    {
      "payload": {
        "value": -1,
        "field": "limit"
      },
      "message_key": "avalon.service.error.invalid_input_value",
      "message": "The value of limit may not be negative",
      "code": 102
    }
  ]
}
```


1.5.3 Albums endpoint

The `albums` endpoint returns data for all the different albums that songs in the music collection belong to.

Path and method

```
GET /avalon/albums
```

Note: This path may be different depending on your `REQUEST_PATH` configuration setting.

Filtering Parameters

Name	Required?	Type	Multiple?	Description
`query`	No	`string`	No	Select only albums whose name contains `query`. The match is not case sensitive and unicode characters will be normalized if possible before being compared (in the `query` and fields being compared). The `query` is compared using prefix matching against each portion of the album (delimited by whitespace).

Other Parameters

Name	Required?	Type	Multiple?	Description
`limit`	No	`integer`	No	If there are more than `limit` results, only `limit` will be returned. This must be a positive `integer`. If the `offset` parameter is present, the `limit` will be applied after the `offset`. If the `limit` is not an integer error code `101` (invalid parameter type) will be returned. If the `limit` is not positive error code `102` (invalid parameter value) will be returned.
`offset`	No	`integer`	No	Skip the first `offset` entries returned as part of a result set. This must be a positive `integer`. This parameter does not have any effect if the `limit` parameter is not also present. If the `offset` is not an integer error code `101` (invalid parameter type) will be returned.
`order`	No	`string`	No	Name of the field to use for ordering the result set. Any valid field of the members of the result set may be used. If the `order` is not a valid field error code `100` (invalid parameter name) will be returned.
`direction`	No	`string`	No	Direction to sort the results in. Valid values are `asc` or `desc`. This parameter does not have any effect if the `order` parameter is not also present. If the `direction` is not `asc` or `desc` error code `102` (invalid parameter value) will be returned.

Example request

- http://localhost:8000/avalon/albums
- http://localhost:8000/avalon/albums?query=live
- http://localhost:8000/avalon/albums?order=name&direction=asc
- http://localhost:8000/avalon/albums?order=name&direction=desc&limit=5

Possible error responses

Code	Message key	HTTP code	Description
100	avalon.service.error.invalid_input_name	400	An error that indicates that the name of a field specified is not a valid field.
101	avalon.service.error.invalid_input_type	400	An error that indicates the type of a parameter is not valid for that particular parameter.
102	avalon.service.error.invalid_input_value	400	An error that indicates the value of a parameter is not valid for that particular parameter.

Success output format

```
{
  "warnings": [],
  "success": [
    {
      "name": "The Living End",
      "id": "9f311017-f1a8-598c-b842-fe873a4d198f"
    },
```

```
    {
      "name": "End of the Century",
      "id": "5209928c-4527-5fa5-a1de-affc4d9f6c11"
    },
    {
      "name": "Endgame",
      "id": "491672c5-adbe-5414-a4b5-cb6f3af03a6a"
    }
  ],
  "errors": []
}
```


Error output format


```
{
  "warnings": [],
  "success": null,
  "errors": [
    {
      "payload": {
        "value": -1,
        "field": "limit"
      },
      "message_key": "avalon.service.error.invalid_input_value",
      "message": "The value of limit may not be negative",
      "code": 102
    }
  ]
}
```


1.5.4 Artists endpoint

The `artists` endpoint returns data for all the different artists that songs in the music collection are performed by.

Path and method

```
GET /avalon/artists
```

Note: This path may be different depending on your `REQUEST_PATH` configuration setting.

Filtering Parameters

Name	Required?	Type	Multiple?	Description
`query`	No	`string`	No	Select only artists whose name contains `query`. The match is not case sensitive and unicode characters will be normalized if possible before being compared (in the `query` and fields being compared). The `query` is compared using prefix matching against each portion of the artist (delimited by whitespace).

Other Parameters

Name	Required?	Type	Multiple?	Description
`limit`	No	`integer`	No	If there are more than `limit` results, only `limit` will be returned. This must be a positive `integer`. If the `offset` parameter is present, the `limit` will be applied after the `offset`. If the `limit` is not an integer error code `101` (invalid parameter type) will be returned. If the `limit` is not positive error code `102` (invalid parameter value) will be returned.
`offset`	No	`integer`	No	Skip the first `offset` entries returned as part of a result set. This must be a positive `integer`. This parameter does not have any effect if the `limit` parameter is not also present. If the `offset` is not an integer error code `101` (invalid parameter type) will be returned.
`order`	No	`string`	No	Name of the field to use for ordering the result set. Any valid field of the members of the result set may be used. If the `order` is not a valid field error code `100` (invalid parameter name) will be returned.
`direction`	No	`string`	No	Direction to sort the results in. Valid values are `asc` or `desc`. This parameter does not have any effect if the `order` parameter is not also present. If the `direction` is not `asc` or `desc` error code `102` (invalid parameter value) will be returned.

Example request

- http://localhost:8000/avalon/artists
- http://localhost:8000/avalon/artists?query=who
- http://localhost:8000/avalon/artists?order=id
- http://localhost:8000/avalon/artists?order=name&limit=10&offset=20

Possible error responses

Code	Message key	HTTP code	Description
100	avalon.service.error.invalid_input_name	400	An error that indicates that the name of a field specified is not a valid field.
101	avalon.service.error.invalid_input_type	400	An error that indicates the type of a parameter is not valid for that particular parameter.
102	avalon.service.error.invalid_input_value	400	An error that indicates the value of a parameter is not valid for that particular parameter.

Success output format

```
{
  "warnings": [],
  "success": [
    {
      "name": "Bad Religion",
      "id": "5cede078-e88e-5929-b8e1-cfda7992b8fd"
    },
```

```
    {
      "name": "Bad Brains",
      "id": "09b00809-23b3-50a3-a4ca-bba26d769c3b"
    }
  ],
  "errors": []
}
```

Error output format

```
{
  "warnings": [],
  "success": null,
  "errors": [
    {
      "payload": {
        "value": "foo",
        "field": "offset"
      },
      "message_key": "avalon.service.error.invalid_input_type",
      "message": "Invalid field value for integer field offset: 'foo'",
      "code": 101
    }
  ]
}
```

1.5.5 Genres endpoint

The `genres` endpoint returns data for all the different genres that songs in the music collection belong to.

Path and method

`GET /avalon/genres`

Note: This path may be different depending on your `REQUEST_PATH` configuration setting.

Filtering Parameters

Name	Required?	Type	Multiple?	Description
`query`	No	`string`	No	Select only genres whose name contains `query`. The match is not case sensitive and unicode characters will be normalized if possible before being compared (in the `query` and fields being compared). The `query` is compared using prefix matching against each portion of the genre (delimited by whitespace).

Other Parameters

Name	Required?	Type	Multiple?	Description
`limit`	No	`integer`	No	If there are more than `limit` results, only `limit` will be returned. This must be a positive `integer`. If the `offset` parameter is present, the `limit` will be applied after the `offset`. If the `limit` is not an integer error code `101` (invalid parameter type) will be returned. If the `limit` is not positive error code `102` (invalid parameter value) will be returned.
`offset`	No	`integer`	No	Skip the first `offset` entries returned as part of a result set. This must be a positive `integer`. This parameter does not have any effect if the `limit` parameter is not also present. If the `offset` is not an integer error code `101` (invalid parameter type) will be returned.
`order`	No	`string`	No	Name of the field to use for ordering the result set. Any valid field of the members of the result set may be used. If the `order` is not a valid field error code `100` (invalid parameter name) will be returned.
`direction`	No	`string`	No	Direction to sort the results in. Valid values are `asc` or `desc`. This parameter does not have any effect if the `order` parameter is not also present. If the `direction` is not `asc` or `desc` error code `102` (invalid parameter value) will be returned.

Example request

- http://localhost:8000/avalon/genres
- http://localhost:8000/avalon/genres?query=rock
- http://localhost:8000/avalon/genres?order=name
- http://localhost:8000/avalon/genres?order=name&limit=10

Possible error responses

Code	Message key	HTTP code	Description
100	avalon.service.error.invalid_input_name	400	An error that indicates that the name of a field specified is not a valid field.
101	avalon.service.error.invalid_input_type	400	An error that indicates the type of a parameter is not valid for that particular parameter.
102	avalon.service.error.invalid_input_value	400	An error that indicates the value of a parameter is not valid for that particular parameter.

Success output format

```
{
  "warnings": [],
  "success": [
    {
      "name": "Hard Rock",
      "id": "ec93d3f1-3642-5beb-bb10-07f29bb18fc5"
    },
```

```
    {
      "name": "Punk Ska",
      "id": "3af7ba62-d87f-5258-af62-d7c5655ec567"
    }
  ],
  "errors": []
}
```

Error output format

```
{
  "warnings": [],
  "success": null,
  "errors": [
    {
      "payload": {
        "value": -10,
        "field": "offset"
      },
      "message_key": "avalon.service.error.invalid_input_value",
      "message": "The value of offset may not be negative",
      "code": 102
    }
  ]
}
```

1.6 UUID Generation

The Avalon Music Server uses UUIDs (version 5) to act as unique identifiers for albums, artists, genres, and tracks. If you wish to generate compatible IDs outside of the Avalon Music Server the process is as follows (in Python):

1.6.1 Albums

Album IDs are generated from the lowercase, UTF-8 encoded, name of the album.

The namespace UUID is `7655e605-6eaa-40d8-a25f-5c6c92a4d31a`.

```
>>> import uuid
>>> album_namespace = uuid.UUID('7655e605-6eaa-40d8-a25f-5c6c92a4d31a')
>>> album_name = u'¡Uno!'.lower().encode('utf-8')
>>> album_id = uuid.uuid5(album_namespace, album_name)
>>> album_id
UUID('32792eb5-03ff-5837-9869-d77ac9b5c99f')
```

1.6.2 Artists

Artist IDs are generated from the lowercase, UTF-8 encoded, name of the artist.

The namespace UUID is `fe4df0f6-2c55-4ba6-acf3-134eae3e710e`.

```
>>> import uuid
>>> artist_namespace = uuid.UUID('fe4df0f6-2c55-4ba6-acf3-134eae3e710e')
>>> artist_name = u'Minor Threat'.lower().encode('utf-8')
>>> artist_id = uuid.uuid5(artist_namespace, artist_name)
>>> artist_id
UUID('debcc564-211b-559c-b810-e72598bdaf47')
```

1.6.3 Genres

Genre IDs are generated from the lowercase, UTF-8 encoded, name of the genre.

The namespace UUID is `dd8dbd9c-8ed7-4719-80c5-71d978665dd0`.

```
>>> import uuid
>>> genre_namespace = uuid.UUID('dd8dbd9c-8ed7-4719-80c5-71d978665dd0')
>>> genre_name = u'Punk Ska'.lower().encode('utf-8')
>>> genre_id = uuid.uuid5(genre_namespace, genre_name)
>>> genre_id
UUID('3af7ba62-d87f-5258-af62-d7c5655ec567')
```

1.6.4 Songs

Song IDs are generated from the case sensitive path of the file, encoded as UTF-8.

The namespace UUID is `4151ace3-6a98-41cd-a3de-8c242654cb67`.

```
>>> import uuid
>>> song_namespace = uuid.UUID('4151ace3-6a98-41cd-a3de-8c242654cb67')
>>> song_path = u'/music/Voodoo_Glow_Skulls/02-adicción,_tradición,_revolución.ogg'.
↪encode('utf-8')
>>> song_id = uuid.uuid5(song_namespace, song_path)
>>> song_id
UUID('56b33b92-d5b6-5971-b166-dc959b442c0c')
```

1.7 Developers

1.7.1 Prerequisites

Make sure you have the virtualenv tool available. You can find further instructions in the *Installation* section or at the virtualenv website.

All steps below assume you are using a virtual environment named `env` inside the root directory of the git checkout. It's not important what name you use, this is only chosen to make the documentation consistent. Most of the commands below reference the `pip`, `virtualenv`, and `python` instances installed in the `env` environment. This ensures that they run in the context of the environment where we've set up the Avalon Music Server.

1.7.2 Environment Setup

First, fork the Avalon Music Server on GitHub.

Check out your fork of the source code.

```
$ git clone https://github.com/you/avalonms.git
```

Add the canonical Avalon Music Server repo as `upstream`. This might be useful if you have to keep your branch / repo up to date before creating a pull request.

```
$ git remote add upstream https://github.com/tshlabs/avalonms.git
```

Create and set up a branch for your awesome new feature or bug fix.

```
$ cd avalonms
$ git checkout -b feature-xyz
$ git push origin feature-xyz:feature-xyz
$ git branch -u origin/feature-xyz
```

Set up a virtual environment.

```
$ virtualenv env
```

Enter the virtualenv install required dependencies.

```
$ source env/bin/activate
$ pip install --allow-external argparse -r requirements.txt
$ pip install -r requirements-dev.txt
$ pip install -r requirements-prod.txt
```

Install the checkout in "development mode".

```
$ pip install -e .
```

1.7.3 Running The Server

The Avalon Music Server WSGI application can be run with Gunicorn (which was installed above from the `requirements-prod.txt` file) or any other WSGI application server. Make sure that you have entered the virtualenv you created earlier.

```
$ gunicorn --preload avalon.app.wsgi:application
```

1.7.4 Memory Profiling

The Avalon Music Server WSGI application can optionally log the memory used by various internal data structures. This can be useful for minimizing the resource footprint of the server when adding new features.

When enabled, memory usage will be writen to the configured logger. This feature is only enabled when the Pympler package is installed and the configured log level is `DEBUG`.

To enable this do the following.

Install the profiler.

```
$ pip install pympler
```

Change the Avalon Music Server log level in your local `settings.py` file.

```
LOG_LEVEL = logging.DEBUG
```

1.7.5 Contributing

Next, code up your feature or bug fix and create a pull request. If you're new to Git or GitHub, take a look at the GitHub help site.

1.7.6 Useful Commands

The Avalon Music Server uses tox to run tests in isolated virtualenvs. You can run the tests using the command below. Make sure that you have entered the virtualenv you created earlier.

```
$ tox test
```

You can also run the unit tests for a specific Python version.

```
$ TOXENV=py33 tox test
```

If you're making changes to the documentation, the command below will build the documentation for you. To view it, open up `doc/build/html/index.html` in your web browser.

```
$ fab clean docs
```

1.8 Maintainers

Note: The intended audience for this section is the Avalon Music Server maintainers. If you are a user of the Avalon Music Server, you don't need worry about this.

These are the steps for releasing a new version of the Avalon Music Server. The steps assume that all the changes you want to release have already been merged to the `master` branch. The steps further assumed that you've run all the unit tests and done some ad hoc testing of the changes.

1.8.1 Versioning

The Avalon Music Server uses semantic versioning of the form `major.minor.patch`. All backwards incompatible changes after version `1.0.0` will increment the major version number. All backwards incompatible changes prior to version `1.0.0` will increment the minor version number.

Since this is a Python project, only the subset of the semantic versioning spec that is compatible with PEP-440 will be used.

The canonical version number for the Avalon Music Server is contained in the file `avalon/__init__.py` (relative to the project root). Increment this version number based on the nature of the changes being included in this release.

Do not commit.

1.8.2 Change Log

Update the *change log* to include all relevant changes since the last release. Make sure to note which changes are backwards incompatible.

Update the date of the most recent version to today's date.

Commit the version number and change log updates.

1.8.3 Tagging

Create a new tag based on the new version of the Avalon Music Server.

```
$ git tag avalonms-0.5.0
```

Push the committed changes and new tags.

```
$ fab push push_tags
```

1.8.4 Building

Clean the checkout before trying to build.

```
$ fab clean
```

Build source and binary distributions and upload them.

```
$ fab pypi
```

1.8.5 Update PyPI

If the package metadata has changed since the last release, login to PyPI and update the description field or anything else that needs it.

https://pypi.python.org/pypi/avalonms

1.9 Change Log

1.9.1 0.6.0 - 2015-11-09

- Add `REQUEST_PATH` configuration setting to allow the base URL for the server to be customized. The default will remain `/avalon`.
- Minor code and documentation cleanup.

1.9.2 0.5.1 - 2015-04-04

- Packaging fixes (use `twine` for uploads to PyPI, stop using the setup.py `register` command).
- Add documentation of the steps for performing a release (*Maintainers*).
- Split usage documentation between *the CLI* and *the server*.

1.9.3 0.5.0 - 2015-01-04

- Add optional support for recording method execution times to Statsd. Enabling timing requires installing the pystatsd client and setting configuration values to point to your statsd instance.
- Remove `supervisor.config` and `supervisor.user` tasks from bundled Fabric script and move `supervisor.restart` to `deploy.supervisor` (along with having Supervisor gracefully reload instead of restart).

1.9.4 0.4.0 - 2014-11-24

- Change to Mutagen for reading audio tags now that it supports Python 3.
- Support for Python 3.3 and 3.4.
- Reduce memory usage during bootstrap by reading metadata in batches.
- Reduce memory usage during collection scanning by inserting tracks in batches.

1.9.5 0.3.1 - 2014-10-12

- Include installation of a Sentry client in Fabric deploy task
- Use Py.test and Tox for running tests.
- Added a "Quick Start" section to the installation docs.
- Use Tunic library in Fabric deploy scripts.

1.9.6 0.3.0 - 2014-08-17

- **Breaking change**: Avalon Music Server is now a WSGI application and CLI scripts, not a stand-alone server.
- **Breaking change**: Response format changed to include `errors`, `warnings`, and `success` top-level elements. Response format of individual results remains unchanged.
- Change from using CherryPy web framework to Flask.
- Change from Mutagen to Mutagenx for potential Python 3 support.
- Change from Py.test to Nosetests.
- Lots of changes to potentially support Python 3.3 and 3.4 including use of the `six` library and testing those versions on Travis CI.
- Include reference Fabric deploy script.
- Include reference Gunicorn, uWSGI, and Supervisor configurations.

1.9.7 0.2.25 - 2014-03-19

- License change from Apache 2 to MIT
- Unit test coverage improvements
- Removed server status page
- Remove dependency on the daemon library

- Various code quality improvements

1.9.8 0.2.24 - 2013-08-19

- Fix bug in setup.py that prevented installation in Python 2.6
- Unit test coverage improvements
- Testing infrastructure improvements (Tox, Travis CI)
- Documentation for development environment setup
- Various typo and documentation updates

1.9.9 0.2.23 - 2013-06-17

- Changes to the names of API errors and setting HTTP statuses correctly
- Sample deploy and init scripts for the Avalon Music Server
- Testability improvements for the avalon.cache layer
- Documentation improvements

1.9.10 0.2.22 - 2013-05-20

- Handle database errors during rescan better
- Various code quality improvements
- Improved test coverage

1.9.11 0.2.21 - 2013-02-18

- Bug fixes for the /heartbeat endpoint
- JSON responses now set the correct encoding (UTF-8)
- Improved test coverage

1.9.12 0.2.20 - 2013-02-02

- Updates to status page to use Twitter Bootstrap
- Packaging fixes

1.9.13 0.2.15 - 2013-01-30

- Changed to Apache license 2.0 instead of FreeBSD license
- Updated copyright for 2013

1.9.14 0.2.14 - 2013-01-21

- Text searching using a Trie for faster matching
- Documentation improvements

1.9.15 0.2.13 - 2013-01-10

- Unicode code folding for better search results
- Beginnings of a test suite for the supporting library
- Documentation links to reference server installation

1.9.16 0.2.12 - 2012-12-28

- Text searching functionality via 'query' param for albums, artists, genres, and songs endpoints
- Documentation updates for installation

1.9.17 0.2.11 - 2012-12-23

- Refactor avalon.scan into avalon.tags package
- Switch to use Mutagen by default instead of TagPy
- Allow avalon.tags package to fall back to TagPy if Mutagen isn't installed

1.9.18 0.2.10 - 2012-12-17

- Fix build dependencies and remove setuptools/distribute requirement

1.9.19 0.2.9 - 2012-12-17

- Minor documentation updates

1.9.20 0.2.8 - 2012-12-15

- Updates to the build process

1.9.21 0.2.5 - 2012-12-13

- Packaging fixes

1.9.22 0.2.0 - 2012-12-13

- **Breaking change**: Use of UUIDs for stable IDs for albums, artists, genres, and songs
- Documentation improvements
- Ordering, limit, and offset parameter support

1.9.23 0.1.0 - 2012-05-20

- Initial release

CHAPTER 2

Indices and tables

- genindex
- modindex
- search