Neutrino Audiomatic Documentation

Release 1.0

Neutrino

Jun 17, 2017

Contents

Audiomatic provides power of professional music studio as a service. Lightning fast generation of waveform? Check! Duration of audio file in millisecond precision? No problem. Conversion of any obscure format to web-ready mp3? 3.. 2... 1... Done! Say 'no' to hours of googling, installing hundreds of plugins or obscure software. Leave these bad parts of audio editing to us, so you have time for the important stuff.

Audiomatic is accessible via an API and has supported client libraries for Ruby. More coming soon.

CHAPTER 1

Contents

HTTP API

Scheduling file processing requires POSTing two parameters to audiomatic api: - url - location of processed file, - callback_url - location where results of analysis should be returned as JSON.

Url depends on desired processor. Root of API2.0 requests: - https://audiomatic.radiokitapp.org/api/process/v2.0 should be concatenated with processor's path:

Processors	Path
duration:	"/analysis/audio/duration",
replaygain:	"/analysis/audio/replaygain",
tags:	"/analysis/audio/tags",
webbrowser:	"/transcode/audio/webbrowser",
waveform:	"/visualisation/audio/waveform"

In short, to obtain duration, POST to: - https://audiomatic.radiokitapp.org/api/process/v2.0/analysis/audio/duration

JSON body: {

"callback_url":"https://example.org/callback", "url":"https://example.org/file.mp3"

}

Don't forget about authentication. Use Basic Auth with login and token you obtained during registration. You can use either Authorization Header or URL encoding. For details, see https://en.wikipedia.org/wiki/Basic_access_authentication#Client_side

NeutrinoAudiomatic Response

<!— TODO: Describe both immediate response, with reference, and give specific examples of every processor. Write about response audiomatic expects. –> After sends request to NeutrinoAudiomatic, server answers twice. First response will be sent immediately and it will contain status of request. Second, will be sent results of analyze.

First response:

If status is 202, mean that NeutrinoAudiomatic get your request and start working on it. Body of successful response contain reference number which is unique attribute of single analyze. Otherwise, something went wrong.

Seccond response:

Results are in json similar to:

{“result”=>[{“value”=>”122323”, “key”=>”duration”}]} # => duration

{“result”=>[{“value”=>”AC/DC”, “key”=>”artist”}, {“value”=>”T.N.T”, “key”=>”title”}]} # => tags

{“result”=>[{“value”=>”http://www.example.org/file.mp3”, “key”=>”webbrowser”}]} # => webbrowser

{“result”=>[{“value”=>0.05550943315029144, “key”=>”replaygain-track-peak”}, {“value”=>20.32999999999999, “key”=>”replaygain-track-gain”}, {“value”=>89, “key”=>”replaygain-reference-level”}]} # => replay gain

{“result”=>[{“value”=>”http://www.example.org/file.png”, “key”=>”waveform”}]} # => waveform

Processor	Results
duration	“duration” [ms]
replaygain	“replaygain-track-peak” “replaygain-track-gain” “replaygain-reference-level”
tags	all tags of audio file
webbrowser	url to mp3 file
waveform	url to png file

NeutrinoAudiomaticRails

Ruby on Rails applications will need to add the following entry into their `Gemfile` specifying the Audiomatic client library.

```
gem 'audiomatic-rails'
```

Update application dependencies with bundler.

Configuration

For heroku users, configuration is really easy. Just install Neutrino_Audiomatic (url) plugin and paste code-block below to `config/initializer/audiomatic_config.rb`:

```
Audiomatic.config do |c|
  c.processor_host = ENV['AUDIOMATIC_URL']
  c.audiomatic_secret = ENV['AUDIOMATIC_SHARED_SECRET']
end
```

you can also provide your own unique url:

```
Audiomatic.config do |c|
  c.processor_host = "https://login:password@audiomatic.radiokitapp.org/api/process"
  c.audiomatic_secret = ENV['AUDIOMATIC_SHARED_SECRET']
end
```

also you have to set the routes default_url_options in environments/production.rb

Getting start

1. Store file which you want to analyze. This gem provides method *store* which simplifies it. Example of controller:

```
def create
  @audio_file = AudioFile.new
  @audio_file.store params[:audio_file][:audio]
  @audio_file.save
end
```

2. Include AudiomaticRails::Model in your ActiveRecord Model

```
class YourModel < ApplicationRecord

  include AudiomaticRails::Model

end
```

3. Configure processors. Set field where result should be saved (don't forget to adding column to database):

```
# model should have 'duration' field
analyze :duration, [:duration]

# or (if file is named different then result)
analyze :duration {duration: {field: :duration_field}}
```

4. Keep adding processors:

```
analyze :duration, {duration: {transformation: to_second= }}
analyze :tags, [:artist, :title]
analyze :waveform, mount_on: :waveform
analyze :browser, mount_on: :mp3_file
analyze :replaygain, save_to: replaygain_method=
```

5.Last step: tell Audiomatic where results should be sent back. Add to your routes.rb:

```
# config/routes.rb

Rails.application.routes.draw do

  audiomatic_for(YourModel)

end
```

And that's it!

Sets analyze

NeutrinoAudiomaticRails gives many possibilities of processors configuration which will be done on your files. Gem has a lot of default configuration, but also gives capabilities to customize it. Method *analyze* describes which result we want to get, and how the result is handled.

Saving directly to ActiveRecord field

The most default option: Select processor and put result to Array. It saves results to fields in yourModel (name of result must be the same as name of field)

```
  #processor #result
analyze :duration, [:duration]
```

To customize this strategy you can use Hash instead of Array, and select extra options:

- :field - allows select field for result
- :transformation - allows convert result before saving to db, transformations methods have to be in Array

Example:

```
analyze :replaygain, {'replaygain-track-peak': {field: :replaygain_track_peak},
↪'replaygain-track-gain': {field: :'replaygain_track_gain', transformation: [:to_s=]}
↪}

analyze :duration, {duration: {transformation: [:to_s=]}}
```

Handle answer

If you want to get all results from processor and save or interpret it on your own, you have to use option *:save_to* which lets you write own method that would get result and do everything you want. Example:

```
    #processor #result
analyze :tags, save_to: :tags=

def tags= result # result is json {'key': name, 'value': value }
  #your code...
end
```

With CarrierWave

Because in some cases NeutrinoAudiomatic returns file, neutrino_audiomatic_rails provides easy way to store this file in your Uploader. Example:

```
mount_uploader :waveform, WaveformUploader
  mount_uploader :mp3_file, Mp3Uploader

  analyze :webbrowser, mount_on: :mp3_file
  analyze :waveform, mount_on: :waveform
```

It means *mount_on: :uploader* is equl to *self.uploader.store! file*. Now, results are in your Uploader.

Stores files

To analyze file NeutrinoAudiomatic needs an url to your file. Neutrino_audiomatic_rails provides *store* method which saves your file in tmp folder and creates routes to get it. It will be used during sending request to NeutrinoAudiomatic.

```
def create
  @audio_file = AudioFile.new
  @audio_file.store params[:audio_file][:audio]
  @audio_file.save
end
```

If you already have audio file stored. you can provide your own method which would return url to file to analyze.

```
audiomatic_file_url :set_url
def set_url
```

```
  #your code
end
```

NeutrinoAudiomatic errors

Sending request to NeutrinoAudiomac starts after saving instance of yourModel. While sending request to NeutrinoAudiomatic something can go wrong. In default configuration if server sends response with status diffrent then 202, method in gem raises exception, rollbacks transaction and method *save* returns false.

Second strategy allows you write your own reaction on error. All you have to do is to write method and set it as audiomatic_error. This method needs to have one argument. If your method returns true, then gem uses ActiveJob and tries sending request to server again after 15 minutes. If method returns false then gem raises excepction and deletes transaction.

```
audiomatic_error :handle_error=

def handle_error= error
  #your code
end
```

Response authentication

Response sent from NeutrinoAudiomatic can be verified by calculation a signature.

All response contain *HTTP_X_NEUTRINO_AUDIOMATIC_HMAC_SHA256* header which is generated using the app's shared secret and body of response.

To verify that the request come from NeutrinoAudiomatic, compute the HMAC digest and then compere value and *HTTP_X_NEUTRINO_AUDIOMATIC_HMAC_SHA256* header. If they the same, you can be sure the response come from NeutrinoAudiomatic.

Simple example:

```
def self.verify_neutrino(data, hmac_header)
  secret = ENV['AUDIOMATIC_SHARED_SECRET']
  calculated_hmac = Digest::SHA1.hexdigest(data + ":" + secret)
  calculated_hmac == hmac_header
end

def self.verify_response request
  request.body.rewind
  data = request.body.read
  verified = verify_neutrino(data, request.headers["HTTP_X_NEUTRINO_AUDIOMATIC_HMAC_
↪SHA256"])
end
```

Gem neutrino_audiomatic_rails authenticates response for you

Heroku Add-on

Provisioning the add-on

Audiomatic can be attached to a Heroku application via the CLI:

Once Neutrino Audiomatic has been added a `NEUTRINO_AUDIOMATIC_URL` setting will be available in the app configuration and will contain the canonical URL used to access the newly provisioned Neutrino Audiomatic service instance. This can be confirmed using the `heroku config:get` command.

After installing Neutrino Audiomatic the application should be configured to fully integrate with the add-on.

Using with Rails 4.x, 5.x

Ruby on Rails applications will need to add the following entry into their `Gemfile` specifying the Neutrino Audiomatic client library.

```
gem 'neutrino-audiomatic-rails'
```

Update application dependencies with bundler.

Configuration

For heroku users, configuration is really easy.Just install Neutrino Audiomatic (url) plugin and paste code-block below to `config/initializer/neutrino_audiomatic_config.rb`:

```
NeutrinoAudiomatic.config do |c|

  c.processor_host = ENV['NEUTRINO_AUDIOMATIC_URL']

end
```

You can also provide your own unique url:

```
NeutrinoAudiomatic.config do |c|

  c.processor_host = "https://login:password@neutrino-audiomatic.radiokitapp.org/api/
↪process"

end
```

Using with ActiveRecord

1. Include NeutrinoAudiomaticRails::Model in your Model

```
class YourModel < ApplicationRecord

  include NeutrinoAudiomaticRails::Model

end
```

2. Tell Neutrino Audiomatic where your files are stored. Just pass method name to neutrino_audiomatic_file_url

```
neutrino_audiomatic_file_url :file_url



    # Example method returning URL

def file_url
```

```
  "www.example.org/files/#{file.slug}.mp3"

end
```

3. Configure processors. Set field where result should be saved (don't forget to about adding column to database):

```
      # model should have 'duration_value' field

analyze :duration, [:duration_value]
```

4. Keep adding processors:

```
analyze :duration, [:duration]

analyze :tags, [:artist, :title]
```

5. Last step: tell Neutrino Audiomatic where results should be sent back. Add to your routes.rb:

```
# config/routes.rb


Rails.application.routes.draw do

  neutrino_audiomatic_for(YourModel)

end
```

And that's it!

In short:

```
config/initializer/neutrino_audiomatic\_config.rb

NeutrinoAudiomatic.config do \|c\|

c.processor\_host = ENV[‘NEUTRINO_AUDIOMATIC\_URL’]

end

config/routes.rb


Rails.application.routes.draw do

neutrino-audiomatic\_for(YourModel)

end

your\_model.rb
```

Support

If you need any support while using Neutrino Audiomatic, please do not hestitate to contact us at admin@neutrino.audio