larigira Documentation

Release 1.3.3

boyska

Jun 21, 2020

Contents

Contents:

CHAPTER 1

About

1.1 What does it do

larigira integrates with MPD (Music Player Daemon) and prevents your playlist from running empty. It also has powerful support for "events": audio that must be played at some time.

1.2 Features

- Simple to install
- WebUI
- modular event system

1.3 Architecture

larigira delegates all the music playing business to MPD. It relies on `tinydb` as a db: it's actually just a json file, to achieve simplicity and flexibility. By default it is stored in `~/.config/larigira/db.json`

1.4 Code structure and core concepts

The code is heavily based on gevent: almost everything is a greenlet.

alarm An alarm is a specification of timings. It is "something that can generate times". For example `{ 'kind': 'single', 'timestamp': 1234567890 }` generates a single time (February 14, 2009 00:31:00), while `{ 'kind': 'frequency', 'interval': 10, 'start': 1234567890 }` generates infinite times, one every 10 seconds, starting from February 14, 2009 00:31:00.

action An action is a specification of audio. It is "something that can generate a list of audio files". For example, `{ 'kind': 'randomdir', 'paths': ['/my/dir', '/other/path'] }` will pick a random file from one of the two paths.

event An event is an alarm plus a list of actions. At given times, do those things

The main object is *larigira.mpc.Controller*, which in turn uses *larigira.mpc.Player* to control MPD. How does it know what to do? there are two main flows: the continous playlist filling and the alarm system.

1.4.1 Continous playlist

larigira.mpc.Controller has a "child" called `larigira.mpc.MpcWatcher`. It watches for events on the playlist; when the playlist is changed it notifies Controller, which in turn will check if the playlist has enough songs. If that's the case, it will run an audiogenerator, and add the resulting audio at the bottom of the playlist.

1.4.2 Alarm system

There is a DB. The lowest level is handled by TinyDB. `larigira.event.EventModel` is a thin layer on it, providing more abstract functions.

There is a *Monitor*, which is something that, well, "monitors" the DB and schedule events appropriately. It will check alarms every `EVENT_TICK_SECS` seconds, or when larigira received a `SIGALRM` (so `pkill -ALRM larigira` might be useful for you).

You can view scheduled events using the web interface, at `/view/status/running`. Please note that you will only see *scheduled* events, which are events that will soon be added to playlist. That page will not give you information about events that will run in more than `2 * EVENT_TICK_SECS` seconds (by default, this amounts to 1 minute).

CHAPTER 2

Installation

Installing larigira is quite simple. You can install latest version from PyPI using `pip install larigira`. Or you can `git clone https://git.lattuga.net/boyska/larigira.git` and run `python setup.py install`. As always, the usage of a virtualenv is recommended.

Python greater or equal than 3.4 is supported.

2.1 Configuration

larigira use MPD in a peculiar way. It can use mpd internal library, but it can also work with "regular" local files, outside of the library. To do so, however, it requires you to connect to MPD through the UNIX socket instead of the TCP port.

So how to create this setup? inside `~/.mpdconf`, add the following line:

```
bind_to_address "~/.mpd/socket"
```

For larigira, you need to set the `MPD_HOST` environment variable to `$HOME/.mpd/socket`. If you don't do this, you'll find many lines like these in the logs:

```
15:37:10|ERROR[Player:93] Cannot insert song file:///tmp/larigira.1002/audiogen-
↪randomdir-8eoklcee.mp3
Traceback (most recent call last):
  File "/home/user/my/ror/larigira/larigira/mpc.py", line 91, in enqueue
    mpd_client.addid(uri, insert_pos)
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 629, in decorator
    return wrapper(self, name, args, bound_decorator(self, returnValue))
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 254, in _execute
    return retval()
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 623, in decorator
```

(continues on next page)

(continued from previous page)

```
    return function(self, *args, **kwargs)
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 384, in _fetch_item
    pairs = list(self._read_pairs())
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 311, in _read_pairs
    pair = self._read_pair(separator)
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 302, in _read_pair
    line = self._read_line()
  File "/home/user/.virtualenvs/larigira3/lib/python3.5/site-packages/python_mpd2-0.5.
↪5-py3.5.egg/mpd.py", line 291, in _read_line
    raise CommandError(error)
mpd.CommandError: [400] {addid} Access denied
```

2.1.1 Options

Options will be presented "grouped" to make them more easily understandable. This is just for documentation purposes, they don't need to be divided in the configuration.

Options values are assumed to be JSON formatted. For strings and numbers, this makes no difference. For booleans, this means that the only valid values are `true` and `false`. For lists and objects, read a JSON reference or learn from examples.

To set an option, you must set an environment variable with the name `LARIGIRA_OPTIONNAME`.

General

MPD_HOST This option let you set the way to connect to your MPD server. Please remind that for complete larigira operatoin you should set the path to a UNIX domain socket, not `localhost`

MPD_PORT If you are not using a socket, but a TCP address (which is *not* suggested), this is how you can specify the port.

DEBUG you can set it to `true` or `false`. Defaults to `false`.

TMPDIR The base for larigira tmpdir. Please note that larigira will create its own directory inside this temporary directory. This defaults to the system-wide `$TMPDIR`, or to `/tmp/` if not `TMPDIR` is not set. Choose it wisely, keeping in mind that in this directory a lot of cache files will be stored, and could therefore require hundreds of MB.

Events

CONTINOUS_AUDIOSPEC when the playlist is too short, larigira picks something new. How? this is controlled by this variable. This variable should be set to the JSON representation of an audiospec describing how to generate new audios. The default is `{"kind": "mpd", "howmany": 1}`. You could, for example, change it to `{ "kind": "randomdir", "paths": ["/var/music"], "howmany": 10}` or to `{"kind": "mpd", "howmany": 5, "prefix": "background"}`.

EVENT_FILTERS See *EventFilters*

LOG_CONFIG Path to an INI-formatted file to configure logging. See python logging documentation

Internals

These are options you probably don't want to change, unless you want to debug

CACHING_TIME larigira needs an estimate on how much time an audiogenerator will need. This option sets this. The default is 10 (seconds). If you set it too low, the events will be scheduled with some delay

CHECK_SECS The interval (in seconds) that will trigger a check for the playlist length. Set it too low, and you'll be consuming resource with no usage. Set it too high and you might miss some moment of playlist shortage. The default is 20

EVENT_TICK_SECS The interval (in seconds) that will trigger a check on the events to see if there's something to schedule. This also determines how much time in advance you can add/delete/change an event for the change to be effective.

SCRIPTS_PATH This options controls the path where scripts will be looked for. This is a single path, not a list. The default is `$XDG_CONFIG_DIR/larigira/scripts/` which might boil down to `$HOME/.config/larigira/scripts/`

DB_URI The path to the events database. The default is `$XDG_CONFIG_DIR/larigira/db.json`

MPD_WAIT_START When larigira starts, it will try to connect to MPD. If MPD doesn't look ready, larigira will wait until it is. This makes larigira depend on MPD. However, this also makes it easier to run larigira at boot without complex dependency on MPD to be fully started and listening. You can disable this behavior setting this to `false`

MPD_WAIT_START_RETRYSECS The behavior described for the previous option requires polling. This variable lets you customize the polling frequency, expressed in seconds. The default is `5`.

Web interface

FILE_PATH_SUGGESTION A list of paths. Those paths will be scanned for suggestions in audiogenerator forms.

UI_CALENDAR_FREQUENCY_THRESHOLD The "calendar" view in the UI will omit events that occur too frequently, to avoid noise. This variable is the threshold, in seconds. Defaults to 4 hours.

BOOTSTRAP_SERVE_LOCAL larigira can serve every js and css by itself. However, you might like to make the user download standard libraries from CDNs. In that case, set this variable to `false`.

Localization

Localization support in larigira is basic to say the least. However, something can be done. Dates (not consistently! sigh) are formatted according to locale.

LARIGIRA_BABEL_DEFAULT_LOCALE The short language code you want to use when the user doesn't specify any. Values are in the form `it`, `en`, `en-US`...

LARIGIRA_UI_CALENDAR_DATE_FMT The format to show in `/db/calendar` page. The format is specified here. Default is `medium`.

As an example, `eee dd LLL` will show `Sun 10 Mar` for english, and `dom 10 mar` for italian.

CHAPTER 3

Timegenerators

A “timegenerator” is something that describe _when_ you want to do something.

There are 3 kinds of timegenerators in larigira:

3.1 single

With single, you just set a single time the event should happen.

3.2 frequency

Sometimes you have something happening very regularly. For example you might want a jingle every 30minutes, or a specific show every friday at 8PM.

3.3 cron

Sometimes your specification is very complex.

For example, you might want a specific message to be sent at 10:00, 13:00, 15:00 and 18:00, but only during working days. That would be hard to do with a single *frequency* event, so you could use a cron

```
0 10,13,15,18 * * 1-5
```

I know, that’s very difficult, but is still a possibility you have. If you want to learn how to write crons, check this helpful site.

CHAPTER 4

Audiogenerators

4.1 mpdrandom

picks `howmany` song randomly from your mpd library. It follows this strategy: it picks `howmany` artists from your MPD library, then picks a random song for each one

4.2 randomdir

Given a directory `path`, scan it recursively, then picks `howmany` random files. Only files whose filename ends in mp3/ogg/oga/wav are considered . Files are copied to `TMPDIR` before being returned.

4.3 static

That simple: every element in `paths` is returned. Before doing so, they are copied to `TMPDIR`.

4.4 http

Given a sequence of `urls`, downloads each one and enqueue it.

This is **not** suitable for streams (a fundamental limitation of larigira), only for audio files.

4.5 mostrecent

It is similar to randomdir, but instead of picking randomly, picks the most recent file (according to the ctime).

4.6 script

see *Write your own audiogenerator*

CHAPTER 5

EventFilters

What is an event filter? It is a mechanism to filter songs before adding them to the playlist. Why is it any useful? To implement something better than a priority system!

Real world example: you have many events. Some are very long ("shows"), some are short ("jingle"). If you have a long show of 2hours, and a jingle every 15minutes, then at the end of your jingle you'll find 8 jingle stacked. That's bad! How to fix it? There are many solutions, none of them is very general or suits every need. EventFilter is a mechanism to have multiple filters running serially to find a reason to exclude an event.

5.1 Using an event filter

larigira provides some basic filter. A simple example can be "maxwait". The principle is just "don't add new events if the current playing song still needs more than X seconds to finish". Setting this to, for example, 15 minutes, would have found only one jingle in the previous example. Great job!

However, it wouldn't be very kind of long shows. If you have two long shows, each 2h long, scheduled let's say at 8:00 and 9:30. Now the second show won't start, because there is 30min remaining, and the maxwait is set to 15min. If you have this solution, maybe *percentwait* will better suit your needs. Set this to 200%, and a jingle can wait up to twice its own duration. This is like setting maxtime=4hours for the long shows, but maxtime=1minute for jingles. Nice!

5.2 Examples

5.2.1 Fixed wait

This snippet will configure larigira to wait a maximum of 5 minutes:

```
LARIGIRA_EVENT_FILTERS='["maxwait"]'
LARIGIRA_EF_MAXWAIT_SEC=300
```

(60*5 = 300seconds). This will basically cancel any event that finds a currently playing audio not at its end. It can be recommended if you have very accurate timing or don't care about "losing" shows because the one preceding it lasted for too long.

5.2.2 Relative wait

This will configure larigira so that a 1-minute jingle can wait for a maximum of 4minutes, while a 20minutes event can wait up to 1h20min:

```
LARIGIRA_EVENT_FILTERS='["percentwait"]'
LARIGIRA_EF_MAXWAIT_PERC=400
```

In fact, `400` means 400%, that is 4 times the lenght of what we're adding

5.3 Write your own

You probably have some very strange usecase here. Things to decide based on your custom naming convention, time of the day, moon phase, whatever. So you can and should write your own eventfilter. It often boils down to very simple python functions and configuration of an entrypoint for the *larigira.eventfilter* entrypoint.

CHAPTER 6

Write your own audiogenerator

It's easy to have situations where you want to run your own custom audiogenerator. Fetching music with some very custom logic? Checking RSS feeds? Whatever. There are two main strategies to add audiogenerators to `larigira`: scripts and setuptools entrypoints. Scripts are easier. setuptools entrypoints are somewhat "tidier". We suggest writing entrypoints if you want to distribute your code somehow. If your script is so custom that only makes sense to you, a script can be enough.

6.1 writing a script

a script is an executable file. The programming language is completely to your choice: bash, perl, python, compiled C code, `larigira` doesn't care. It must be in a specific directory (`~/.config/larigira/scripts/`), be executable (as in `chmod +x`), and be owned by the same user running larigira.

When executed, this script must output URIs, one per line. Only UTF-8 is supported as encoding. The script should expect limited environment (for security reaons). Please also see *`larigira.audiogen_script`* for more details.

6.2 writing an audiogen entrypoint

TODO

CHAPTER 7

Debugging and inspecting

The highly asynchronous nature of `larigira` might lead to difficulty in debugging. However, `larigira` has some features to help you with debugging.

7.1 Debugging options

First of all, you might want to run larigira with the environment variable `LARIGIRA_DEBUG` set to `true`. `env LARIGIRA_DEBUG=true larigira` will do.

With this on, message logging is much more verbose. Please observe that log messages provide information about the logger name from which the message originated: this is typically the class name of the object.

7.2 Debug API

`larigira` also provides HTTP API to help you with debug: `/api/debug/running`, for example, provides detailed information about running greenlets, with several representations:

- the `audiogens` object will contain informations about greenlets associated with *scheduled* events. Here you can see its `audiospec`, `timespec`, and many other details
- the `greenlets` object provides a simple list of every greenlets. Associated with every greenlet there is as much information as possible: object name, documentation, etc.
- the `greenlets_tree` has the same information as `greenlets`, but is shown as a tree: this is often easier to understand. For example, the `Controller` greenlet should have three children (`MpcWatcher`, `Timer` and `Monitor`).

A user-friendly, but more limited, visualization is available at `/view/status/running`

7.3 Signals

When `larigira` receives the `ALRM` signal, three things happen: the playlist length is checked (as if `CHECK_SECS` passed); the event system "ticks" (as if `EVENT_TICK_SECS` passed); the DB is reloaded from disk. This is useful if you want to debug the event system, or if you manually changed the data on disk. Please note that the event system will automatically reload the DB from disk when appropriate. However, the WebUI will not, so you might have a misleading `/db/list` page; send an `ALRM` in this case.

The same effect can be triggered performing an HTTP `GET /rpc/refresh`.

CHAPTER 8

larigira

8.1 larigira package

8.1.1 Subpackages

larigira.filters package

Submodules

larigira.filters.basic module

larigira.filters.basic.**get_duration**(*path*)
get track duration in seconds

larigira.filters.basic.**maxwait**(*songs*, *context*, *conf*)

larigira.filters.basic.**percentwait**(*songs*, *context*, *conf*, *getdur=<function get_duration>*)
Similar to maxwait, but the maximum waiting time is proportional to the duration of the audio we're going to add.

This filter observe the EF_MAXWAIT_PERC variable. The variable must be an integer representing the percentual. If the variable is 0 or unset, the filter will not run.

For example, if the currently playing track still has 1 minute to go and we are adding a jingle of 40seconds, then if EF_MAXWAIT_PERC==200 the audio will be added (40s*200% = 1m20s) while if EF_MAXWAIT_PERC==100 it will be filtered out.

Module contents

8.1.2 Submodules

8.1.3 larigira.audioform_http module

class larigira.audioform_http.**AudioForm**(**args*, ***kwargs*)
Bases: flask_wtf.form.Form

nick = <UnboundField(StringField, ('Audio nick',), {'validators': [<wtforms.validator

populate_from_audiospec(*audiospec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

urls = <UnboundField(StringField, ('URLs',), {'validators': [<wtforms.validators.Requ

larigira.audioform_http.**audio_receive**(*form*)

8.1.4 larigira.audioform_mostrecent module

class larigira.audioform_mostrecent.**AudioForm**(**args*, ***kwargs*)
Bases: flask_wtf.form.Form

maxage = <UnboundField(StringField, ('Max age',), {'validators': [<wtforms.validators

nick = <UnboundField(StringField, ('Audio nick',), {'validators': [<wtforms.validator

path = <UnboundField(AutocompleteStringField, ('dl-suggested-dirs', 'Path'), {'validat

populate_from_audiospec(*audiospec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

validate_maxage(*field*)

larigira.audioform_mostrecent.**audio_receive**(*form*)

8.1.5 larigira.audioform_randomdir module

class larigira.audioform_randomdir.**Form**(**args*, ***kwargs*)
Bases: flask_wtf.form.Form

howmany = <UnboundField(IntegerField, ('Number',), {'validators': [<wtforms.validator

nick = <UnboundField(StringField, ('Audio nick',), {'validators': [<wtforms.validator

path = <UnboundField(AutocompleteStringField, ('dl-suggested-dirs', 'Path'), {'validat

populate_from_audiospec(*audiospec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

larigira.audioform_randomdir.**receive**(*form*)

8.1.6 larigira.audioform_script module

class larigira.audioform_script.**ScriptAudioForm**(**args*, ***kwargs*)
Bases: flask_wtf.form.Form

args = <UnboundField(StringField, ('Arguments',), {'description': 'arguments, separat

name = <UnboundField(AutocompleteStringField, ('dl-suggested-scripts', 'Name'), {'vali

nick = <UnboundField(StringField, ('Audio nick',), {'validators': [<wtforms.validator

populate_from_audiospec(*audiospec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

validate_name(*field*)

larigira.audioform_script.**scriptaudio_receive**(*form*)

8.1.7 larigira.audioform_static module

class larigira.audioform_static.**StaticAudioForm**(**args*, ***kwargs*)
Bases: flask_wtf.form.Form

nick = <UnboundField(StringField, ('Audio nick',), {'validators': [<wtforms.validator

path = <UnboundField(AutocompleteStringField, ('dl-suggested-files', 'Path'), {'valida

populate_from_audiospec(*audiospec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

larigira.audioform_static.**staticaudio_receive**(*form*)

8.1.8 larigira.audiogen module

larigira.audiogen.**audiogenerate**(*spec*)

larigira.audiogen.**check_spec**(*spec*)

larigira.audiogen.**get_audiogenerator**(*kind*)
Messes with entrypoints to return an audiogenerator function

larigira.audiogen.**get_parser**()

larigira.audiogen.**main**()
Main function for the "larigira-audiogen" executable

larigira.audiogen.**read_spec**(*fname*)

8.1.9 larigira.audiogen_http module

larigira.audiogen_http.**generate**(*spec*)
resolves audiospec-static

Recognized argument is "paths" (list of static paths)

larigira.audiogen_http.**put**(*url*, *destdir=None*, *copy=False*)

8.1.10 larigira.audiogen_mostrecent module

larigira.audiogen_mostrecent.**generate**(*spec*)
resolves audiospec-randomdir

Recognized arguments:

- path [mandatory] source dir
- maxage [default=ignored] max age of audio files to pick

larigira.audiogen_mostrecent.**get_mtime**(*fname*)

larigira.audiogen_mostrecent.**recent_choose**(*paths*, *howmany*, *minepoch*)

8.1.11 larigira.audiogen_mpdrandom module

larigira.audiogen_mpdrandom.**generate_by_artist**(*spec*)
choose HOWMANY random artists, and for each one choose a random song

8.1.12 larigira.audiogen_randomdir module

larigira.audiogen_randomdir.**candidates**(*paths*)

larigira.audiogen_randomdir.**generate**(*spec*)
resolves audiospec-randomdir

Recognized arguments:

- paths [mandatory] list of source paths
- howmany [default=1] number of audio files to pick

8.1.13 larigira.audiogen_script module

script audiogenerator: uses an external program to generate audio URIs

a script can be any valid executable in $XDG_CONFIG_DIR/larigira/scripts/<name>; for security reasons, it must be executable and owned by the current user. The audiospec can specify arguments to the script, while the environment cannot be customized (again, this is for security reasons).

The script should assume a minimal environment, and being run from /. It must output one URI per line; please remember that URI must be understood from mpd, so file paths are not valid; file:///file/path.ogg is a valid URI instead. The output MUST be UTF-8-encoded. Empty lines will be skipped. stderr will be logged, so please be careful. any non-zero exit code will result in no files being added.and an exception being logged.

larigira.audiogen_script.**generate**(*spec*)

Recognized arguments (fields in spec):

- name [mandatory] script name
- args [default=empty] arguments, colon-separated

8.1.14 larigira.audiogen_static module

larigira.audiogen_static.**generate**(*spec*)
resolves audiospec-static

Recognized argument is "paths" (list of static paths)

8.1.15 larigira.config module

Taken from flask-appconfig

larigira.config.**from_envvars**(*prefix=None*, *envvars=None*, *as_json=True*)
Load environment variables in a dictionary

Values are parsed as JSON. If parsing fails with a ValueError, values are instead used as verbatim strings.

Parameters

- **prefix** – If `None` is passed as envvars, all variables from `environ` starting with this prefix are imported. The prefix is stripped upon import.
- **envvars** – A dictionary of mappings of environment-variable-names to Flask configuration names. If a list is passed instead, names are mapped 1:1. If `None`, see prefix argument.
- **as_json** – If False, values will not be parsed as JSON first.

larigira.config.**get_conf**(*prefix='LARIGIRA_'*)
This is where everyone should get configuration from

8.1.16 larigira.db module

class larigira.db.**EventModel**(*uri*)
Bases: `object`

add_action(*action*)

add_alarm(*alarm*)

add_event(*alarm*, *actions*)

delete_action(*actionid*)

delete_alarm(*alarmid*)

get_action_by_id(*action_id*)

get_actions_by_alarm(*alarm*)

get_alarm_by_id(*alarm_id*)

get_all_actions()

get_all_alarms()

get_all_alarms_expanded()

reload()

update_action(*actionid*, *new_fields={}*)

update_alarm(*alarmid*, *new_fields={}*)

8.1.17 larigira.entrypoints_utils module

larigira.entrypoints_utils.**get_avail_entrypoints**(*group*)

larigira.entrypoints_utils.**get_one_entrypoint**(*group*, *kind*)
Messes with entrypoints to return an entrypoint of a given group/kind

8.1.18 larigira.event module

class larigira.event.**Monitor**(*parent_queue*, *conf*)
Bases: *larigira.eventutils.ParentedLet*

Manages timegenerators and audiogenerators for DB events

The mechanism is partially based on ticks, partially on scheduled actions. Ticks are emitted periodically; at every tick, *on_tick* checks if any event is "near enough". If an event is near enough, it is "*scheduled*": a greenlet is run which will wait for the right time, then generate the audio, then submit to Controller.

The tick mechanism allows for events to be changed on disk: if everything was scheduled immediately, no further changes would be possible. The scheduling mechanism allows for more precision, catching exactly the right time. Being accurate only with ticks would have required very frequent ticks, which is cpu-intensive.

on_tick()
this is called every EVENT_TICK_SECS. Checks every event in the DB (which might be slightly CPU-intensive, so it is advisable to run it in its own greenlet); if the event is "near enough", schedule it; if it is too far, or already expired, ignore it.

process_action(*timespec*, *audiospecs*)
Generate audio and submit it to Controller

schedule(*timespec*, *audiospecs*, *delta=None*)
prepare an event to be run at a specified time with the specified actions; the DB won't be read anymore after this call.

This means that this call should not be done too early, or any update to the DB will be ignored.

8.1.19 larigira.event_manage module

larigira.event_manage.**main**()

larigira.event_manage.**main_add**(*args*)

larigira.event_manage.**main_getaction**(*args*)

larigira.event_manage.**main_list**(*args*)

8.1.20 larigira.eventutils module

class larigira.eventutils.**ParentedLet**(*queue*)
Bases: gevent._greenlet.Greenlet

ParentedLet is just a helper subclass that will help you when your greenlet main duty is to "signal" things to a parent_queue.

It won't save you much code, but "standardize" messages and make explicit the role of that greenlet

parent_msg(*kind*, **args*)

send_to_parent(*kind*, **args*)

class larigira.eventutils.**Timer**(*milliseconds*, *queue*)
Bases: *larigira.eventutils.ParentedLet*

continously sleeps some time, then send a "timer" message to parent

do_business()

parent_msg(*kind*, **args*)

8.1.21 larigira.forms module

larigira.forms.**get_audioform**(*kind*)
Messes with entrypoints to return a AudioForm

larigira.forms.**get_timeform**(*kind*)
Messes with entrypoints to return a TimeForm

8.1.22 larigira.formutils module

class larigira.formutils.**AutocompleteStringField**(*datalist*, **args*, ***kwargs*)
Bases: wtforms.fields.core.StringField

class larigira.formutils.**AutocompleteTextInput**(*datalist=None*)
Bases: wtforms.widgets.core.Input

class larigira.formutils.**DateTimeInput**(*input_type=None*)
Bases: wtforms.widgets.core.Input

input_type = 'datetime-local'

class larigira.formutils.**EasyDateTimeField**(*label=None*, *validators=None*, ***kwargs*)
Bases: wtforms.fields.core.Field

a "fork" of DateTimeField which uses HTML5 datetime-local

The format is not customizable, because it is imposed by the HTML5 specification.

This field does not ensure that browser actually supports datetime-local input type, nor does it provide polyfills.

formats = ('%Y-%m-%dT%H:%M:%S', '%Y-%m-%dT%H:%M')

process_formdata(*valuelist*)
Process data received over the wire from a form.

This will be called during form construction with data supplied through the *formdata* argument.

Parameters **valuelist** – A list of strings to process.

widget = <larigira.formutils.DateTimeInput object>

8.1.23 larigira.fsutils module

larigira.fsutils.**is_audio**(*fname*)

larigira.fsutils.**multi_fnmatch**(*fname*, *extensions*)

larigira.fsutils.**scan_dir**(*dirname*, *extension=None*)

larigira.fsutils.**scan_dir_audio**(*dirname*, *extensions=('mp3', 'oga', 'wav', 'ogg')*)

larigira.fsutils.**shortname**(*path*)

8.1.24 larigira.larigira module

This module is for the main application logic

class larigira.larigira.**Larigira**
Bases: object

start()

larigira.larigira.**main**()

larigira.larigira.**on_main_crash**(**args*, ***kwargs*)

larigira.larigira.**sd_notify**(*ready=False*, *status=None*)

8.1.25 larigira.mpc module

class larigira.mpc.**Controller**(*conf*)
Bases: gevent._greenlet.Greenlet

class larigira.mpc.**MPDWatcher**(*queue*, *conf*, *client=None*)
Bases: *larigira.eventutils.ParentedLet*

MPDWatcher notifies parent about any mpd event

do_business()

refresh_client()

class larigira.mpc.**Player**(*conf*)
Bases: object

The player contains different mpd-related methods

check_playlist determines whether the playlist is long enough and run audiogenerator accordingly

enqueue receive audios that have been generated by Monitor and (if filters allow it) enqueue it to MPD playlist

check_playlist()

continous_audiospec

enqueue(*songs*)

enqueue_filter(*songs*)

larigira.mpc.**get_mpd_client**(*conf*)

8.1.26 larigira.test_unused module

8.1.27 larigira.timeform_base module

class larigira.timeform_base.**FrequencyAlarmForm**(**args*, ***kwargs*)
Bases: flask_wtf.form.Form

end = <UnboundField(EasyDateTimeField, ('End date and time',), {'validators': [<wtfor

interval = <UnboundField(StringField, ('Frequency',), {'validators': [<wtforms.valida

nick = <UnboundField(StringField, ('Alarm nick',), {'validators': [<wtforms.validator

populate_from_timespec(*timespec*)

start = <UnboundField(EasyDateTimeField, ('Start date and time',), {'validators': [<w

submit = <UnboundField(SubmitField, ('Submit',), {})>

validate_interval(*field*)

weekdays = <UnboundField(SelectMultipleField, ('Days on which the alarm should be play

class larigira.timeform_base.**SingleAlarmForm**(**args*, ***kwargs*)

Bases: flask_wtf.form.Form

dt = <UnboundField(EasyDateTimeField, ('Date and time',), {'validators': [<wtforms.va

nick = <UnboundField(StringField, ('Alarm nick',), {'validators': [<wtforms.validator

populate_from_timespec(*timespec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

larigira.timeform_base.**frequencyalarm_receive**(*form*)

larigira.timeform_base.**singlealarm_receive**(*form*)

8.1.28 larigira.timeform_cron module

class larigira.timeform_cron.**CronAlarmForm**(**args*, ***kwargs*)

Bases: flask_wtf.form.Form

cron_format = <UnboundField(StringField, ('cron-like format',), {'validators': [<wtfo

exclude = <UnboundField(TextAreaField, ('cron-like format; any matching time will be e

nick = <UnboundField(StringField, ('Alarm nick',), {'validators': [<wtforms.validator

populate_from_timespec(*timespec*)

submit = <UnboundField(SubmitField, ('Submit',), {})>

validate_cron_format(*field*)

validate_exclude(*field*)

larigira.timeform_cron.**cronalarm_receive**(*form*)

8.1.29 larigira.timegen module

main module to read and get informations about alarms

larigira.timegen.**check_spec**(*spec*)

larigira.timegen.**get_parser**()

larigira.timegen.**get_timegenerator**(*kind*)

Messes with entrypoints to return an timegenerator function

larigira.timegen.**main**()

Main function for the "larigira-timegen" executable

larigira.timegen.**read_spec**(*fname*)

larigira.timegen.**timegenerate**(*spec*, *now=None*, *howmany=1*)

8.1.30 larigira.timegen_cron module

class larigira.timegen_cron.**CronAlarm**(*obj*)
Bases: *larigira.timegen_every.Alarm*

description = 'Frequency specified by cron-like format. nerds preferred'

has_ring(*current_time=None*)
returns True IFF the alarm will ring exactly at time

is_excluded(*dt*)

next_ring(*current_time=None*)
if current_time is None, it is now()

returns the next time it will ring; or None if it will not anymore

8.1.31 larigira.timegen_every module

class larigira.timegen_every.**Alarm**
Bases: object

all_rings(*current_time=None*)
all future rings this, of course, is an iterator (they could be infinite)

has_ring(*time=None*)
returns True IFF the alarm will ring exactly at time

next_ring(*current_time=None*)
if current_time is None, it is now()

returns the next time it will ring; or None if it will not anymore

class larigira.timegen_every.**FrequencyAlarm**(*obj*)
Bases: *larigira.timegen_every.Alarm*

rings on {t | exists a k integer >= 0 s.t. t = start+k*t, start<t<end}

description = 'Events at a specified frequency. Example: every 30minutes'

has_ring(*current_time=None*)
returns True IFF the alarm will ring exactly at time

next_ring(*current_time=None*)
if current_time is None, it is now()

class larigira.timegen_every.**SingleAlarm**(*obj*)
Bases: *larigira.timegen_every.Alarm*

rings a single time

description = 'Only once, at a specified date and time'

has_ring(*current_time=None*)
returns True IFF the alarm will ring exactly at time

next_ring(*current_time=None*)
if current_time is None, it is now()

larigira.timegen_every.**getdate**(*val*)

8.1.32 larigira.unused module

This component will look for files to be removed. There are some assumptions:

- Only files in $TMPDIR are removed. Please remember that larigira has its own specific TMPDIR
- MPD URIs are parsed, and only file:/// is supported

class larigira.unused.**UnusedCleaner**(*conf*)
Bases: object

check_playlist()
check playlist + internal watchlist to see what can be removed

watch(*uri*)
adds fpath to the list of "watched" file

as soon as it leaves the mpc playlist, it is removed

larigira.unused.**old_commonpath**(*directories*)

8.1.33 Module contents

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

G

H

I

L

M

N

O

P

R

S

T

U

V

W