LyriX Documentation

Release 0.1

Davide Lo Re, Leonardo Barcaroli

June 22, 2012

CONTENTS

Contents:

CHAPTER

ONE

PLANS FOR THE LIBRARY

Note: Not all of this has been implemented yet. See *Current status* for more informations. Pretty much everything that hasn't the *todo* status is done

This is a sort of project of what the library should look like when finished.

- A decent name :)
- Low, if not none, dependencies
- There is a simple but powerful api
  - `get(artist='foo', album='asd', title='xyz', otherinfo='this', timeout=10)` Just do it.
  - `get(... as above ...)` The first site to give an 'OK' will return
  - `get(... plugins=('this', 'that'))`

    the user is able to "filter" from the set of plugins: this way badly-behaving plugins can be avoided without uninstalling them
  - **Todo**

    `get_best(... as above ...)`

    Try to catch as much information as possible
  - **Todo**

    `get_all(... as above ...)`

    Return all results (useful for user selection) *
  - **Todo**

    `get(filename='/path/to/file.mp3, id3=True)`

    will automatically discover album, artist, title information using file metadata. This will work, however, only if eyed3 is installed
- It's not limited to lyrics, but supports artist info, cover, tabs, whatsoever
- Extensibility is provided through eggs

  - see http://base-art.net/Articles/64/ (only useful if you already know eggs)
  - plugin are separate packages
  - any package can contain entrypoint for our library
  - each plugin is called a Retriever
- Retrievers run concurrently, results analysis is performed real*time
  - This way we can take "the fastest", or wait for the best
  - **An asyncronous API should be provided**
    - This way a software can see the lyrics coming in real*time when he is calling `get_all`, not having to wait for all the lyrics to be fetched (it is especially useful because there will probably some plugin behaving badly, and we don't want it to ruin our work). And, yeah, there is timeout, but it's not a complete solution
  - **See Also:**

    Section on *Retrievers*
- ---

  Todo

  A command*line utility provides the same functionalities

  - Different outputs (human, colour, parseable)
- ---

  Todo

  A C library that wraps the python one

1.1 Retrievers

A retriever is an object that can fetch lyrics, coverart or other stuff. It is a class with a lot of metadata about his capabilities (optional), and one mandatory static method, `get_data`

No subclassing is required, only conventional class attributes are needed.

Let's see an example:

```
class FooRetriever(object):
    name = 'Foo will do'
    features = ('lyrics', 'coverart')

    @staticmethod
    def get_data(song_metadata, options)
```

The more interesting part is `get_data`: here all the fetching part is done. Both his tho arguments, `song_metadata` and `options` are dict. `song_metadata` has four main fields: `artist`, `title`, `album`, `filename`. Some of them could be None. `options` has currently only one field, but it may grow:

- `searching` A tuple containing what the user wants (similar to features). It can be useful to reduce time: suppose, for example, that your function can fetch both lyrics and coverart, but is slow on the latter. If the user is only searching lyrics, there's no need to fetch coverart

To know how to create a retriever plugin, read *Plugin HOWTO*

1.1.1 setup.py

The `setup.py` you'll find into the plugin skeleton is slightly modified to make it more "automatic": the entrypoint name is equal to `Retriever.name`, and attempts are done to autoconfigure it. If you have a complex file structure, or defines other classes than the Retriver one, it will probably fail. It should be easy, anyway, to configure it!

CHAPTER TWO

CURRENT STATUS

The pluginsystem is basic, but OK.

`get` seems to work.

A plugin skeleton (with a short HOWTO in it) has been written and seems to work; it still misses some features, though.

2.1 TODO:

Todo

`get_best(... as above ...)`

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/lyricseek/checkouts/latest/doc/
line 24.)

Todo

`get_all(... as above ...)`

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/lyricseek/checkouts/latest/doc/
line 27.)

Todo

`get(filename='/path/to/file.mp3, id3=True)`

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/lyricseek/checkouts/latest/doc/
line 30.)

Todo

A command*line utility provides the same functionalities

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/lyricseek/checkouts/latest/doc/
line 57.)

Todo

A C library that wraps the python one

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/lyricseek/checkouts/latest/doc/
line 61.)

Todo

support plugin versions

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/lyricseek/envs/latest/local/lib/p
packages/lyricseek/_pluginsystem.py:docstring of lyricseek._pluginsystem.load_plugins, line 3.)

2.2 Needing documentation

CHAPTER

THREE

API REFERENCE

3.1 User API

User API is just one function, nothing more to worry about!

The typical usage is:

```
import lyricseek
lyricseek.get(artist='The Beatles', title='Let it be', request=('lyrics',))
```

`lyricseek.get` (*artist=None*, *album=None*, *title=None*, *otherinfo=None*, *request=()*, *timeout=None*, *filename=None*, *analyzer='first_match'*, *plugin_filter=None*)

Get data about a song

Parameters

- **otherinfo** (*dict*) – Other metadata, not worthing a function parameter
- **request** (*tuple*) – all needed metadata. If empty, all will be searched
- **timeout** – timeout in seconds, None for no timeout

Return type dict

3.2 Developer API

This modules are meant to be used for internal developers, or for the user who know what he is doing.

3.2.1 _run

Main module: provide simple, ready-to-use functions to get lyrics

`lyricseek._run.get_data` (*artist=None*, *album=None*, *title=None*, *otherinfo=None*, *request=()*, *timeout=None*, *filename=None*, *analyzer='first_match'*, *plugin_filter=None*)

Get data about a song

Parameters

- **otherinfo** (*dict*) – Other metadata, not worthing a function parameter
- **request** (*tuple*) – all needed metadata. If empty, all will be searched
- **timeout** – timeout in seconds, None for no timeout

Return type dict

`lyricseek._run.`**`get_ready_retrievers`**(*artist=None*, *album=None*, *title=None*, *otherinfo=None*, *request=()*, *timeout=-1*, *filename=None*, *filter_=None*)

Note: this is not meant to be used by the casual user. Use it if you are a developer or if you really know what you're doing

This function will return an iterator over functions that take no arguments and will try to get data (that is, retrievers with arguments filled in)

Parameters

- **otherinfo** (*dict*) – Other metadata, not worthing a function parameter
- **request** (*tuple*) – all needed metadata. If empty, all will be searched
- **timeout** – currently not supported

Return type iterator

`lyricseek._run.`**`set_parallel`**(*method*)

Allows to choose between process based and threading based concurrency

Parameters **method** – can be 'process' or 'thread'

Raises ValueError If an invalid argument is given

3.2.2 pluginsystem

This module provides access to plugins, metadata and other tools

`lyricseek._pluginsystem.`**`get_plugins`**()

Returns name:class dict of plugins (if loaded)

`lyricseek._pluginsystem.`**`load_plugins`**()

Searches for plugins, and load them into memory.

Todo

support plugin versions

`lyricseek._pluginsystem.`**`register_plugin`**(*plugin*)

Warning: This is only intended for debug or dirty h4x. If unsure, you shouldn't use this

Register a plugin. Useful to add Retrievers without packing proper eggs

Parameters

- **name** – The name of the plugin. In the standard case, this is the name of the entrypoint
- **plugin** – the Retriever class implementing your plugin (must implement the interface described in *Retrievers*

Raises ValueError If the check on the plugin fails

CHAPTER

FOUR

PLUGIN HOWTO

You will find a “plugin skeleton”: it consists of a `setup.py` plus the “real” plugin directory.

The plugin is basicly a class with a lot of metadata and one (static)method. You should just fill-in metadata and code your method as you prefer.

To use the plugin, we use the egg system: for you, this means that you have to build a “distribution” (a .egg file) for your plugin. You can do it with `python setup.py bdist_egg`. Then, anyone can install it using `easy_install`.

4.1 Development mode

If you are developing the plugin, you’ll probably find yourself continously creating an egg and instaling it. This is boring!

It’s probably easier to do `sudo python setup.py develop`: this will create a sort of link, so that you won’t need to reinstall it.

4.2 Testing it easy

The skeleton contain a basic command-line interface to test your plugin. This will eliminate the need to use the library to test it.

4.3 Dependencies

If your plugin has any dependencies, you should write this in `setup.py`.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX