libreant Documentation

Release 0.3

insomnialab

December 18, 2015

Contents

Contents:

CHAPTER 1

About libreant

Libreant is a book manager for both digital and paper documents. It can store any kind of digital data actually, not only books. It's db structure makes Libreant highly customizable, documents can be archived by their types with different metadata set, moreover you can create your own preset and choose default descriptors for that kind of volume. The search function looks throught over the db, and rank matches powered by ElasticSearch. The language of metadata (as title, or description) is a compulsory field, since the db will use it to optimize the search.

Elements into Libreant are defined as volumes, for each volume you can attach many files, usually this files are pdf or book scansions. Libreant is built and intended as a federation of nodes, every node is an archive. From a node you can search into friend-nodes, with OpenSearch protocol. Possible extensions into Web are suspended.

Libreant aims to share, find and save books. It can be used by librarian who needs an archive system or to collect digital items in a file sharing project.

Libreant is created by InsomniaLab, a hacklab in Rome. for any doubts, suggestion or similar write to: insomnialab@hacari.org

Libreant is Ubercool

CHAPTER 2

Libreant architecture

Libreant is meant to be a distributed system. Actually, you can even think of nodes as standalone-systems. A node is not aware of other nodes. It is a single point of distribution with no knowledge of other points.

The system that binds the nodes together is the aggregator; an aggregator acts only as a client with respect to the nodes. Therefore multiple aggregators can coexist. This also implies that the node administration does not involve the management of the aggregation mechanism and of the aggregators themselves. Similarly, it is possible to run an aggregator without running a libreant node. As a consequence, a node cannot choose whether to be aggregated or not.

The aggregation mechanism is based on Opensearch, and relies on two mandatory fields:

- the Opensearch description
- the Opensearch response

meaning that this entries are mandatory on a node in order to be aggregated. The result component heavily relies on the relevance extension of the response spec.

We blindly trust this relevance field, so a malicious node could bias the overall result, simply increasing the relevance fields of its entries. In this way, the management of the aggregators implies also the task of checking the fairness of the aggregated nodes.

2.1 How to set up an aggregator

1. Install Libreant. Follow the instructions on *Installation*.
2. Launch Libreant setting the `AGHERANT_DESCRIPTIONS` configuration parameters. Its value should be a list of URLs. Each URL represents the Opensearch description. For Libreant it's located in `/description.xml`, so a typical URL looks like:

```
http://your.doma.in/description.xml
```

   and a typical invocation looks like:

```
libreant --agherant-descriptions "http://your.doma.in/description.xml http://other.node/descript
```

   If you want to aggregate the same libreant instance that you are running, there's a shortcut: just use `SELF`. Here's an example:

```
libreant --agherant-descriptions "SELF http://other.node/description.xml"
```

Note: Through *agherant* command line program, it's possible to run an aggregator without launching the whole libreant software

CHAPTER 3

Librarian

This chapter is dedicated to librarians, people who manage the libreant node, decide how to structure the database, organize informations and supervise the catalogue.

3.1 Presets system

One of the things that make libreant powerful is that there are almost no assumptions and restrictions about informations you can catalog with it. You can use libreant to store digital book, organize physical book metadata, CDs, comics, organization reports, posters and so on.

Stored object informations are organized in a collection of key-values pairs:

```
title:   Heart of Darkness
author:  Joseph Conrad
year:    1899
country: United Kingdom
```

Normally, when users insert new objects in the database they can choose the number and the type of key-values pairs to save, without any restrictions. Language field is the only one information that is always required.

All this freedom could be difficult to administrate, so libreant provide the preset system as a useful tool to help librarians.

3.1.1 Preset

A preset is a set of rules and properties that denote a class of object. For example, if you want to store physical book metadata in your libreant node and for every book you want to remember the date in which you bought that book, in this case you can create a preset for class `bought-book` that has always a property with id `date`.

3.1.2 Quick steps creation

To create a new preset you need to create a new json file, populate it and configure libreant to use it.

Every preset is described by one json formatted text file. So in order to create a new preset you need to create a new text file with `.json` extension. This is the simplest preset you can do:

```
{
    "id": "bought-book",
    "properties": []
}
```

Once you have created all your presets you can use the `PRESET_PATHS` configuration variable to make libreant use them. `PRESET_PATHS` accepts a list of paths (strings), you can pass paths to file or folders containing presets.

Start libreant and go to the add page, you should have a list menu from which you can choose one of your presets. If some of your presets are not listed, you can take a look at log messages to investigate the problem.

3.1.3 Preset structure

The preset file has some general fields that describe the matadata of the preset (id, description, etc...) and a list of properties describing informations that objects belonging to this preset must/should have.

Preset example:

```
{
    "id": "bought-book",
    "allow_upload": false,
    "description": "bought physical book",
    "properties": [{ "id": "title",
                     "description": "title of the book",
                     "required": true
                   },
                   { "id": "author",
                     "description": "author of the book",
                     "required": true
                   },
                   { "id": "date",
                     "description": "date in which book was bought",
                     "required": true
                   },
                   { "id": "genre",
                     "description": "genre of the book",
                     "required": true,
                     "type": "enum",
                     "values": ["novel", "scientific", "essay", "poetry"]
                   }]
}
```

General fields:

Key	Type	Required	Default	Description
id	string	True		id of the preset
description	string	False	“”	a brief description of the preset
allow_upload	boolean	False	True	permits upload of files during submission
properties	list	True		list of properties

Property fields:

Key	Type	Required	Default	Description
id	string	True		id of the property
description	string	False	“”	a brief description of the property
required	boolean	False	False	permits to leave this property empty during submission
type	string	False	“string”	the type of this property
values	list	*Enum type*		used if type is “enum”

String type

String type properties will appear in the add page as a plain text field.

Enum type

Enum type properties will appear in the add page as a list of values. Possible values must be placed in *values* field as list of strings. *values* field are required if the type of the same property is "enum".

CHAPTER 4

Sysadmin

4.1 Installation

4.1.1 System dependencies

Debian wheezy / Debian jessie / Ubuntu

Download and install the Public Signing Key for elasticsearch repo:

```
wget -qO - http://packages.elasticsearch.org/GPG-KEY-elasticsearch | sudo apt-key add -
```

Add elasticsearch repos in /etc/apt/sources.list.d/elasticsearch.list:

```
echo "deb http://packages.elasticsearch.org/elasticsearch/1.7/debian stable main" | sudo tee /etc/apt
```

Install requirements:

```
sudo apt-get update && sudo apt-get install python2.7 gcc python2.7-dev python-virtualenv openjdk-7-j
```

Note: if you have problem installing elasticsearch try to follow the official installation guide

Arch

Install all necessary packages:

```
sudo pacman -Sy python2 python2-virtualenv elasticsearch
```

4.1.2 Python dependencies

Create a virtual env:

```
virtualenv -p /usr/bin/python2 ve
```

Install libreant and all python dependencies:

```
./ve/bin/pip install libreant
```

4.2 Execution

4.2.1 Start elsticsearch

Debian wheezy / Ubuntu

Start elasticsearch service:

```
sudo service elasticsearch start
```

Note: If you want to automatically start elasticsearch during bootup:

```
sudo update-rc.d elasticsearch defaults 95 10
```

Arch / Debian jessie

Start elasticsearch service:

```
sudo systemctl start elasticsearch
```

Note: If you want to automatically start elasticsearch during bootup:

```
sudo systemctl enable elasticsearch
```

4.2.2 Start libreant

To execute libreant:

```
./ve/bin/libreant
```

CHAPTER 5

How to write documentation

We care a lot about documentation. So this chapter is both about technical reference and guidelines.

5.1 Markup language

Documentation is written using restructuredText; it's a very rich markup language, so learning it all may be difficult. You can start reading a quick guide; you can then pass to a slightly longest guide.

As with all the code, you can learn much just reading pre-existing one. So go to next section and you'll know where it is placed.

5.2 Documentation directory

Documentation is placed in `doc/source/` in libreant repository. Yes, it's just a bunch of `.rst` files. The main one is `index.rst`, and hist main part is the `toctree` directive; the list below it specifies the order in which to include all the other pages.

Note: If you are trying to add a new page to the documentation, remember to add its filename to the `toctree` in `index.rst`

To build html documentation from it, you should first of all `pip install Sphinx` inside your virtualenv. Then you can run `python setup.py build_sphinx`. This command will create documentation inside `build/sphinx/html/`. So run `firefox build/sphinx/html/index.html` and you can read it.

See also:

Installation

5.3 Documenting code

If you are a developer, you know that well-documented code is very important: it makes newcomers more comfortable hacking your project, it helps clarifying what's the goal of the code you are writing and how other parts of the project should use it. Keep in mind that libreant must be easily hackable, and the code should be kept reusable at all levels as much as possible.

Since 99% of libreant code is Python, we'll focus on it, and especially on python docstrings.

If you are writing a new module, or anyway creating a new file, the "module docstring" (that is, the docstring just at the start of the file) should explain what this module is useful for, which kind of objects will it contain, and clarify any possible caveat.

The same principle applies to classes and, to a lesser degree, to methods. If a class docstring is complete enough, it can be the case that function docstring is redundant. Even in that case, you should at least be very careful in giving meaningful names to function parameters: they help a lot, and come for free!

CHAPTER 6

How to develop

This chapter is dedicated to developers, and will guide you through code organization, design choices, etc. This is not a tutorial to python, nor to git. It will provide pointers and explanation, but will not teach you how to program.

6.1 Ingredients

libreant is coded in python2.7. Its main components are an elasticsearch db, a Fsdb and a web interface based on Flask.

6.1.1 Details about libraries

Elasticsearch is a big beast. It has a lot of features and it can be scaring. We can suggest this elasticsearch guide. The python library for elasticsearch, elasticsearch-py, is quite simple to use, and has a nice documentation.

Fsdb is a quite simple "file database": the main idea behind it is that it is a content-addressable storage. The address is simply the sha1sum of the content.

Flask is a "web microframework for python". It's not a big and complete solution like django, so you'll probably get familiar with it quite soon.

6.2 Installation

6.2.1 Using virtualenv

We will assume that you are familiar with virtualenvs. If you are not, please get familiar!

Inside a clean virtualenv, run

```
python setup.py develop
```

You are now ready to develop. And you'll find two tools inside your `$PATH`: `webant` and `libreant-manage`. The first is a webserver that will run the web interface of libreant, while the second is a command-line tool to do basic operations with libreant: exporting/importing items, searching, etc.

6.2.2 Using Vagrant

Download, setup and run the virtual machine:

```
vagrant up
```

You will then find in /liberant the installation of liberant, you can login to the vagrant box with:

```
vagrant ssh
```

6.3 Code design

This section is devoted to get a better understanding on why the code is like it is, the principles that guides us, and things like that.

6.3.1 Design choices

few assumptions about data We try to be very generic about the items that libreant will store. We do not adhere to any standard about book catalogation, nor metadata organization, nor nothing like that. We leave the libraries free to set metadata how they prefer. There is only one mandatory field in items, which is `language`. The reason it is this way, is that it's important to know the language of the metadata in order for full-text search to work properly. There are also two somewhat-special fields: `title` and `actors`; they are not required, but are sometimes used in the code (being too much agnostic is soo difficult!)

no big framework we try to avoid huge frameworks like django or similar stuff. This is both a precise need, and a matter of taste. First of all, libreant uses many different storage resources (elasticsearch, fsdb, and this list will probably grow), so most frameworks will not fit our case. But it's also because we want to avoid that the code is "locked" in a framework and therefore difficult to fork.

6.3.2 File organization

`setup.py` is the file that defines how libreant is installed, how are packages built, etc. The most common reason you could care about it, is if you need to add some dependency to libreant.

libreantdb

`libreantdb/` is a package containing an abstraction over elasticsearch. Again: this is elasticsearch-only, and completely unaware of any other storage, or the logic of libreant itself.

webant

`webant/` is a package; you could think that it only contains web-specific logic, but this is not the case. Instead, all that is not in `libreantdb` is in `webant`, which is surely a bit counterintuitive.

The web application (defined in `webant.py`) "contains" a Blueprint called `agherant`. Agherant is the part of libreant that cares about "aggregating" multiple nodes in one single search engine. We believe that agherant is an important component, and if we really want to make libreant a distributed network, it should be very reusable. That's why agherant is a blueprint: it should be reusable easily.

`manage.py` is what will be installed as `libreant-manage`: a simple command-line manager for lot of libreant operations. `libreant-manage` is meant to be a tool for developers (reproduce scenarios easily) and sysadmins

(batch operations, debug), surely not for librarians! This program is actually based on flask-script, so you may wonder why we use flask for something that is not web related at all; the point is that we use flask as an application framework more than a web framework.

`templates/` is... well, it contains templates. They are written with jinja templating language. The *render_template* function

documentation

Documentation is kept on `doc/source/` and is comprised of `.rst` files. The syntax used is restructuredText. Don't forget to update documentation when you change something!

6.3.3 API

You can read API

6.3.4 Coding style

PEP8 must be used in all the code.

Docstrings are used for autogenerating api documentation, so please don't forget to provide clear, detailed explanation of what the module/class/function does, how to use it, when is it useful, etc. If you want to be really nice, consider using restructured-text directives to improve the structure of the documentation: they're fun to use.

We care a lot about documentation, so please don't leave documentation out-of-date. If you change the parameters that a function is accepting, please document it. If you are making changes to the end user's experience, please fix the user manual.

Never put "binary" files in the source. With 'binary', we also mean "any files that could be obtained programmatically, instead of being included". This is, for example, the case of `.mo`.

6.4 Testing

Unit tests are important both as a way of avoding regressions and as a way to document how something behaves. If your code is testable, you should test it. Yes, even if its behaviour might seem obvious. If the code you are writing is not easy to test, you should think of making it more easy to test. We use nose suite to manage tests, you can run all the tests and read coverage summary by typing:

```
python setup.py test
```

We usually follow these simple steps to add new tests:

- create a directory named `test` inside the package you want to test
- create a file in this folder `test/test_sometestgroupname.py`
- write test functions inside this file

We prefer not to have one big file, instead we usually group tests in different file with a representative name. You can see a full testing example in the preset package.

Note: if you are testing a new package remember to add the new package name in `cover-package` directive under `[nosetests]` section in /setup.cfg file.

6.5 Contributing

Like `libreant`? You can help!

We have a bugtracker, and you are welcome to pick tasks from there :) We use it also for discussions. Our most typical way of proposing patches is to open a pull request on github; if, for whatever reason, you are not comfortable with that, you can just contact us by email and send a patch, or give a link to your git repository.

CHAPTER 7

API

7.1 archivant package

class archivant.**Archivant**(*conf={}*)
Implementation of a Data Access Layer

Archivant handles both an fsdb instance and a libreantdb one and exposes an high-level API to operate on 'volumes'.

A 'volume' represents a physical/digital object stored within archivant. Volumes are structured as described in *normalize_volume()*; shortly, they have language, metadata and attachments. An attachment is an URL plus some metadata.

If you won't configure the FSDB_PATH parameter, fsdb will not be initialized and archivant will start in metadata-only mode. In metdata-only mode all file related functions will raise FileOpNotSupported.

dangling_files()
iterate over fsdb files no more attached to any volume

delete_attachments(*volumeID*, *attachmentsID*)
delete attachments from a volume

delete_volume(*volumeID*)

static denormalize_attachment(*attachment*)
convert attachment metadata from archivant to es format

static denormalize_volume(*volume*)
convert volume metadata from archivant to es format

get_all_volumes()
iterate over all stored volumes

get_attachment(*volumeID*, *attachmentID*)

get_file(*volumeID*, *attachmentID*)

get_volume(*volumeID*)

insert_attachments(*volumeID*, *attachments*)
add attachments to an already existing volume

insert_volume(*metadata*, *attachments=[]*)
Insert a new volume

Returns the ID of the added volume

metadata must be a dict containg metadata of the volume:

```
{
  "_language" : "it",  # language of the metadata
  "key1" : "value1",   # attribute
  "key2" : "value2",
  ...
  "keyN" : "valueN"
}

The only required key is `_language`
```

attachments must be an array of dict:

```
{
  "file"  : "/prova/una/path/a/caso" # path or fp
  "name"  : "nome_buffo.ext"         # name of the file (extension included) [optional if a
  "mime"  : "application/json"       # mime type of the file [optional]
  "notes" : "this file is awesome"   # notes that will be attached to this file [optional]
}
```

is_file_op_supported()

static normalize_attachment(*attachment*)
Convert attachment metadata from es to archivant format

This function makes side effect on input attachment

static normalize_volume(*volume*)
convert volume metadata from es to archivant format

This function makes side effect on input volume

output example:

```
{
    'id': 'AU0paPZOMZchuDv1iDv8',
    'type': 'volume',
    'metadata': {'_language': 'en',
                 'key1': 'value1',
                 'key2': 'value2',
                 'key3': 'value3'},
    'attachments': [{'id': 'a910e1kjdo2d192d1dko1p2kd1209d',
                     'type' : 'attachment',
                     'url': 'fsdb:///624bffa8a6f90813b7982d0e5b4c1475ebec40e3',
                     'metadata': {'download_count': 0,
                                  'mime': 'application/json',
                                  'name': 'tmp9fyat_',
                                  'notes': 'this file is awsome',
                                  'sha1': '624bffa8a6f90813b7982d0e5b4c1475ebec40e3',
                                  'size': 10}
                    }]
}
```

shrink_local_fsdb(*dangling=True*, *corrupted=True*, *dryrun=False*)
shrink local fsdb by removing dangling and/or corrupted files

return number of deleted files

update_attachment(*volumeID*, *attachmentID*, *metadata*)
update an existing attachment

the given metadata dict will be merged with the old one. only the following fields could be updated: [name, mime, notes, download_count]

update_volume(*volumeID*, *metadata*)
update existing volume metadata the given metadata will substitute the old one

7.1.1 Submodules

class archivant.archivant.**Archivant**(*conf={}*)
Implementation of a Data Access Layer

Archivant handles both an fsdb instance and a libreantdb one and exposes an high-level API to operate on 'volumes'.

A 'volume' represents a physical/digital object stored within archivant. Volumes are structured as described in *normalize_volume()*; shortly, they have language, metadata and attachments. An attachment is an URL plus some metadata.

If you won't configure the FSDB_PATH parameter, fsdb will not be initialized and archivant will start in metadata-only mode. In metdata-only mode all file related functions will raise FileOpNotSupported.

dangling_files()
iterate over fsdb files no more attached to any volume

delete_attachments(*volumeID*, *attachmentsID*)
delete attachments from a volume

delete_volume(*volumeID*)

static denormalize_attachment(*attachment*)
convert attachment metadata from archivant to es format

static denormalize_volume(*volume*)
convert volume metadata from archivant to es format

get_all_volumes()
iterate over all stored volumes

get_attachment(*volumeID*, *attachmentID*)

get_file(*volumeID*, *attachmentID*)

get_volume(*volumeID*)

insert_attachments(*volumeID*, *attachments*)
add attachments to an already existing volume

insert_volume(*metadata*, *attachments=[]*)
Insert a new volume

Returns the ID of the added volume

metadata must be a dict containg metadata of the volume:

```
{
  "_language" : "it",  # language of the metadata
  "key1" : "value1",   # attribute
  "key2" : "value2",
  ...
  "keyN" : "valueN"
}
```

```
The only required key is `_language`
```

attachments must be an array of dict:

```
{
  "file"  : "/prova/una/path/a/caso" # path or fp
  "name"  : "nome_buffo.ext"         # name of the file (extension included) [optional if a
  "mime"  : "application/json"       # mime type of the file [optional]
  "notes" : "this file is awesome"   # notes that will be attached to this file [optional]
}
```

is_file_op_supported()

static normalize_attachment(*attachment*)
Convert attachment metadata from es to archivant format

This function makes side effect on input attachment

static normalize_volume(*volume*)
convert volume metadata from es to archivant format

This function makes side effect on input volume

output example:

```
{
    'id': 'AU0paPZOMZchuDv1iDv8',
    'type': 'volume',
    'metadata': {'_language': 'en',
                 'key1': 'value1',
                 'key2': 'value2',
                 'key3': 'value3'},
    'attachments': [{'id': 'a910e1kjdo2d192d1dko1p2kd1209d',
                     'type' : 'attachment',
                     'url': 'fsdb:///624bffa8a6f90813b7982d0e5b4c1475ebec40e3',
                     'metadata': {'download_count': 0,
                                  'mime': 'application/json',
                                  'name': 'tmp9fyat_',
                                  'notes': 'this file is awsome',
                                  'sha1': '624bffa8a6f90813b7982d0e5b4c1475ebec40e3',
                                  'size': 10}
                    }]
}
```

shrink_local_fsdb(*dangling=True*, *corrupted=True*, *dryrun=False*)
shrink local fsdb by removing dangling and/or corrupted files

return number of deleted files

update_attachment(*volumeID*, *attachmentID*, *metadata*)
update an existing attachment

the given metadata dict will be merged with the old one. only the following fields could be updated: [name, mime, notes, download_count]

update_volume(*volumeID*, *metadata*)
update existing volume metadata the given metadata will substitute the old one

exception archivant.exceptions.**FileOpNotSupported**
Bases: exceptions.Exception

exception archivant.exceptions.**NotFoundException**
Bases: exceptions.Exception

7.2 conf package

7.2.1 Submodules

conf.config_utils.**from_envvar_file**(*envvar*, *environ=None*)

conf.config_utils.**from_envvars**(*prefix=None*, *environ=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.
- **environ** – use this dictionary instead of os.environ; this is here mostly for mockability
- **as_json** – If False, values will not be parsed as JSON first.

conf.config_utils.**from_file**(*fname*)

conf.config_utils.**load_configs**(*envvar_prefix*, *defaults=None*, *path=None*)
Load configuration

The following steps will be undertake:

- if *defaults* is provided, defaults are loded.
- It will attempt to load configs from file: if *path* is provided, it will be used, otherwise the path will be taken from envvar *envvar_prefix* + "SETTINGS".
- all envvars starting with *envvar_prefix* will be loaded.

conf.defaults.**get_def_conf**()
return default configurations as simple dict

conf.defaults.**get_help**(*conf*)
return the help message of a specific configuration parameter

7.3 libreantdb package

class libreantdb.**DB**(*es*, *index_name*)
Bases: object

This class contains every query method and every operation on the index

The following elasticsearch body response example provides the typical structure of a single document.

```
{
  "_index" : "libreant",
  "_type" : "book",
  "_id" : "AU4RleAfD1zQdqx6OQ8Y",
```

```
    "_version" : 1,
    "found" : true,
    "_source": {"_language": "en",
                "_text_en": "marco belletti pdf file latex manual",
                "author": "marco belletti",
                "type": "pdf file",
                "title": "latex manual",
                "_attachments": [{"sha1": "dc8dc34b3e0fec2377e5cf9ea7e4780d87ff18c5",
                                  "name": "LaTeX_Wikibook.pdf",
                                  "url": "fsdb:///dc8dc34b3e0fec2377e5cf9ea7e4780d87ff18c5",
                                  "notes": "A n example bookLatex wikibook",
                                  "mime": "application/pdf",
                                  "download_count": 7,
                                  "id": "17fd3d898a834e2689340cc8aacdebb4",
                                  "size": 23909451}]
                }
}
```


add_book(***book*)

Call it like this: db.add_book(doc_type='book', body={'title': 'foobar', '_language': 'it'})

autocomplete(*fieldname*, *start*)

delete_book(*id*)

file_is_attached(*url*)
return true if at least one book has file with the given url as attachment

get_all_books(*size=30*)

get_book_by_id(*id*)

get_books_by_actor(*authorname*)

get_books_by_title(*title*)

get_books_multilanguage(*query*)

get_books_querystring(*query*, ***kargs*)

get_books_simplequery(*query*)

get_last_inserted(*size=30*)

increment_download_count(*id*, *attachmentID*, *doc_type='book'*)
Increment the download counter of a specific file

iterate_all()

mlt(*_id*)
High-level method to do "more like this".

Its exact implementation can vary.

modify_book(*id*, *body*, *doc_type='book'*, *version=None*)
replace the entire book body

Instead of *update_book* this function will overwrite the book content with param body

If param *version* is given, it will be checked that the changes are applied upon that document version. If the document version provided is different from the one actually found, an *elasticsearch.ConflictError* will be raised

setup_db(*wait_for_ready=True*)
Create and configure index

If *wait_for_ready* is True, this function will block until status for *self.index_name* will be *yellow*

update_book(*id*, *body*, *doc_type='book'*)
Update a book

The "body" is merged with the current one. Yes, it is NOT overwritten.

In case of concurrency conflict this function could raise *elasticsearch.ConflictError*

user_search(*query*)
This acts like a "wrapper" that always point to the recommended function for user searching.

7.3.1 Submodules

class libreantdb.api.**DB**(*es*, *index_name*)
Bases: object

This class contains every query method and every operation on the index

The following elasticsearch body response example provides the typical structure of a single document.

```
{
  "_index" : "libreant",
  "_type" : "book",
  "_id" : "AU4RleAfD1zQdqx6OQ8Y",
  "_version" : 1,
  "found" : true,
  "_source": {"_language": "en",
              "_text_en": "marco belletti pdf file latex manual",
              "author": "marco belletti",
              "type": "pdf file",
              "title": "latex manual",
              "_attachments": [{"sha1": "dc8dc34b3e0fec2377e5cf9ea7e4780d87ff18c5",
                                "name": "LaTeX_Wikibook.pdf",
                                "url": "fsdb:///dc8dc34b3e0fec2377e5cf9ea7e4780d87ff18c5",
                                "notes": "A n example bookLatex wikibook",
                                "mime": "application/pdf",
                                "download_count": 7,
                                "id": "17fd3d898a834e2689340cc8aacdebb4",
                                "size": 23909451}]
             }
}
```

add_book(***book*)

Call it like this: db.add_book(doc_type='book', body={'title': 'foobar', '_language': 'it'})

autocomplete(*fieldname*, *start*)

delete_book(*id*)

file_is_attached(*url*)
return true if at least one book has file with the given url as attachment

get_all_books(*size=30*)

get_book_by_id(*id*)

get_books_by_actor(*authorname*)

get_books_by_title(*title*)

get_books_multilanguage(*query*)

get_books_querystring(*query*, ***kargs*)

get_books_simplequery(*query*)

get_last_inserted(*size=30*)

increment_download_count(*id*, *attachmentID*, *doc_type='book'*)
Increment the download counter of a specific file

iterate_all()

mlt(*_id*)
High-level method to do "more like this".

Its exact implementation can vary.

modify_book(*id*, *body*, *doc_type='book'*, *version=None*)
replace the entire book body

Instead of *update_book* this function will overwrite the book content with param body

If param *version* is given, it will be checked that the changes are applied upon that document version. If the document version provided is different from the one actually found, an *elasticsearch.ConflictError* will be raised

setup_db(*wait_for_ready=True*)
Create and configure index

If *wait_for_ready* is True, this function will block until status for *self.index_name* will be *yellow*

update_book(*id*, *body*, *doc_type='book'*)
Update a book

The "body" is merged with the current one. Yes, it is NOT overwritten.

In case of concurrency conflict this function could raise *elasticsearch.ConflictError*

user_search(*query*)
This acts like a "wrapper" that always point to the recommended function for user searching.

libreantdb.api.**collectStrings**(*leftovers*)

libreantdb.api.**validate_book**(*body*)
This does not only accept/refuse a book. It also returns an ENHANCED version of body, with (mostly fts-related) additional fields.

This function is idempotent.

7.4 presets package

class presets.**PresetManager**(*paths*, *strict=False*)
Bases: object

PresetManager deals with presets loading, validating, storing

you can use it like this:

```
pm = PresetManager(["/path/to/presets/folder", "/another/path"])
```

MAX_DEPTH = 5

7.4.1 Submodules

class presets.presetManager.**Preset**(*body*)
Bases: *presets.presetManager.Schema*

A preset is a set of rules and properties denoting a class of object

Example: A preset could be used to describe which properties an object that describe a book must have. (title, authors, etc)

check_id()

fields = {'allow_upload': {'default': True, 'required': False, 'type': <type 'bool'>}, 'description': {'default': '', 'requir

validate(*data*)
Checks if *data* respects this preset specification

It will check that every required property is present and for every property type it will make some specific control.

exception presets.presetManager.**PresetException**(*message*)
Bases: exceptions.Exception

exception presets.presetManager.**PresetFieldTypeException**(*message*)
Bases: *presets.presetManager.PresetException*

class presets.presetManager.**PresetManager**(*paths*, *strict=False*)
Bases: object

PresetManager deals with presets loading, validating, storing

you can use it like this:

```
pm = PresetManager(["/path/to/presets/folder", "/another/path"])
```

MAX_DEPTH = 5

exception presets.presetManager.**PresetMissingFieldException**(*message*)
Bases: *presets.presetManager.PresetException*

class presets.presetManager.**Property**(*body*)
Bases: *presets.presetManager.Schema*

A propety describe the format of a peculiarity of a preset

check_id()

check_type()

check_values()

fields = {'values': {'required': 'required_values', 'type': <type 'list'>, 'check': 'check_values'}, 'required': {'default':

required_values()

types = ['string', 'enum']
fields is used as in Preset class

class presets.presetManager.**Schema**
Bases: object

Schema is the parent of all the classes that needs to verify a specific object structure.

all child class in order to use schema validation must:

- describe the desired object schema using *self.fields*

- save input object in *self.body*

self.fields must be a dict, where keys match the relative *self.body* keys and values describe how relative self.body valuse must be.

Example:

```
    self.fields = { 'description': {
                        'type': basestring,
                        'required': False,
                        'default': ""
                    },
                    'allow_upload': {
                        'type': bool,
                        'required': False,
                        'default': True
                    }
                  }
```

fields = {}

7.5 users package

The *users* package manages the models and the API about users, groups and capabilities. Note that this package does **not** specify permissions for objects. Actual permissions are handled at the UI level.

The main concepts are:

- A *User* is what you think it is; something that you can login as.
- A *Group* is a collection of users. Note that a user can belong to multiple groups. A group has capabilities.
- A *Capability* is a "granted permission". You can think of it like a piece of paper saying, ie. "you can create new attachments".
  - Its *action* is a composition of Create, Read, Update, Delete (it follows the CRUD model).
  - A *domain* is a regular expression that must "match" to the description of an object. ie. `/cars/*` means "every car", while `/cars/*/tires/` means "the tires of every car"

This also means that a user has no capability (directly). It just belongs to groups, which, in turn, have capabilities.

The rationale behind what a Capability is may seem baroque, but there are several advantages to it:

- it is decoupled from the actual domains used by the UI
- the regular expression make it possible to create groups that can operate on everything (`*`).

class users.**SqliteFKDatabase**(*database*, *pragmas=None*, **args*, ***kwargs*)
Bases: peewee.SqliteDatabase

SqliteDatabase with foreignkey support enabled

initialize_connection(*conn*)

users.**create_tables**(*database*)
Create all tables in the given database

users.**gen_crypt_context**(*salt_size=None*, *rounds=None*)

users.**init_db**(*dbURL*, *pwd_salt_size=None*, *pwd_rounds=None*)
Initialize users database

initialize database and create necessary tables to handle users oprations.

Parameters **dbURL** – database url, as described in *init_proxy()*

users.**init_proxy**(*dbURL*)
Instantiate proxy to the database

Parameters **dbURL** – the url describing connection parameters to the choosen database. The url must have format explained in the Peewee url documentation.

examples:

- sqlite: `sqlite:///my_database.db`
- postgres: `postgresql://postgres:my_password@localhost:5432/my_database`
- mysql: `mysql://user:passwd@ip:port/my_db`

users.**populate_with_defaults**()
Create user admin and grant him all permission

If the admin user already exists the function will simply return

7.5.1 Submodules

exception users.api.**ConflictException**
Bases: exceptions.Exception

exception users.api.**NotFoundException**
Bases: exceptions.Exception

users.api.**add_capability**(*domain*, *action*, *simplified=True*)

users.api.**add_capability_to_group**(*capID*, *groupID*)

users.api.**add_group**(*name*)

users.api.**add_user**(*name*, *password*)

users.api.**add_user_to_group**(*userID*, *groupID*)

users.api.**delete_capability**(*capID*)

users.api.**delete_group**(*id*)

users.api.**delete_user**(*id*)

users.api.**get_anonymous_user**()

users.api.**get_capabilities_of_group**(*groupID*)

users.api.**get_capability**(*capID*)

users.api.**get_group**(*id=None*, *name=None*)

users.api.**get_groups_of_user**(*userID*)

users.api.**get_groups_with_capability**(*capID*)

users.api.**get_user**(*id=None*, *name=None*)

users.api.**get_users_in_group**(*groupID*)

users.api.**is_anonymous**(*user*)

users.api.**remove_capability_from_group**(*capID*, *groupID*)

users.api.**remove_user_from_group**(*userID*, *groupID*)

users.api.**update_capability**(*id*, *updates*)

users.api.**update_group**(*id*, *updates*)

users.api.**update_user**(*id*, *updates*)

class users.models.**Action**
Bases: int

Actions utiliy class

You can use this class attributes to compose the actions bitmask:: bitmask = Action.CREATE | Action.DELETE

The following actions are supported:

- CREATE
- READ
- UPDATE
- DELETE

ACTIONS = ['CREATE', 'READ', 'UPDATE', 'DELETE']

CREATE = 1

DELETE = 8

READ = 2

UPDATE = 4

classmethod action_bitmask(*action*)
return the bitmask associated withe the given action name

classmethod from_list(*actions*)
convert list of actions into the corresponding bitmask

to_list()
convert an actions bitmask into a list of action strings

class users.models.**ActionField**(*null=False*, *index=False*, *unique=False*, *verbose_name=None*, *help_text=None*, *db_column=None*, *default=None*, *choices=None*, *primary_key=False*, *sequence=None*, *constraints=None*, *schema=None*)
Bases: peewee.IntegerField

db_field = 'action'

db_value(*value*)

python_value(*value*)

class users.models.**BaseModel**(**args*, ***kwargs*)
Bases: peewee.Model

DoesNotExist
alias of BaseModelDoesNotExist

id = <peewee.PrimaryKeyField object>

to_dict()

class users.models.**Capability**(**args*, ***kwargs*)
Bases: *users.models.BaseModel*

Capability model

A capability is composed by a *domain* and an *action*. It represent the possibility to perform a specific set of actions on the resources described by the domain

domain
is a regular expression that describe all the resources involved in the capability. You can use *simToReg()* and *regToSim()* utility function to easily manipulate domain regular expressions.

action
an *ActionField* *what* can be done on *domain*

DoesNotExist
alias of CapabilityDoesNotExist

action = <users.models.ActionField object>

domain = <peewee.CharField object>

groups = <playhouse.fields.ManyToManyField object>

grouptocapability_set
Back-reference to expose related objects as a *SelectQuery*.

id = <peewee.PrimaryKeyField object>

match(*dom*, *act*)
Check if the given *domain* and *act* are allowed by this capability

match_action(*act*)
Check if the given *act* is allowed from this capability

match_domain(*dom*)
Check if the given *dom* is included in this capability domain

classmethod regToSim(*reg*)
Convert regular expression to simplified domain expression

classmethod simToReg(*sim*)
Convert simplified domain expression to regular expression

to_dict()

class users.models.**Group**(**args*, ***kwargs*)
Bases: *users.models.BaseModel*

Group model

A group has a set of capabilities and a number of users belonging to it. It's an handy way of grouping users with the same capability.

DoesNotExist
alias of GroupDoesNotExist

can(*domain*, *action*)

capabilities = <playhouse.fields.ManyToManyField object>

grouptocapability_set
Back-reference to expose related objects as a *SelectQuery*.

id = <peewee.PrimaryKeyField object>

name = <peewee.CharField object>

to_dict()

users = <playhouse.fields.ManyToManyField object>

usertogroup_set
Back-reference to expose related objects as a *SelectQuery*.

class users.models.**GroupToCapability**(**args*, ***kwargs*)
Bases: *users.models.BaseModel*

DoesNotExist
alias of GroupToCapabilityDoesNotExist

capability = <peewee.ForeignKeyField object>

capability_id = None

group = <peewee.ForeignKeyField object>

group_id = None

class users.models.**User**(***kargs*)
Bases: *users.models.BaseModel*

User model

DoesNotExist
alias of UserDoesNotExist

can(*domain*, *action*)
Can perform *action* on the given *domain*.

capabilities

groups = <playhouse.fields.ManyToManyField object>

id = <peewee.PrimaryKeyField object>

name = <peewee.CharField object>

pwd_hash = <peewee.CharField object>

set_password(*password*)
set user password

Generate random salt, derivate the given password using pbkdf2 algorith and store a summarizing string in *pwd_hash*. For hash format refer to passlib documentation.

to_dict()

usertogroup_set
Back-reference to expose related objects as a *SelectQuery*.

verify_password(*password*)
Check if the given password is the same stored for this user

class users.models.**UserToGroup**(**args*, ***kwargs*)
Bases: *users.models.BaseModel*

DoesNotExist
alias of UserToGroupDoesNotExist

group = <peewee.ForeignKeyField object>

group_id = None

user = <peewee.ForeignKeyField object>

user_id = None

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

a

c

l

p

u

A

B

C

D

N

P

R

S

T

U

V