mobyle2.lib Documentation

Release 0.1

Emeline Legros

January 31, 2015

Contents

Contents:

CHAPTER 1

Overview

CHAPTER 2

mobyle.common API documentation

2.1 config

Each entry point of the application must instantiate a config object with an explicit path to a configuration file before any other mobyle module import. for instance:

```
import mobyle.common.config
cfg = mobyle.common.config.Config(file = os.path.normpath(os.path.join(os.path.dirname(__file__), '.
```

This will create a kind of singleton of the config object, so the other calls to config just return this object. If you have a warning like the following message:

```
2013-03-14 11:37:08,345 - mobyle - WARNING - No configuration file available, using defaults
```

then you need to check carefully the config object.

2.1.1 config API reference

Created on Nov. 27, 2012

@author: Olivier Sallou @contact: olivier.sallou@irisa.fr @license: GPLv3

class mobyle.common.config.**Config**(*file=None*)

Config loads mobyle configuration file.

Configuration file can be specified or loaded in $HOME/.mobyle or current directory if not specified.
File also contains configuration for the logger

Config object must be instanciated once at application startup

Example

Config('path_to_my_configfile')

To get access to a Config element, one must specify the section and key.

Example

Config.get("app:main","db_uri")

To get a logger, just use the following.

Example

Config.logger("mylogger") or Config.logger() for defaults

static _Config__parse_config_file (*file=None*)
Parse the configuration file
Parameters **file** (*String*) – Configuration file to parse

__init__ (*file=None*)
Parameters **file** (*String*) – Path to the configuration file

static config ()
Get the configuration parser
Returns the config parser
Return type ConfigParser
See also:

ConfigParser

static logger (*mylogger='mobyle'*)
Gets a logger from configuration file. If no logger is defined, create a default one.
Parameters **mylogger** – name of the Logger, defaults to mobyle
Type String
Returns logger
Return type Logger

static reload ()
Reload configuration file

2.2 connection

2.2.1 Create a new MongoKit class

1. Create your class using MongoKit syntax ;-)
2. import object connection from the `mobyle.common.connection` module.
3. to register the class in mongokit add the decorator `@connection.register`
4. if you need to use an object from mongokit you need to import the corresponding module. The registration of the class in the mongokit connection is made at the import step.
5. If object need to be added to the web dashboard, on `mobyle.web://mobyle/web/__init__.py`, add the decorator `@mf_decorator` to your class.

for instance

```
from mongokit import Document
from mf.annotation import mf_decorator

from .connection import connection
from .config import Config

@mf_decorator
@connection.register
class User(Document):

    __collection__ = 'users'
    __database__ = Config.config().get('app:main','db_name')
```

2.2.2 How to use a MongoKit class

Following an example to retrieve User object in mongodb. If the module is an entry point of the application then instantiate a config object with a the path of a configuration file:

```
from mobyle.common.config import Config
config = Config( os.path.join( os.path.dirname(__file__), 'test.conf'))
```

Import the Mongokit connection object:

```
from mobyle.common.connection import connection
```

By importing users all class in users which are decorated with @connection.register are registered in connection:

```
from mobyle.common import users
```

Use Mongokit to retrieve User:

```
objects = connection.User.find({})
```

2.3 mobyleConfig

handle the part of mobyle configuration stored in the mongodb

2.3.1 mobyleConfig API reference

Created on Nov. 28, 2012

@author: Olivier Sallou @contact: olivier.sallou@irisa.fr @license: GPLv3

class mobyle.common.mobyleConfig.**MobyleConfig**(*doc=None*, *gen_skel=True*, *collection=None*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

Config loads mobyle configuration from database. It contains configuration that can be updated by administrators

bind_form(*request*)

Binds a request dictionnary to the object

Parameters **request** (*list*) – request.params.items() in the form [(key1,value1), (key1,value2), (key2,value1), ...]

Returns list of fields in error

classmethod **get_current**()

Return current active configuration

Returns active MobyleConfig

classmethod **get_display_list_fields**(*klass*)

Get the list of fields to display for a list

Returns dict

classmethod **get_renderer**(*klass*, *name*)

Gets the renderer for an object attribute

render(*fields=None*)

Render in HTML form an object

param: fields List of fields to show type: list rparam: HTML form rtype: str

render_search (*fields=None*)
Render in HTML a search form an object

param: fields List of fields to show, limited to first level of document type: list rparam: HTML form rtype: str

static renderer (*klass*, *attr_name*, *attr*, *parent=''*)
Gets a renderer for an attribute

Parameters
- **attr_name** (*str*) – name of the attribute
- **attr** (*object*) – attribute of the object

Returns selected renderer

classmethod search_by (*klass*, *field*)
In REST requests, search element by field instead of *_id*. Field must be a unique identifier. For the moment, this method
works only on primary attributes, not sub attributes of the object.
..TODO: manage sub attributes

Parameters **field** (*str*) – Field name to use as key in search

classmethod set_display_fields (*klass*, *fields*)
Sets the fields to be displayed, and in which order

param: fields List of fields to show type: list

classmethod set_display_list_fields (*klass*, *fields*)
Sets the fields to be displayed for lists, and in which order Defaults to all fields (first level only)
param: fields List of fields to show type: list

classmethod set_renderer (*klass*, *name*, *renderer*)
Sets the renderer for an object attribute

to_json ()
" Return JSON representation of the object

Returns JSON representation of the config
Return type str

2.4 project

2.4.1 Project API reference

2.5 users

2.5.1 User API reference

2.6 data

2.6.1 data API reference

Created on Nov. 12, 2012

@author: O. Doppelt @author: E. Legros @author: H. Menager @author: B. Neron @author: O. Sallou @license: GPLv3

class mobyle.common.data.**AbstractData**(*doc=None*, *gen_skel=True*, *gen_auth_types=True*, *validate=True*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

Abstract super class for all kinds of data

expr_value()

Get the value used for evaluated expressions (precond, ctrl, format, etc.)

class mobyle.common.data.**ListData**(*doc=None*, *gen_skel=True*, *gen_auth_types=True*, *validate=True*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

A data formed by a list of data sharing the same type/format

expr_value()

Get the value used for evaluated expressions (precond, ctrl, format, etc.) i.e. the list of its elements expr_values.

class mobyle.common.data.**RefData**(*doc=None*, *gen_skel=True*, *gen_auth_types=True*, *validate=True*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

A data whose value is stored on the file system on one or more files

clean()

Clean "text" data from Windows(TM) encoding

expr_value()

Get the value used for evaluated expressions (precond, ctrl, format, etc.) i.e. the list of file names.

class mobyle.common.data.**SimpleData**(*doc=None*, *gen_skel=True*, *gen_auth_types=True*, *validate=True*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

A data which has a simple type

class mobyle.common.data.**StructData**(*doc=None*, *gen_skel=True*, *gen_auth_types=True*, *validate=True*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

A data formed by a list properties referencing different data

expr_value()

Get the value used for evaluated expressions (precond, ctrl, format, etc.) i.e. the dictionary of properties with their expr_values.

class mobyle.common.data.**ValueData**(*doc=None*, *gen_skel=True*, *gen_auth_types=True*, *validate=True*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

A data whose value is stored directly in the object

mobyle.common.data.**safe_file_name**(*file_name*, *reserved*=[ ])
return a file name which will not cause any potential basic security issue for job execution or collide with "reserved" file names

2.7 service

2.7.1 service API reference

2.8 job

2.8.1 Job API reference

2.8.2 ClJob API reference

2.8.3 WorkflowJob API reference

2.9 status

2.9.1 state diagram of Status

2.9.2 Status API reference

2.10 stats

2.10.1 stat API reference

class mobyle.common.stats.stat.**ServiceUsageStatistic**(*doc=None*, *gen_skel=True*, *collection=None*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)
Service statistics

Collects usage info on which service is used after an other service

classmethod add(*service*, *next_service*)
Add a new usage statistic
Parameters

- **service** (*str*) – name of the service
- **next_service** (*str*) – name of the following service

class mobyle.common.stats.stat.**Statistic**(*doc=None*, *gen_skel=True*, *collection=None*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

Base statistic to create time related statistics.

classmethod add(*job*, *location*)

Add a new stat for the job (hourly/daily/monthly).

Parameters

- **job** (*str*) – job name e.g. service name
- **location** (*str*) – IP address of the user

2.11 mobyleError

2.11.1 mobyleError API reference

2.12 Object Manager

The ObjectManager makes use of an *options* dict to specify the current options to manage the file(s):

```
options['project'] = id of the project, as str
options['uncompress'] = bool , is the base file an archive uncompressed
if uncompress is True:
    options['name'] = name of the base file ('test.zip' for ex) to detect the unarchiver to use
    options['group'] = bool , if uncompress is True, should all files be a single dataset or one data
   options['files'] = list of file paths to copy in the dataset(s) according to group
else:
    options['files'] = list of path to the files to copy in the dataset

options['format'] = EDAM format or "auto" . If auto, a call to BioFormat detector will be done to try
options['type'] = EDAM type
```

2.12.1 Create one dataset in the pairtree

To create a projectdata from a file available on local filesystem:

```
options = {}
options['project'] = project_id  # as str

# optional set status to go directly in STATUS instead of QUEUED
# options["status"] = READY

my_dataset = ObjectManager.add("my file name", options, is_persistent)
# now, my dataset status is "QUEUED"

# Get path to the objects
my_path = my_dataset.get_file_path()
...
```

```
write file(s) in my_path
...
# Update the schema according to the files, example:
my_schema = RefData()
my_schema['path'] = relative_path_in_directory
my_schema['size'] = size_of_the_data_element
my_schema['format'] = "EDAM:..."
my_schema['type'] = "EDAM:..."

my_dataset.schema(my_schema)
my_dataset.status(ObjectManager.READY)

my_dataset.save_with_history([ List_of_new_or_updated_relative_files_path ], "message why")
```

2.12.2 Create one dataset with data in an existing directory

If the project data must refer to a data located i a specific place, but should not be copied:

```
options = {}
options['project'] = project_id

options["path"] = path_to_my_job_result_dir # for example

// Optionally set the schema
options["schema"] = StructData(....) or RefData or ... (the schema defining the data)
my_dataset = ObjectManager.add("my file name", options, is_persistent)

# Will return path_to_my_job_result_dir
my_path = my_dataset.get_file_path()

...
# Update the schema according to the files
my_dataset.schema(my_schema)

my_dataset.status(ObjectManager.READY)

my_dataset.save_with_history([ List_of_new_or_updated_relative_files_path ], "message why")
```

2.12.3 Get/update dataset/ dataset info

To get an existing dataset (ProjectData)

```
my_dataset = ObjectManager.get(my_dataset_id) #to get an existing dataset
...
my_path = my_dataset.get_file_path()
# write new files, delete ,... in path
....
# Update the schema according to the files
my_dataset.schema(my_schema)

my_dataset.save_with_history([ List_of_new_or_updated_relative_files_path ], "message why")
```

2.12.4 Delete dataset

Delete dataset in data and all data in the pairtree If *path* is set (data not in pairtree), data are not deleted

```
ObjectManager.delete(my_dataset_id)
```

2.12.5 Request a token

To request a token for limited access in time to a file

```
my_token = manager.get_token(my_dataset_id, file_path, AccessMode.READ or AccessMode.READWRITE)
```

This will grant access to token *my_token* during a default period with READ or WRITE access.

2.12.6 ObjectManager API reference

2.13 notifications

Notifications are the messages raised by the Mobyle system, the Mobyle administrator or one of the project members. Notifications are stored in the database and can be read/managed in the notification center. According to the preferences, messages will also be sent via one or more channels (email, ...), according to their type.

Project members can only create project notifications.

Notification sending requires the setup of the mail gateway in the MobyleConfig configuration.

2.13.1 mobyleConfig API reference

class mobyle.common.notifications.**Notification**(*doc=None*, *gen_skel=True*, *collection=None*, *lang='en'*, *fallback_lang='en'*, *schema_2_restore=None*)

User notifications

bind_form(*request*)

Binds a request dictionnary to the object

Parameters **request** (*list*) – request.params.items() in the form [(key1,value1), (key1,value2), (key2,value1), ...]

Returns list of fields in error

classmethod **get_display_list_fields**(*klass*)

Get the list of fields to display for a list

Returns dict

classmethod **get_renderer**(*klass*, *name*)

Gets the renderer for an object attribute

notify()

Send notification to user according to its preferences

Returns True is notification sent, False if an error occured.

render (*fields=None*)
Render in HTML form an object

param: fields List of fields to show type: list rparam: HTML form rtype: str

render_search (*fields=None*)
Render in HTML a search form an object

param: fields List of fields to show, limited to first level of document type: list rparam: HTML form rtype: str

static renderer (*klass*, *attr_name*, *attr*, *parent=''*)
Gets a renderer for an attribute

Parameters
- **attr_name** (*str*) – name of the attribute
- **attr** (*object*) – attribute of the object

Returns selected renderer

classmethod search_by (*klass*, *field*)
In REST requests, search element by field instead of *_id*. Field must be a unique identifier. For the moment, this method
works only on primary attributes, not sub attributes of the object.
..TODO: manage sub attributes

Parameters **field** (*str*) – Field name to use as key in search

sendMail (*emails=[]*)
Send notification by email

classmethod set_display_fields (*klass*, *fields*)
Sets the fields to be displayed, and in which order

param: fields List of fields to show type: list

classmethod set_display_list_fields (*klass*, *fields*)
Sets the fields to be displayed for lists, and in which order Defaults to all fields (first level only)
param: fields List of fields to show type: list

classmethod set_renderer (*klass*, *name*, *renderer*)
Sets the renderer for an object attribute

user_wants_notif (*user*, *notif_mean*, *notif_type*)
Checks user preferences to get a notification

Parameters
- **user** (*User*) – User to check
- **notif_mean** (*basestring*) – Notif to use (mail)
- **notif_type** (*int*) – Type of notif (project, data, ...)

Returns bool

CHAPTER 3

Indices and tables

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

m

Symbols

A

B

C

E

G

L

M

N

R

S

T

U

V