MkDocs-Material Sandbox

None

None

None

Table of contents

1. Welcome to MkDocs

For full documentation visit mkdocs.org.

1.1 Commands

- `mkdocs new [dir-name]` - Create a new project.
- `mkdocs serve` - Start the live-reloading docs server.
- `mkdocs build` - Build the documentation site.
- `mkdocs -h` - Print help message and exit.

1.2 Project layout

```
mkdocs.yml    # The configuration file.
docs/
    index.md  # The documentation homepage.
    ...       # Other markdown pages, images and other files.
```

2. Module

2.1 `function(a, b)`

A docstring in the Google style.

This is the description of what the function does.

Args: a: float anything, really b: a second instance of something

Returns: The sum of `a` and `b`, i.e. `a+b`.

Raises: Nothing novel, but probably some sort of error if `a` and `b` can't be added.

3. Onyx

3.1 `OnyxClient`

3.1.1 `__init__(username=None, env_password=False, directory=None)`

Initialise the client.

PARAMETER	DESCRIPTION
`username`	User to act as. If not provided, acts as the default user in the config. **TYPE:** `str` **DEFAULT:** `None`
`env_password`	If set to `True`, gets the user's password from the `ONYX_<username>_PASSWORD` environment variable. **TYPE:** `bool` **DEFAULT:** `False`
`directory`	Path to config directory. If not provided, uses directory stored in the `ONYX_CLIENT_CONFIG` environment variable. **TYPE:** `str` **DEFAULT:** `None`

3.1.2 `admin_approve(username)`

Admin-approve another user.

3.1.3 `admin_list_users()`

List all users.

3.1.4 `admin_list_waiting()`

List users waiting for admin approval.

3.1.5 `choices(project, field)`

View choices for a field.

3.1.6 `create(project, fields, test=False)`

Post a record to the database.

3.1.7 `csv_create(project, csv_path=None, csv_file=None, fields=None, delimiter=None, multithreaded=False, test=False)`

Post a .csv or .tsv containing records to the database.

3.1.8 `csv_delete(project, csv_path=None, csv_file=None, delimiter=None, multithreaded=False)`

Use a .csv or .tsv to delete records in the database.

3.1.9 `csv_update(project, csv_path=None, csv_file=None, fields=None, delimiter=None, multithreaded=False, test=False)`

Use a .csv or .tsv to update records in the database.

3.1.10 `delete(project, cid)`

Delete a record in the database.

3.1.11 `fields(project)`

View fields for a project.

3.1.12 `filter(project, fields=None, include=None, exclude=None, scope=None)`

Filter records from the database.

3.1.13 `get(project, cid, include=None, exclude=None, scope=None)`

Get a record from the database.

3.1.14 `login()`

Log in as a particular user, get a new token and store the token in the client.

If no user is provided, the `default_user` in the config is used.

3.1.15 `logout()`

Log out the user in this client.

3.1.16 `logoutall()`

Log out the user in all clients.

3.1.17 `query(project, query=None, include=None, exclude=None, scope=None)`

Get records from the database.

3.1.18 `register(config, first_name, last_name, email, site, password)`

`classmethod`

Create a new user.

3.1.19 `site_approve(username)`

Site-approve another user.

3.1.20 `site_list_users()`

Get the current users within the site of the requesting user.

3.1.21 `site_list_waiting()`

List users waiting for site approval.

3.1.22 `update(project, cid, fields=None, test=False)`

Update a record in the database.

3.2 `OnyxConfig`

Class for managing the config directory (and files within) that are used by `OnyxClient`.

3.2.1 `__init__(directory=None)`

Initialise the config.

PARAMETER	DESCRIPTION
`directory`	Path to config directory. If not provided, uses directory stored in the `ONYX_CLIENT_CONFIG` environment variable. **TYPE:** `str` **DEFAULT:** `None`

3.2.2 `add_user(username)`

Add a new user to the config.

PARAMETER	DESCRIPTION
`username`	Name of the user being added. **TYPE:** `str`

3.2.3 `get_default_user()`

Get the default user in the config.

3.2.4 `list_users()`

Get a list of the users in the config.

3.2.5 `set_default_user(username)`

Set the default user in the config.

PARAMETER	DESCRIPTION
`username`	Name of the user being set as default. **TYPE:** `str`

3.2.6 `write_token(username, token, expiry)`

Update the tokens file for `username`.

PARAMETER	DESCRIPTION
`username`	User within config who is having tokens updated. **TYPE:** `str`
`token`	The token being written to their tokens file. **TYPE:** `str`
`expiry`	Expiry of the token. **TYPE:** `str`

4. Varys

4.1 `varys`

A high-level wrapper for the producer and consumer classes used by varys, abstracting away the tedious details.

...

ATTRIBUTE	DESCRIPTION
`profile`	profile name inside the configuration file to use when connecting to RabbitMQ **TYPE:** `str`
`configuration_path`	path to varys confiruration JSON file, provided with either the config_path argument or the VARYS_CFG environment variable **TYPE:** `str`
`__logfile`	the path to the logfile to use for logging, provided with the logfile argument **TYPE:** `str`
`__log_level`	the log level to use for logging, provided with the log_level argument, defaults to DEBUG (the most verbose logging level) **TYPE:** `str`
`__credentials`	an instance of the configurator class, used to store the RabbitMQ connection credentials **TYPE:** `class`
`__in_channels`	a dictionary of consumer classes and queues that have been connected to for receiving messages **TYPE:** `dict`
`__out_channels`	a dictionary of producer classes and queues that have been connected to for sending messages **TYPE:** `dict`

METHOD	DESCRIPTION
`send`	Either send a message to an existing exchange, or create a new exchange connection and send the message to it. queue_suffix must be provided when sending a message to a queue for the first time to instantiate a new connection.
`receive`	Either receive a message from an existing exchange, or create a new exchange connection and receive a message from it. queue_suffix must be provided when receiving a message from a queue for the first time to instantiate a new connection. block determines whether the receive method should block until a message is received or not.
`receive_batch`	Either receive a batch of messages from an existing exchange, or create a new exchange connection and receive a batch of messages from it. queue_suffix must be provided when receiving a message from a queue for the first time to instantiate a new connection.
`get_channels`	Return a dict of all the channels that have been connected to with the keys "consumer_channels" and "producer_channels"

4.1.1 `get_channels()`

Return all open channels.

4.1.2 `receive(exchange, queue_suffix=False, block=True)`

Either receive a message from an existing exchange, or create a new exchange connection and receive a message from it.

4.1.3 `receive_batch(exchange, queue_suffix=False)`

Either receive all messages available from an existing exchange, or create a new exchange connection and receive all messages available from it.

4.1.4 `send(message, exchange, queue_suffix=False)`

Either send a message to an existing exchange, or create a new exchange connection and send the message to it.

4.2 `init_logger(name, log_path, log_level)`