climesync Documentation

Release 0.0.1

OSU Open Source Lab

Aug 12, 2016

Contents:

CHAPTER 1

Climesync - TimeSync Front End on the Command Line

Contents

Climesync is a command line interface to the Pymesync frontend for the OSU Open Source Lab's TimeSync API.

Climesync currently supports the following versions of the TimeSync API:

- v0

1.1 Install Climesync

To install Climesync from git, run the following commands:

```
$ git clone https://github.com/osuosl/climesync && cd climesync
$ python setup.py install
```

1.2 Running Climesync

Once the virtualenv has been created and all of the required Python packages have been installed, you can run climesync with

```
$ climesync
```

Climesync also accepts several optional command line arguments

-c <URL>, --connect <URL> Connect to a TimeSync server on startup

-u <username>, --user <username> Attempt to authenticate on startup with the given username

-p <password>, --password <password> Attempt to authenticate on startup with the given password

Since server information and user credentials can be specified in multiple places (See *Climesync Configuration* below), these values are prioritized in the following order:

User input inside program > Command line arguments > Configuration file values

1.3 Interactive Mode

Through an interactive shell, users have the following options:

c	Connect to a TimeSync server
dc	Disconnect from a TimeSync server
s	Sign in to the TimeSync server
so	Sign out from the TimeSync server

Once connected and authenticated, the following options are available:

ct	Submit a new time
ut	Update a previously submitted time with new/revised information
st	Sum the total time worked on a specific project
dt	Delete a time
gt	Query the TimeSync server for submitted times with optional filters
gp	Query the TimeSync server for projects with optional filters
ga	Query the TimeSync server for activities with optional filters
gu	Query the TimeSync server for users with optional filters

Admin-only options:

cp	Create a new project
up	Update a project with new/revised information
dp	Delete a project
ca	Create a new activity
ua	Update an activity with new/revised information
da	Delete an activity
cu	Create a new user
uu	Update a user with new/revised information
du	Delete a user

1.4 Scripting Mode

In addition to providing an interactive shell, Climesync also allows commands to be run from the command line. This is useful when calling Climesync from shell scripts and makes automating repetitive tasks for admins a breeze!

Scripting mode accepts arguments and options in the usual bash script format with one addition. To pass a list of values to a command, you format the values as a space-separated list enclosed within square brackets. For example:

```
(venv) $ ./climesync.py get-times --user="[user1 user2 user3]"
```

This example gets all the time entries submitted either by user1, user2, or user3.

When running Climesync in scripting mode, authentication can be done by specifying the username and password as command line arguments or by using the configuration file (See below)

To get a list of scripting mode commands, run

```
(venv) $ ./climesync.py --help
```

To get help for a specific scripting mode command, run

```
(venv) $ ./climesync.py <command_name> --help
```

1.5 Climesync Configuration

On the first run of the program in interactive mode, the configuration file *.climesyncrc* is created in the user's home directory. This configuration file stores server information and user credentials. If Climesync is going to only be run in interactive mode then manually editing this file manually won't be necessary because Climesync will handle updating these values while it's being run in interactive mode,

Information on the structure of this file can be obtained here.

The following configuration values are stored under the "climesync" header in .climesyncrc:

Key	Description
timesync_url	The URL of the TimeSync server to connect to on startup
username	The username of the user to authenticate as on startup
password	The password of the user to authenticate as on startup
autoup-date_config	Turn off prompts to automatically update your config when connecting to a new server or signing in as a new user

CHAPTER 2

Developer Documentation for Climesync

Contents

2.1 Setting up the Development Environment

Climesync is developed using the **'virtualenvwrapper'_** utility to manage versions and dependencies. To install virtualenvwrapper, run

```
$ pip install virtualenvwrapper
```

To create a new virtualenv and install all of Climesync's dependencies, do

```
$ mkvirtualenv venv
...
(venv) $ pip install -r requirements.txt
```

2.2 Testing Climesync

To lint climesync for non-PEP8 compliance, run

```
(venv) $ flake8 climesync.py commands.py util.py testing
```

To run unit tests, use this command:

```
(venv) $ nosetests
```

To enable Pymesync test mode when writing unit tests, call

```
connect(test=True)
```

instead of

```
connect()
```

2.3 Docopt

docopt is a module that creates command line parsers from docstrings. In interactive mode, docopt is used once to parse command line arguments such as username and password, but in scripting mode it's called twice. The first time it's called, it uses the main docstring to parse any global options, and if it sees that a command has been provided then the arguments after the command name are given to the command, which uses its own docstring to parse arguments and options.

2.4 The @climesync_command Decorator

If you don't know what a decorator is in Python, this article is a good starting point to understanding what they are and how they are used. In essence, decorators are Python's form of metaprogramming that are somewhat analagous to C/C++ #define macros.

Every Climesync command that is accessible from both interactive mode and scripting mode uses a decorator as a wrapper to handle both use cases. If the command is called in scripting mode, it handles calling `docopt()` to parse command line arguments as well as `util.fix_args()` to fix the names of the parsed arguments. If the program is in interactive mode, the decorator simply calls the command.

The decorator takes two arguments: `select_arg` and `optional_args`. `optional_args` is the simpler of the two arguments. It simply indicates whether options that are left blank should be included as non-truthy values (such as `None`) or simply left out of the dictionary that is given to Pymesync.

`select_arg` is slightly more complicated. Certain Pymesync methods don't just take a dictionary of values and also require that another keyword argument be given to select a specific object (The most notable examples being the `update_*()` methods). Since there's no good way in docopt to distinguish these select arguments from other arguments that do get put in the values dictionary, these arguments must be specified to the decorator so it handles them correctly.

Because some commands can't be called in scripting mode (Such as `connect()` and `sign_in()`, they don't have the decorator. In the command_lookup table, this is shown by putting `None` for the scripting mode name

2.5 Function Documentation

For the most part, Climesync functions match 1 to 1 with menu options. However, there are several utility functions (Such as print_json and get_fields) that help eliminate cluttered and unnecessary repeated code.

Detailed information on how to use these functions is included in the docstrings inside the Climesync source code.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search