FPlaneServer Documentation

Release 1.0.10.post0

Jan Snigula

May 31, 2019

Contents

Version: 1.0.10.post0

`fplaneserver` is a RESTFul API server to retrieve and update the focal plane configuration files for the HETDEX Survey <http://hetdex.org>‘_.

`fplaneserver` supports at least python 3.4.

This documentation is also available on fplaneserver.readthedocs.io.

CHAPTER 1

User documentation

1.1 User interface

1.1.1 FPlane retrieval

To retrieve the current fplane file call:

```
get_fplane.sh
```

this retrieves the actual fplane file and saves it as `fplaneYYYMMDD.txt`

To retrieve it and write to a different filename e.g. `fplane.txt` call:

```
get_fplane.sh -o fplane.txt
```

To retrieve the fplane file for a different date e.g. `20180715` call:

```
get_fplane.sh 20180715
```

this retrieves the actual fplane file and saves it as `fplane20180715.txt` The server also accepts other date formats including `YYYYMMDDTHH:MM:SS` so you can directly use the DATE-OBS from a fits header. In this case the default filename is `fplaneYYYYMMDDTHH:MM:SS.txt`

By default the script retrieves an fplane file like the one stored in virus_config. If you want just a short version (only active virus units, no LRS, no HRS) call:

```
get_fplane.sh -s
```

To retrieve (of available) exact IFU positions call:

```
get_fplane.sh -p
```

Access from python

This code snippet will retrieve an fplane file and writes it to filename.

datestr can be a fits datestr, something of the form YYYYMMDD (or some other formats). If you give it in the form YYYYMMDD it will retrieve the fplane file for midnight of night starting on datestr. If you leave the datestr empty, it will retrieve it for the "now" timestamp.

With actpos=True it retrieves the latest set of exact ifu positions for that date (These are usually valid since the last IFU updates).

full=True retrieves a complete fplane file with all entries and the not used commented out, with full=False, it retrieves a file with only the lines for the active VIRUS IFUs (No commented IFUS, no LRS, no HRS, no HPF).

The code as is should work with Python2 and 3.

```
try:
    # Python 3
    from urllib.request import urlopen
    from urllib.error import HTTPError
except ImportError:
    # Python 2
    from urllib2 import urlopen, HTTPError


def get_fplane(filename, datestr='', actpos=False, full=True):

    url = 'https://luna.mpe.mpg.de/fplane/' + datestr

    if actpos:
        url += '?actual_pos=1'
    else:
        url += '?actual_pos=0'

    if full:
        url += '&full_fplane=1'
    else:
        url += '&full_fplane=0'

    try:
        resp = urlopen(url)
    except HTTPError as e:
        raise Exception(' Failed to retrieve fplane file, server '
                        'responded with %d %s' % (e.getcode(), e.reason))

    with open(filename, 'w') as f:
        f.write(resp.read().decode())
```

1.1.2 FPlane update

To update the fplane data in the database a script is supplied that should be run on a daily basis as a cron job at HET, updating the database from the TCS information

CHAPTER 2

Admin documentation

2.1 Installation

2.1.1 Instructions

The recommended way

The recommended way to install FPlaneServer is using pip:

```
pip install --extra-index-url https://gate.mpe.mpg.de/pypi/simple/ fplaneserver
```

It's possible to set the extra index URL permanently by adding the following lines to the `$HOME/.pip/pip.conf` file:

```
[global]
extra-index-url = https://gate.mpe.mpg.de/pypi/simple
```

or exporting the environment variable:

```
export PIP_EXTRA_INDEX_URL=https://gate.mpe.mpg.de/pypi/simple
```

The list of released versions can be seen on the MPE pypi server. A specific version can be installed using specifiers, e.g. issuing `pip install fplaneserver==1.0`.

We suggest you install FPlaneServer into a virtualenv, in an anaconda/conda or in similar environments.

Of course it is also possible to install FPlaneServer without any of the above with:

```
pip install --user --extra-index-url https://gate.mpe.mpg.de/pypi/simple/ fplaneserver
```

This way the FPlaneServer executables are installed in `$HOME/.local/bin`, so make sure to add this to the environment variable `PATH` to be able to easily use them on the command line. The use of `sudo` when installing with pip is discouraged and potentially harmful.

From the online svn repository

These steps are to be followed if you want to install the latest version.

First get a local copy of `FPlaneServer`, you can checkout the repository with:

```
svn checkout svn://luna.mpe.mpg.de/fplaneserver/trunk fplaneserver
```

Similarly you can check out any branch. Now you can install with:

```
pip install /path/to/fplaneserver
```

or:

```
cd /path/to/fplaneserver
pip install .
```

It's also possible to install fplaneserver directly from the svn repository without checking it out:

```
pip install svn+svn://luna.mpe.mpg.de/fplaneserver/trunk#egg=fplaneserver
```

If necessary replace `trunk` with the desired tag or branch to checkout and install.

2.1.2 Dependances

Mandatory dependences

```
werkzeug
flask
flask_login
peewee
pyopenssl
gunicorn
```

Python dependencies

- testing:

```
robotframework
robotframework-requests
coverage>=4.2

tox  # for automatizing the tests
tox-pyenv
```

- documentation:

```
sphinx
numpydoc
alabaster
pyhetdex
```

- automatic documentation build:

```
sphinx-autobuild => 0.5.2
```

2.1.3 Development

If you develop `fplaneserver` we suggest you checkout the svn repository and install them in "editable" mode . We also recommend installing all of the optional dependances:

```
cd /path/to/fplaneserver
pip install -e .
```

See *Contribute to FPlaneServer* for more information.

2.1.4 Notes and problems

- It is possible to change the version to install from svn by selecting a specific commit:

  ```
  pip install svn+svn://luna.mpe.mpg.de/fplaneserver//trunk@5#egg=fplaneserver
  ```

  or a different branch/tag:

  ```
  pip install svn+svn://luna.mpe.mpg.de/fplaneserver/tag/v0.0.0#egg=fplaneserver
  ```

- If the installation gets interrupted with an error like:

  ```
  ImportError: No module named 'flask'
  ```

  run `pip install flask` and then retry `fplaneserver` installation

2.2 Setting up the server

Create the directory structure where the fplaneserver should be running e.g. fplaneserver

```
> mkdir fplaneserver
> cd fplaneserver
```

Within this directory create the subdirectory structure for fplaneserver

```
> mkdir {data,logs,spool}
```

In the work directory create a configuration file e.g. `fplaneserver.conf` overwriting the relevant configuration variables (See *FPlaneServer Configuration Variables* for details). Copy an existing `users.db` from another fplaneserver instance or add new users using the *add_dpu_user* tool. Use the make_ds_struct tool to create the hash based directory structure in the *data* directory.

2.2.1 Starting up the FPlaneServer Server

To start the fplaneserver server change into its run directory and start it with:

```
FPS_SETTINGS=$PWD/server.conf gunicorn -w 2 --threads 4 -D -b euclid04.opt.rzg.mpg.
↪de:9100 fplaneserver:app --access-logfile access.log
```

In this example for a fplaneserver running on port 9100 on euclid04.opt.rzg.mpg.de

2.2.2 Running in HTTPS mode

To run it in HTTPS mode, create a certificate / key pair (e.g. fplaneserver.crt and fplaneserver.key) and start it:

```
gunicorn -b euclid04.opt.rzg.mpg.de:9100 fplaneserver:app \
--access-logfile access.log \
--cert-file fplaneserver.crt --keyfile fplaneserver.key
```

2.3 Console scripts

2.3.1 `add_dpu_user`

`add_dpu_user -d db -u username -e email firstname lastname`

--db, -d	Name of the database file
--username, -u	Username for the new user
--email, -e	E-mail address of the user

Create a new user in the database. Prints the apikey of the newly created user.

2.3.2 `get_fps_apikey`

`get_fps_apikey -d database username`

--db, -d	Name of the database file

Retrieve the apikey for `username` from the database.

2.4 FPlaneServer Configuration Variables

2.4.1 Server settings

SERVER_URL The `hostname:port` url the server is running at.

Default: `'localhost:5000'`

USER_DATABASE Name and path of the user database

Default: `./users.db`

2.4.2 Logging settings

LOG_DIR Directory where the fplaneserver logfile is stored

Default: `./logs/`

LOG_LOWER_LEVEL Minimum loglevel to write to the logfile

Default: `logging.DEBUG`

2.4.3 FPlane storage settings

SPOOL_DIR Directory name for caching of the retrieved fplane file

2.4.4 HETWise configuration parameters

DB_USERNAME Username to use for connection to HETWise database

DB_PASSWORD Password to use for connection to HETWise database

DB_NAME HETWise database name

DB_PROJECT HETWise database project to use

Default: `HETDEX`

CHAPTER 3

Developer documentation

3.1 FPlaneServer ReSTful API

GET /fplane/(**string:** *datestr*)

GET /fplane/
Get the fplane file for a given date, or the current one of datestr is not given or is one of latest or current

Retrieve the the fplane file for a given date or the current one.

POST /fplane
Update the spectrograph configuration

The datestr is used as the starting date of the new configuration

The data section of the request must be a json dictionary. Its structure must be:

```
{config: config dictionary
 commit: True | False}
```

The `config` is the dictionary retrieved from TCS using:

```
syscmd -V 'get_hardware_status'
```

The `commit` is boolean flag, specifying if the update should be commited to the database

The fplaneserver updates the database tables and returns a new fplane file.

3.2 Contribute to FPlaneServer

3.2.1 How To

The suggested workflow for implementing bug fixes and/or new features is the following:

- Identify or, if necessary, add to our redmine issue tracker one or more issues to tackle. Multiple issues can be addressed together if they belong together. Assign the issues to yourself.
- Create a new branch from the trunk with a name either referring to the topic or the issue to solve. E.g. if you need to add a new executable, tracked by issue #1111 `do_something`:

```
svn cp ^/trunk ^/branches/do_something_1111\
-m 'create branch to solve issue #1111'
```

- Switch to the branch:

```
svn switch ^/branches/do_something_1111
```

- Implement the required changes and don't forget to track your progress on redmine. If the feature/bug fix requires a large amount of time, we suggest, when possible, to avoid one big commit at the end in favour of smaller commits. In this way, in case of breakages, is easier to traverse the branch history and find the offending code. For each commit you should add an entry in the `Changelog` file.

  If you work on multiple issues on the same branch, close one issue before proceeding to the next. When closing one issue is good habit to add in the description on the redmine the revision that resolves it.

- Every function or class added or modified should be adequately documented as described in *Coding style*.

  Documentation is essential both for users and for your fellow developers to understand the scope and signature of functions and classes. If a new module is added, it should be also added to the documentation in the appropriate place. See the existing documentation for examples.

  Each executable should be documented and its description should contain enough information and examples to allow users to easily run it.

- Every functionality should be thoroughly tested for python 3.5 or 3.6 in order to ensure that the code behaves as expected and that future modifications will not break existing functionalities. When fixing bugs, add tests to ensure that the bug will not repeat. For more information see *Testing*.

- Once the issue(s) are solved and the branch is ready, merge any pending change **from** the trunk:

```
svn merge ^/trunk
```

  While doing the merge, you might be asked to manually resolve one or more conflicts. Once all the conflicts have been solved, commit the changes with a meaningful commit message, e.g.: `merge ^/trunk into ^/branches/do_something_1111`. Then rerun the test suite to make sure your changes do not break functionalities implemented while you were working on your branch.

- Then contact the maintainer of `fplaneserver` and ask to merge your branch **back to the trunk**.

Information about branching and merging can be found in the svn book. For any questions or if you need support do not hesitate to contact the maintainer or the other developers.

3.2.2 Coding style

All the code should be compliant with the official python style guidelines described in **PEP 8**. To help you keep the code in spec, we suggest to install plugins that check the code for you, like Synstastic for vim or flycheck for Emacs.

The code should also be thoroughly documented using the numpy style. See the existing documentation for examples.

3.2.3 Testing

Note: Every part of the code should be tested and should run at least under python 3.5 and possibly 3.6

`fplaneserver` uses the testing framework provided by the robot framework package. The tests should cover every aspect of a function or method. If exceptions are explicitly raised, this should also tested to ensure that the implementation behaves as expected.

The preferred way to run the tests is using tox, an automatised test help package. If you have installed tox, with e.g. `pip install tox`, you can run it by typing:

```
tox
```

It will take care of creating virtual environments for every supported version of python, if it exists on the system, install `fplaneserver`, its dependences and the packages necessary to run the tests and runs `py.test`

You can run the tests for a specific python version using:

```
python -m robot
```

A code coverage report is also created and can be visualized opening into a browser `cover/index.html`.

Besides running the tests, the `tox` command also builds, by default, the documentation and collates the coverage tests from the various python interpreters and can copy then to some directory. To do the latter create, if necessary, the configuration file `~/.config/little_deploy.cfg` and add to it a section called `fplaneserver` with either one or both of the following options:

```
[fplaneserver]
# if given the deploys the documentation to the given dir
doc = /path/to/dir
# if given the deploys the coverage report to the given dir
cover = /path/to/other/dir

# it's also possible to insert the project name and the type of the document
# to deploy using the {project} and {type_} placeholders. E.g
# cover = /path/to/dir/{project}_{type_}
# will be expanded to /path/to/dir/fplaneserver_cover
```

For more information about the configuration file check little_deploy.

3.2.4 Documentation

To build the documentation you need the additional dependences described in *Python dependencies*. They can be installed by hand or during `fplaneserver` installation by executing one of the following commands on a local copy:

```
pip install /path/to/fplaneserver[doc]
pip install /path/to/fplaneserver[livedoc]
```

The first install `sphinx`, the `alabaster` theme and the `numpydoc` extension; the second also installs `sphinx-autobuild`.

To build the documentation in html format go to the `doc` directory and run:

```
make html
```

The output is saved in `_doc/build/html`. For the full list of available targets type `make help`.

If you are updating the documentation and want avoid the `edit-compile-browser refresh` cycle, and you have installed `sphinx-autobuild`, type:

```
make livehtml
```

then visit http://127.0.0.1:8000. The html documentation is automatically rebuilt after every change of the source and the browser reloaded.

Please make sure that every module in `fplaneserver` is present in the *Code documentation*.

3.3 Code documentation

3.3.1 Flask Web Interface

`before_after` - Flask request functions

This module contain all the code that need to execute before and after every request

`fplaneserver.flask.before_after.`**`before`**`()`
before request function. Setup the logger

`logger` - Flask logging interface

Logger implementation. IMPORTANT:

> In the format of the output two custom fields are added: * ip: the ip address of the sender * req: what kind of request is processed * path: the page required Every message logged must be submitted with the keyword ‘extra = {“ip”: request.remote_addr,
>
> > “method”: request.method, “path”: request.path}‘
>
> *extra* is created by cureweb.before_after:before and set into the *g* object

`fplaneserver.flask.logger.`**`set_logger`**`(`*app*`)`
Create and add the logging handlers to the default flask logger. Also deal with the fall-back logger

`views` - Flask url definitions

`fplaneserver.flask.views.`**`get_current_fplane`**`(`*datestr=None*`)`
Get the fplane file for a given date, or the current one of datestr is not given or is one of latest or current

Retrieve the the fplane file for a given date or the current one.

`fplaneserver.flask.views.`**`update_fplane`**`()`
Update the spectrograph configuration

The datestr is used as the starting date of the new configuration

The data section of the request must be a json dictionary. Its structure must be:

```
{config: config dictionary
 commit: True | False}
```

The `config` is the dictionary retrieved from TCS using:

```
syscmd -V 'get_hardware_status'
```

The `commit` is boolean flag, specifying if the update should be commited to the database

The fplaneserver updates the database tables and returns a new fplane file.

3.3.2 FPlaneServer system modules

`helpers` - FPlaneServer helper functions

G.. :py:currentmodule:: fplaneserver.lib.user

`user` - Flask user login handling

This module contains a user model, the database setup and the Flask-login hooks for storing usernames and passwords on the server

exception fplaneserver.lib.user.**InvalidUserException**(*message*, *status_code=None*)
Bases: *fplaneserver.lib.user.UserException*

Overloaded from `UserException` with a default status code 403

status_code = 403

exception fplaneserver.lib.user.**NoUserException**(*message*, *status_code=None*)
Bases: *fplaneserver.lib.user.UserException*

Overloaded from `UserException` with a default status code 404

status_code = 401

exception fplaneserver.lib.user.**UserException**(*message*, *status_code=None*)
Bases: `Exception`

Overloaded from Exception for handling user login errors.

Parameters

message [str] The error message of the exception

status_code [int, optional] The status_code to be sent with response. Default is 400

Attributes

message [str] The error message of the exception

status_code [int] The status_code to be sent with response.

to_dict(*self*)
Create a dictionart containing the message. This dictionary can be turned into a json and sent with the response

Returns

dict

status_code = 400

class fplaneserver.lib.user.**User**(***kwargs*)
Bases: `peewee.Model`, `flask_login.mixins.UserMixin`

User object that wraps around a sqlite 3 database containing a table with the registered users.

The user apikey should be created using `binascii.hexlify(os.urandom(24))`

Attributes

id [int] The user id

username [str] The username

firstname [str] First name of the user

lastname [str] Last name of the user

email [str] E-mail address of the user

password [str] The users password *Currently unused*

apikey [str] The key used for login.

DoesNotExist
alias of `UserDoesNotExist`

get_id(*self*)
Return the id of the user

is_active(*self*)
Return True if the user is active. (Always True)

is_anonymous(*self*)
Return True if the user is anonymous. (Always False)

is_authenticated(*self*)
Return True if the user authenticated successfully

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

apikey = <TextField: User.apikey>

email = <TextField: User.email>

firstname = <TextField: User.firstname>

id = <AutoField: User.id>

lastname = <TextField: User.lastname>

password = <TextField: User.password>

username = <TextField: User.username>

fplaneserver.lib.user.**close_db**()
Closes the database again at the end of the request.

fplaneserver.lib.user.**connect_db**()
Connects to the database specified in the USER_DATABASE configuration field.

Returns

SqliteDatabase

fplaneserver.lib.user.**get_db**()
Opens a new database connection if there is none yet for the current application context.

Returns

SqliteDatabase

fplaneserver.lib.user.**handle_user_error**(*error*)

Handler for errors, jsonifies the error as a dictionary and sets the response status code from the error status code.

Parameters

error [`UserException`] The user exception

Returns

`flask.Response`

`fplaneserver.lib.user.`**`init_db`**(*app*)

Initialize the database.

Parameters

app: Flask object application

`fplaneserver.lib.user.`**`load_user_from_request`**(*request*)

Interface hook for flask. For a given request object, try to load the user from the Authorization header. Return None, if the user cannot be found, or if the Authorization format is wrong.

The request object must contain an Authorization header of the form "FPS_API APIKEY=<user apikey>"

Parameters

request [`flask.Request`] The request object.

CHAPTER 4

About

4.1 Authors

The HETDEX collaboration:

- Jan Snigula <snigula@mpe.mpg.de>
- Francesco Montesano <montefra@mpe.mpg.de>

4.2 License

4.3 Changelog

```
2018-11-02  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Bumped version to 1.0.10-post

2018-11-02  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Bumped version to 1.0.10
        * fplaneserver/lib/helpers.py: Fix id in new controller message

2018-09-18  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Bumped version to 1.0.9-post

2018-09-18  Jan Snigula  <snigula@mpe.mpg.de>

        * fplaneserver/lib/helpers.py: Fixed timestamp_end for new
        spectrographs, added more output to e-mail.
        * tests/robot/data/check_db.py: Update for fix in timestamp_end
```

```
        * tests/robot/05__server_ops.robot: Update to new controller output

2018-09-13  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Updated dependency list for readthedocs

2018-09-13  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Bumped version to 1.0.8-post

pip 2018-09-13  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Bumped version to 1.0.8
        * doc/source/user.rst: Added code example to retrieve fplane from
        python
        * tox.ini: Disable doc generation for now, requires login
        * tests/*: Updates resulting from login changes
        * fplaneserver/lib/helpers.py: Allow some offset into future for
        10 minutes before logging it.
        * fplaneserver/flask/views.py: Moved profile creation into
        functions, making sure, that the correct configuration is used.

2018-08-01  Jan Snigula  <snigula@mpe.mpg.de>

        * fplaneserver/flask/views.py: Use get_bool to read commit flag
        * setup.py: Bumped version to 1.0.7
        * tests/*: Updates

2018-07-31  Jan Snigula  <snigula@mpe.mpg.de>

        * fplaneserver/lib/helpers.py: Fix time comparison bug,
        __data_time is in UT, refuse if older than 6 hours, warn if
        timestamp is the future.
        * setup.py: Bumped version to 1.0.6

2018-07-31  Jan Snigula  <snigula@mpe.mpg.de>

        * scripts/update_fplane.sh: Fix check for file contents
        * fplaneserver/flask/views.py: Another logging update
        * setup.py: Bumped version to 1.0.5

2018-07-27  Jan Snigula  <snigula@mpe.mpg.de>

        * fplaneserver/flask/views.py: Output files should be called .txt
        not .dat
        * setup.py: Bumped version to 1.0.4

2018-07-27  Jan Snigula  <snigula@mpe.mpg.de>

        * fplaneserver/lib/helpers.py: Temporarily disable timestamp check
        * setup.py: Bumped version to 1.0.3

2018-07-27  Jan Snigula  <snigula@mpe.mpg.de>
```

```
        * setup.py: Bumped version to 1.0.2

2018-07-27  Jan Snigula  <snigula@mpe.mpg.de>

        * scripts/* updates
        * fplaneserver/flask/views.py: Move database login up
        * doc/source/user.rst: Updated

2018-07-26  Jan Snigula  <snigula@mpe.mpg.de>

        * fplaneserver/lib/helpers.py: Publish newly created specs to
        level 2
        * setup.py: Bumped version to 1.0.1

2018-07-26  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Bumped version to 1.0.0
        * fplaneserver/*: More minor updates

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * doc/source/user.rst: Minor doc update

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Flask-login not flask-login...

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * requirements_rtd.txt: Added imports for doc building

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * setup.py: Flask not flask...

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * requirements_rtd.txt: Updated awise requirement

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * requirements_rtd.txt: Added for readthedocs

2018-07-25  Jan Snigula  <snigula@mpe.mpg.de>

        * doc/source/conf.py: Added import for readthedocs

2018-07-23  Jan Snigula  <snigula@mpe.mpg.de>

        * multiple: Bugfixing, added tests
```

4.4 TODO

CHAPTER 5

Links

- genindex
- modindex
- search

Python Module Index

f

/fplane

Index

Symbols

A

B

C

D

E

F

G

H

I

L

N

P

S

T

U