VDRP Documentation

Release 1.0.5.post0

Jan Snigula

Feb 20, 2019

Contents

1 User documentation **3**
- 1.1 Astrometry routines . . . 3
- 1.2 Throughput routines . . . 3
- 1.3 Fluxlimit routines . . . 3
- 1.4 Spectral line extraction routines . . . 3

2 Developer documentation **5**
- 2.1 Contribute to VDRP . . . 5
- 2.2 Code documentation . . . 8

3 About **11**
- 3.1 Authors . . . 11
- 3.2 Changelog . . . 11
- 3.3 TODO . . . 11

4 Links **13**

Python Module Index **15**

Version: 1.0.5.post0

VDRP the Virus Data Reduction Pipeline is a collection of scripts and FORTRAN programs for astrometry, throughput and flux limit calculation.

VDRP currently supports only python 2.7.

CHAPTER 1

User documentation

1.1 Astrometry routines

1.2 Throughput routines

1.3 Fluxlimit routines

1.3.1 Setting up and running the fluxlimit calculations

To calculate the fluxlimit cube of a given night shot call:

```
vdrp_setup_flim night shot
```

This will create a subdirectory tree of the form `nightvshot/flim` and in there a slurm batch script named `flimnightvshot.slurm` and the corresponding input files. Running the script as

```
vdrp_setup_flim --commit night shot
```

the slurm script will be sent to the batch system automatically. If needed the default runtime of `06:00:00` can be modified using –runtime on the command line.

1.4 Spectral line extraction routines

CHAPTER 2

Developer documentation

2.1 Contribute to VDRP

2.1.1 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.

2.1.2 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.

2.1.3 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
```

(continues on next page)

(continued from previous page)

```
# 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.

2.1.4 Documentation

To build the documentation you need the additional dependences described in pydep. 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*.

2.2 Code documentation

2.2.1 `astrometry` - Astrometry routines

2.2.2 `calc_fluxlim` - Fluxlimit calculation routines

2.2.3 `cltools` - Commandine tools

2.2.4 `cofes_vis` - Visualization routines

2.2.5 `containers` - Container structures

2.2.6 `daophot` - Daophot helper routines

2.2.7 `extraction` - Spectrum extraction routines

2.2.8 `file_tools` - File access routines

vdrp.file_tools.**get_dithall_file**(*basedir*, *night*, *shot*)

vdrp.file_tools.**get_mulitfits_file**(*basedir*, *night*, *shot*, *expname*, *fname*)

vdrp.file_tools.**get_norm_file**(*path*, *fname*)

vdrp.file_tools.**get_throughput_file**(*path*, *night*, *shot*)

Equivalent of rtp0 script.

Checks if a night/shot specific throughput file exists.

If true, return the filename, otherise the filename for an average throughput file.

Parameters

path [str] Path to the throughput files

shotname [str] Name of the shot

2.2.9 `fit_radec` - RA/DEC fitting routines

2.2.10 `fplane_client` - FPlane retrieval routines

2.2.11 `jobsplitter` - Jobsplitter - slurm setup tool

vdrp.jobsplitter.**create_job_file**(*fname*, *commands*, *n_nodes*, *jobs_per_file*, *jobs_per_node*, *args*)

vdrp.jobsplitter.**getDefaults**()

Get the defaults for the argument parser. Separating this out from the get_arguments routine allows us to use different defaults when using the jobsplitter from within a differen script.

vdrp.jobsplitter.**get_arguments**(*parser*)

Add command line arguments for the jobsplitter, this function can be called from another tool, adding job splitter support.

Parameters

parser [argparse.ArgumentParser]

vdrp.jobsplitter.**main**(*args*)

vdrp.jobsplitter.**n_needed**(*njobs*, *limit*)

vdrp.jobsplitter.**parse_args**(*argv*)
Command line parser

Parameters

argv [list of strings] list to parsed

Returns

namespace: Parsed arguments

vdrp.jobsplitter.**run**()

2.2.12 mphelpers - MPHelpers - Parallel processing routines

class vdrp.mphelpers.**MPPool**(*jobnum*, *num_proc*)
Pool of threads consuming tasks from a queue

add_task(*func*, **args*, ***kargs*)
Add a task to the queue

wait_completion()
Wait for completion of all the tasks in the queue

class vdrp.mphelpers.**MPWorker**(*name*, *tasks*)
Bases: multiprocessing.process.Process

Thread executing tasks from a given tasks queue

run()
Method to be run in sub-process; can be overridden in sub-class

class vdrp.mphelpers.**ThreadPool**(*num_threads*)
Pool of threads consuming tasks from a queue

add_task(*func*, **args*, ***kargs*)
Add a task to the queue

wait_completion()
Wait for completion of all the tasks in the queue

class vdrp.mphelpers.**ThreadShutDownException**

class vdrp.mphelpers.**ThreadWorker**(*name*, *tasks*)
Bases: threading.Thread

Thread executing tasks from a given tasks queue

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

vdrp.mphelpers.**mp_run**(*func*, *args*, *rargv*, *parser*)

vdrp.mphelpers.**shutdownThread**()

2.2.13 mplog - Mplog - Parallel process logging

2.2.14 photometry - Throughput measurement routines

2.2.15 programs - FORTRAN program interfaces

2.2.16 setup_fluxlim - Fluxlimit setup routines

2.2.17 star_extraction - Stellar Extraction routines

2.2.18 utils - VDRP utility routines

2.2.19 vdrp_helpers - VDRP helper routines

class vdrp.vdrp_helpers.**VdrpInfo**(**args*, ***kwargs*)
Bases: collections.OrderedDict

classmethod read(*dir*, *filename='vdrp_info.pickle'*)

save(*dir*, *filename='vdrp_info.pickle'*)

vdrp.vdrp_helpers.**read_data**(*filename*)

vdrp.vdrp_helpers.**run_command**(*cmd*, *input=None*, *wdir=None*)
Run and fortran command sending the optional input string on stdin.

Parameters

cmd [str] The command to be run, must be full path to executable

input [str, optional] Input to be sent to the command through stdin.

vdrp.vdrp_helpers.**save_data**(*d*, *filename*)

2.2.20 vdrprunner - VDRP batch runner

CHAPTER 3

About

3.1 Authors

The HETDEX collaboration:

- Jan Snigula <snigula@mpe.mpg.de>
- Maximilian Fabricius <mxhf@mpe.mpg.de>
- Daniel Farrow <dfarrow@mpe.mpg.de>

3.2 Changelog

3.3 TODO

CHAPTER 4

Links

- genindex
- modindex
- search

Python Module Index

v

vdrp.file_tools, 8
vdrp.jobsplitter, 8
vdrp.mphelpers, 9
vdrp.vdrp_helpers, 10

Index

A

add_task() (vdrp.mphelpers.MPPool method), 9
add_task() (vdrp.mphelpers.ThreadPool method), 9

C

create_job_file() (in module vdrp.jobsplitter), 8

G

get_arguments() (in module vdrp.jobsplitter), 8
get_dithall_file() (in module vdrp.file_tools), 8
get_mulitfits_file() (in module vdrp.file_tools), 8
get_norm_file() (in module vdrp.file_tools), 8
get_throughput_file() (in module vdrp.file_tools), 8
getDefaults() (in module vdrp.jobsplitter), 8

M

main() (in module vdrp.jobsplitter), 9
mp_run() (in module vdrp.mphelpers), 9
MPPool (class in vdrp.mphelpers), 9
MPWorker (class in vdrp.mphelpers), 9

N

n_needed() (in module vdrp.jobsplitter), 9

P

parse_args() (in module vdrp.jobsplitter), 9
Python Enhancement Proposals
    PEP 8, 6

R

read() (vdrp.vdrp_helpers.VdrpInfo class method), 10
read_data() (in module vdrp.vdrp_helpers), 10
run() (in module vdrp.jobsplitter), 9
run() (vdrp.mphelpers.MPWorker method), 9
run() (vdrp.mphelpers.ThreadWorker method), 9
run_command() (in module vdrp.vdrp_helpers), 10

S

save() (vdrp.vdrp_helpers.VdrpInfo method), 10
save_data() (in module vdrp.vdrp_helpers), 10
shutdownThread() (in module vdrp.mphelpers), 9

T

ThreadPool (class in vdrp.mphelpers), 9
ThreadShutDownException (class in vdrp.mphelpers), 9
ThreadWorker (class in vdrp.mphelpers), 9

V

vdrp.file_tools (module), 8
vdrp.jobsplitter (module), 8
vdrp.mphelpers (module), 9
vdrp.vdrp_helpers (module), 10
VdrpInfo (class in vdrp.vdrp_helpers), 10

W

wait_completion() (vdrp.mphelpers.MPPool method), 9
wait_completion() (vdrp.mphelpers.ThreadPool method), 9