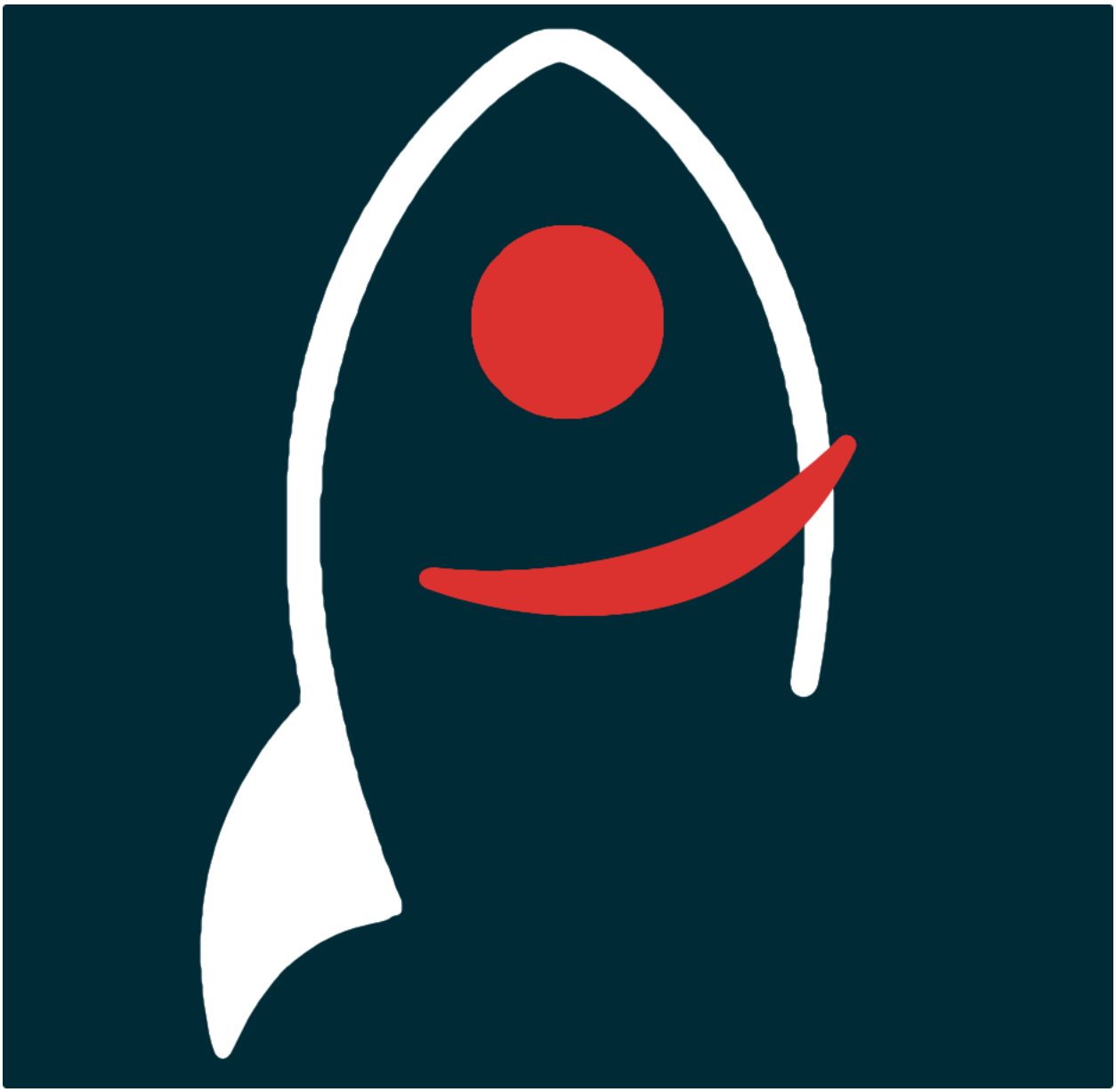

marshallEngine Documentation

Release v1.0.9

Dave Young

2021

TABLE OF CONTENTS

the engine behind the marshall webapp.

Documentation for marshallEngine is hosted by Read the Docs (development version and master version). The code lives on github. Please report any issues you find here.

CHAPTER

ONE

FEATURES

-

1.1 Installation

The easiest way to install marshallEngine is to use `pip` (here we show the install inside of a conda environment):

```
conda create -n marshallEngine python=3.7 pip
conda activate marshallEngine
pip install marshallEngine
```

Or you can clone the github repo and install from a local version of the code:

```
git clone git@github.com:thespacedoctor/marshallEngine.git
cd marshallEngine
python setup.py install
```

To upgrade to the latest version of marshallEngine use the command:

```
pip install marshallEngine --upgrade
```

To check installation was successful run `marshallEngine -v`. This should return the version number of the install.

1.1.1 Development

If you want to tinker with the code, then install in development mode. This means you can modify the code from your cloned repo:

```
git clone git@github.com:thespacedoctor/marshallEngine.git
cd marshallEngine
python setup.py develop
```

Pull requests are welcomed!

1.2 Initialisation

Before using marshallEngine you need to use the `init` command to generate a user settings file. Running the following creates a yaml settings file in your home folder under `~/.config/marshallEngine/marshallEngine.yaml`:

```
marshallEngine init
```

The file is initially populated with marshallEngine's default settings which can be adjusted to your preference.

If at any point the user settings file becomes corrupted or you just want to start afresh, simply trash the `marshallEngine.yaml` file and rerun `marshallEngine init`.

1.2.1 Modifying the Settings

Once created, open the settings file in any text editor and make any modifications needed.

1.2.2 Basic Python Setup

If you plan to use `marshallEngine` in your own scripts you will first need to parse your settings file and set up logging etc. One quick way to do this is to use the `fundamentals` package to give you a logger, a settings dictionary and a database connection (if connection details given in settings file):

```
## SOME BASIC SETUP FOR LOGGING, SETTINGS ETC
from fundamentals import tools
from os.path import expanduser
home = expanduser("~")
settingsFile  = home + "/.config/marshallEngine/marshallEngine.yaml"
su = tools(
    arguments={"settingsFile": settingsFile},
    docString=__doc__,
)
arguments, settings, log, dbConn = su.setup()
```

1.3 Todo List

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/atels/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/atlas/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- create a frankenstein template for importer

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/data.py:docstring of marshallEngine.feeders.data.data, line 6.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/panstarrs/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/tns/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/useradded/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/ztf/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add a tutorial about `update_transient_summaries` to documentation

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/housekeeping/update_transient_summaries.py:docstring of marshallEngine.housekeeping.update_transient_summaries.update_transient_summaries, line 18.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/atels/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/atlas/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- create a frankenstein template for importer

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/data.py:docstring of marshallEngine.feeders.data.data, line 6.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/panstarrs/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method

- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/tns/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/useradded/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/feeders/ztf/data.py:docstring of marshallEngine.feeders.data.data.clean_up, line 28.)

Todo:

- add a tutorial about `update_transient_summaries` to documentation

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/envs/develop/lib/python3.7/site-packages/marshallEngine-1.0.9-py3.7.egg/marshallEngine/housekeeping/update_transient_summaries.py:docstring of marshallEngine.housekeeping.update_transient_summaries.update_transient_summaries, line 18.)

Todo:

- nice!

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/marshallengine/checkouts/develop/docs/source/_tem
line 1.)

1.4 Marshall Engine Release Notes

adding tns-marker into the TNS search requests

fixed atlas and panstarrs image stamp to download from new URLs

v1.0.9 - May 7, 2021

- **REFACTOR**: resurrection code optimised to resurface a few more objects
- **REFACTOR**: some MySQL queries rewritten and table indexes added to optimise common queries
- **REFACTOR**: ATLAS forced photometry plotting code refactored to bring plots in-line with those produced by ATLAS forced photometry server. Biggest improvement is the more robust clipping of outlying data points.

v1.0.8 - March 17, 2021

- **REFACTOR**: Occasional check to make sure all akas are set (i.e. more than just for transients discovered in the last 3 weeks).
- **FIXED**: Links to the ASASSN Sky Patrol now added to all TNS reported ASASSN transient names (credentials discoverable in hover-over tool-tip)
- **FIXED**: ATel comments where getting added correctly to associated object but ticket "ATel" drop-up menu was missing some ATel links occasionally.

v1.0.7 - February 20, 2021

- **ENHANCEMENT**: Added cleanup function at end of ingests so objects appear in inbox quicker and akas are updated more frequently
- **REFACTORING**: Reduced the crossmatch radius from 7 to 4 arcsec (it is easier to merge than split transients later on)

v1.0.6 - January 29, 2021

- **FEATURE**: match transients to astronotes (not yet visualised in the webapp)
- **ENHANCEMENT**: HTM indexing added the transientBucketSummaries table so we can spatially crossmatch

v1.0.5 - January 13, 2021

- **REFACTORING**: atel parsing and matching within the marshall database now upgraded

v1.0.4 - January 11, 2021

- **REFACTORING**: some database schema changes for latest version of Sherlock to run.
- **FIXED**: added a function to recalculate sherlock original radii for merged sources. Fixes webapp visualisation.
- **FIXED**: reduced number of ATLAS LC to be generated in a single batch.

v1.0.3 - December 10, 2020

- **REFACTORING**: reading settings from marshall config folder instead of marshallEngine
- **FIXED**: the save location of lightcurve files was resulting in files not being found in webapp

v1.0.2 - November 14, 2020

- **feature** PS2 survey added as a new import (discoveries, lightcurves and stamps)

v1.0.1 - October 20, 2020

- **REFACTORING**: added time filtering on ATLAS summary CSV file (thanks Ken)
- **FIXED**: small fix to panstamp location map downloader

v1.0.0 - May 28, 2020

- Now compatible with Python 3.* *

CHAPTER

TWO

API REFERENCE

2.1 Modules

marshallEngine.commonutils	*common tools used throughout package*
marshallEngine.feeders	*Import codes for the various transients surveys that 'feed' the marshall inbox*
marshallEngine.feeders.atels	*import code for the panstarrs survey*
marshallEngine.feeders.atlas	*import code for the panstarrs survey*
marshallEngine.feeders.panstarrs	*import code for the panstarrs survey*
marshallEngine.feeders.tns	*import code for the panstarrs survey*
marshallEngine.feeders.useradded	*import code for the panstarrs survey*
marshallEngine.feeders.ztf	*import code for the panstarrs survey*
marshallEngine.housekeeping	*Housekeeping and bookkeeping actions for the marshall database*
marshallEngine.lightcurves	*Lightcurve plotting for the Marshall*
marshallEngine.services	*small services and tools for the marhall*
marshallEngine.feeders.atlas.lightcurve	*Generate the force-photometry lightcurves for ATLAS sources found in the marshall database*
marshallEngine.utKit	*Unit testing tools*

2.1.1 commonutils *(module)*

common tools used throughout package

Functions

getpackagepath()	*Get the root path for this python package*

Sub-modules

getpackagepath()	*Get the root path for this python package*

2.1.2 feeders *(module)*

Import codes for the various transients surveys that 'feed' the marshall inbox

Classes

data()	*This baseclass for the feeder survey data imports*
images()	*The base class for the feeder image cachers*

Sub-modules

atels	*import code for the panstarrs survey*
atlas	*import code for the panstarrs survey*
data()	*This baseclass for the feeder survey data imports*
images()	*The base class for the feeder image cachers*
panstarrs	*import code for the panstarrs survey*
tns	*import code for the panstarrs survey*
useradded	*import code for the panstarrs survey*
ztf	*import code for the panstarrs survey*

2.1.3 atels *(module)*

import code for the panstarrs survey

Classes

data(log, dbConn[, settings])	*Import the atels transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the atels image stamps*

Sub-modules

data(log, dbConn[, settings])	*Import the atels transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the atels image stamps*

2.1.4 atlas *(module)*

import code for the panstarrs survey

Classes

data(log, dbConn[, settings])	*Import the ATLAS transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the ATLAS image stamps*

Sub-modules

data(log, dbConn[, settings])	*Import the ATLAS transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the ATLAS image stamps*
lightcurve	*Generate the force-photometry lightcurves for ATLAS sources found in the marshall database*

2.1.5 panstarrs *(module)*

import code for the panstarrs survey

Classes

data(log, dbConn[, settings])	*Import the PanSTARRS transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the panstarrs image stamps*

Sub-modules

data(log, dbConn[, settings])	*Import the PanSTARRS transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the panstarrs image stamps*

2.1.6 tns *(module)*

import code for the panstarrs survey

Classes

data(log, dbConn[, settings])	*Import the tns transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the tns image stamps*

Sub-modules

data(log, dbConn[, settings])	*Import the tns transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the tns image stamps*

2.1.7 useradded *(module)*

import code for the panstarrs survey

Classes

data(log, dbConn[, settings])	*Import the useradded transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the useradded image stamps*

Sub-modules

data(log, dbConn[, settings])	*Import the useradded transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the useradded image stamps*

2.1.8 ztf *(module)*

import code for the panstarrs survey

Classes

data(log, dbConn[, settings])	*Import the ZTF transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the ZTF image stamps*

Sub-modules

data(log, dbConn[, settings])	*Import the ZTF transient data into the marshall database*
images(log, dbConn[, settings])	*cacher for the ZTF image stamps*

2.1.9 housekeeping *(module)*

Housekeeping and bookkeeping actions for the marshall database

Classes

update_transient_summaries(log, dbConn[, ...])	*Update the transient summaries table in the marshall database*

Sub-modules

update_transient_summaries(log, dbConn[, ...])	*Update the transient summaries table in the marshall database*

2.1.10 lightcurves *(module)*

Lightcurve plotting for the Marshall

Classes

marshall_lightcurves(log, dbConn[, ...])	*The worker class for the marshall_lightcurves module*

Sub-modules

marshall_lightcurves(log, dbConn[, ...])	*The worker class for the marshall_lightcurves module*

2.1.11 services *(module)*

small services and tools for the marhall

Classes

panstarrs_location_stamps(log, dbConn[, ...])	*The worker class for the panstarrs_location_stamps module*

Sub-modules

panstarrs_location_stamps(log, dbConn[, ...])	*The worker class for the panstarrs_location_stamps module*

2.1.12 lightcurve *(module)*

Generate the force-photometry lightcurves for ATLAS sources found in the marshall database

Author David Young

Classes

conversions(log[, settings])	*The worker class for the conversions module*
database(log[, dbSettings, autocommit])	*a database object that can setup up a ssh tunnel (optional) and a database connection*
datetime(year, month, day[, hour[, minute[, ...)	The year, month and day arguments are required.
itemgetter	itemgetter(item, ...) –> itemgetter object
str([object])	str(bytes_or_buffer[, encoding[, errors]]) -> str
tools(arguments, docString[, logLevel, ...])	*common setup methods & attributes of the main function in cl-util*
zip	zip(*iterables) –> zip object

Functions

fmultiprocess(log, function, inputArray[, ...])	multiprocess pool
generate_atlas_lightcurves(dbConn, log, settings)	generate all atlas FP lightcurves (clipped and stacked)
get_twin(ax, axis)	
get_twin_axis(ax, axis)	
old_div(a, b)	Equivalent to `a / b` on Python 2 without `from __future__ import division`.
plot_single_result(transientBucketId, log, ...)	*plot single result*

continues on next page

Table 25 – continued from previous page

readquery(sqlQuery, dbConn, log[, quiet])	Given a mysql query, read the data from the database and return the results as a list of dictionaries (database rows)
rolling_window_sigma_clip(log, array, ...)	*given a sorted list of values, median sigma-clip values based on a window of values either side of each value (rolling window) and return the array mask*
sigma_clip_data(log, fpData[, clippingSigma])	*clean up rouge data from the files by performing some basic clipping*
stack_photometry(log, magnitudes[, binningDays])	*stack the photometry for the given temporal range*
writequery(log, sqlQuery, dbConn[, Force, ...])	*Execute a MySQL write command given a sql query*

Sub-modules

conversions(log[, settings])	*The worker class for the conversions module*
database(log[, dbSettings, autocommit])	*a database object that can setup up a ssh tunnel (optional) and a database connection*
dates	AVOID INSTALLING THESE C-DEPENDENT PACKAGES
datetime(year, month, day[, hour[, minute[, ...)	The year, month and day arguments are required.
division	
fmultiprocess(log, function, inputArray[, ...])	multiprocess pool
generate_atlas_lightcurves(dbConn, log, settings)	generate all atlas FP lightcurves (clipped and stacked)
get_twin(ax, axis)	
get_twin_axis(ax, axis)	
itemgetter	itemgetter(item, ...) –> itemgetter object
math	This module provides access to the mathematical functions defined by the C standard.
mpl	AVOID INSTALLING THESE C-DEPENDENT PACKAGES
mtick	AVOID INSTALLING THESE C-DEPENDENT PACKAGES
np	AVOID INSTALLING THESE C-DEPENDENT PACKAGES
old_div(a, b)	Equivalent to `a / b` on Python 2 without `from __future__ import division`.
os	OS routines for NT or Posix depending on what system we're on.
plot_single_result(transientBucketId, log, ...)	*plot single result*
plt	AVOID INSTALLING THESE C-DEPENDENT PACKAGES
print_function	

continues on next page

Table 26 – continued from previous page

readquery(sqlQuery, dbConn, log[, quiet])	Given a mysql query, read the data from the database and return the results as a list of dictionaries (database rows)
rolling_window_sigma_clip(log, array, ...)	*given a sorted list of values, median sigma-clip values based on a window of values either side of each value (rolling window) and return the array mask*
sigma_clip_data(log, fpData[, clippingSigma])	*clean up rouge data from the files by performing some basic clipping*
stack_photometry(log, magnitudes[, binningDays])	*stack the photometry for the given temporal range*
str([object])	str(bytes_or_buffer[, encoding[, errors]]) -> str
sys	This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.
tools(arguments, docString[, logLevel, ...])	*common setup methods & attributes of the main function in cl-util*
warnings	Python part of the warnings subsystem.
writequery(log, sqlQuery, dbConn[, Force, ...])	*Execute a MySQL write command given a sql query*
zip	zip(*iterables) –> zip object

generate_atlas_lightcurves(*dbConn*, *log*, *settings*)
generate all atlas FP lightcurves (clipped and stacked)

Key Arguments

- dbConn – mysql database connection
- log – logger
- settings – settings for the marshall.

```
from marshallEngine.feeders.atlas.lightcurve import generate_atlas_lightcurves
generate_atlas_lightcurves(
    log=log,
    dbConn=dbConn,
    settings=settings
)
```

get_twin(*ax*, *axis*)

get_twin_axis(*ax*, *axis*)

plot_single_result(*transientBucketId*, *log*, *fig*, *converter*, *ax*, *settings*)
plot single result

Key Arguments:

```
- `transientBucketId` -- thr transient ID for the source in the database
- `log` -- logger
- `fig` -- the matplotlib figure to use for the plot
- `converter` -- converter to switch mjd to ut-date
- `ax` -- plot axis
- `settings` -- dictionary of settings (from yaml settings file)
```

Return:

```
- `filePath` -- path to the output PNG plot
```

sigma_clip_data(*log*, *fpData*, *clippingSigma=2.2*)
clean up rouge data from the files by performing some basic clipping

Key Arguments:

- `fpData` – data dictionary of force photometry
- `clippingSigma` – the level at which to clip flux data

Return:

- `epochs` – sigma clipped and cleaned epoch data

stack_photometry(*log*, *magnitudes*, *binningDays=1.0*)
stack the photometry for the given temporal range

Key Arguments:

```
- `magnitudes` -- dictionary of photometry divided into filter sets
- `binningDays` -- the binning to use (in days)
```

Return:

```
- `summedMagnitudes` -- the stacked photometry
```

2.1.13 utKit *(module)*

Unit testing tools

Classes

utKit(moduleDirectory[, dbConn])	*Override dryx utKit*

Sub-modules

utKit(moduleDirectory[, dbConn])	*Override dryx utKit*

class utKit(*moduleDirectory*, *dbConn=False*)
Bases: `fundamentals.utKit.utKit`

Override dryx utKit

get_project_root()
Get the root of the ``python`` package - useful for getting files in the root directory of a project

Return

- `rootPath` – the root path of a project

refresh_database()
Refresh the unit test database

setupModule()
The setupModule method

Return

- `log` – a logger
- `dbConn` – a database connection to a test database (details from yaml settings file)
- `pathToInputDir` – path to modules own test input directory
- `pathToOutputDir` – path to modules own test output directory

tearDownModule()
The tearDownModule method

2.2 Classes

marshallEngine.feeders.atels.data	*Import the atels transient data into the marshall database*
marshallEngine.feeders.atels.images	*cacher for the atels image stamps*
marshallEngine.feeders.atlas.data	*Import the ATLAS transient data into the marshall database*
marshallEngine.feeders.atlas.images	*cacher for the ATLAS image stamps*
marshallEngine.feeders.data	*This baseclass for the feeder survey data imports*
marshallEngine.feeders.images	*The base class for the feeder image cachers*
marshallEngine.feeders.panstarrs.data	*Import the PanSTARRS transient data into the marshall database*
marshallEngine.feeders.panstarrs.images	*cacher for the panstarrs image stamps*
marshallEngine.feeders.tns.data	*Import the tns transient data into the marshall database*
marshallEngine.feeders.tns.images	*cacher for the tns image stamps*
marshallEngine.feeders.useradded.data	*Import the useradded transient data into the marshall database*
marshallEngine.feeders.useradded.images	*cacher for the useradded image stamps*
marshallEngine.feeders.ztf.data	*Import the ZTF transient data into the marshall database*
marshallEngine.feeders.ztf.images	*cacher for the ZTF image stamps*
marshallEngine.housekeeping.update_transient_summaries	*Update the transient summaries table in the marshall database*
marshallEngine.lightcurves.marshall_lightcurves	*The worker class for the marshall_lightcurves module*
marshallEngine.services.panstarrs_location_stamps	*The worker class for the panstarrs_location_stamps module*

2.2.1 data *(class)*

class data(*log*, *dbConn*, *settings=False*)
Bases: marshallEngine.feeders.data.data

Import the atels transient data into the marshall database

Key Arguments:

- log – logger
- dbConn – the marshall database connection
- settings – the settings dictionary

Usage:

```
To setup your logger, settings and database connections, please use the␣
↪``fundamentals`` package (`see tutorial here <http://fundamentals.readthedocs.
↪io/en/latest/#tutorial>`_).

To initiate a data object, use the following:

.. code-block:: python

    from marshallEngine.feeders.atels.data import data
    ingester = data(
        log=log,
        settings=settings,
        dbConn=dbConn
    ).ingest(withinLastDays=withInLastDay)
```

Methods

clean_up()	*A few tasks to finish off the ingest*
download_new_atels()	*download new atel html files*
get_csv_data(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
ingest(withinLastDays)	*Ingest the data into the marshall feeder survey table*
insert_into_transientBucket([...])	*insert objects/detections from the feeder survey table into the transientbucket*
parse_atels_to_database()	*parse content of atels to the marshall database.*
update_git_repo()	*update the atel data git repo (if it exists)*

clean_up()
A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

download_new_atels()
download new atel html files

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

ingest(*withinLastDays*)
Ingest the data into the marshall feeder survey table

Key Arguments:

- `withinLastDays` – within the last number of days. *Default: 50*

insert_into_transientBucket(*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)

- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.
2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

parse_atels_to_database()
parse content of atels to the marshall database.

This populates the `atel_coordinates`, `atel_fullcontent` and `atel_names` database tables.

update_git_repo()
update the atel data git repo (if it exists) ```

2.2.2 images *(class)*

class images(*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.images.images`

cacher for the atels image stamps

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `dbConn` – the marshall database connection.

Usage:

```
To setup your logger, settings and database connections, please use the␣
↪``fundamentals`` package (`see tutorial here <http://fundamentals.readthedocs.
↪io/en/latest/#tutorial>`_).

To initiate a images object, use the following:

.. code-block:: python

    from marshallEngine.feeders.atels import images
    cacher = images(
        log=log,
        settings=settings,
```

(continues on next page)

(continued from previous page)

```
        dbConn=dbConn
    ).cache(limit=1000)
```

Methods

cache([limit])	*cache the image for the requested survey*

cache (*limit=1000*)
cache the image for the requested survey

Key Arguments

- `limit` – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.3 data *(class)*

class data (*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.data.data`

Import the ATLAS transient data into the marshall database

Key Arguments

- `log` – logger
- `dbConn` – the marshall database connection
- `settings` – the settings dictionary

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a data object, use the following:

```
from marshallEngine.feeders.atlas.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
).ingest(withinLastDays=withInLastDay)
```

Methods

`clean_up`()	*A few tasks to finish off the ingest*
`get_csv_data`(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
`ingest`(withinLastDays)	*Ingest the data into the marshall feeder survey table*
`insert_into_transientBucket`([...])	*insert objects/detections from the feeder survey table into the transientbucket*

clean_up()
A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
```

(continues on next page)

(continued from previous page)

```
    dbConn=dbConn
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

ingest (*withinLastDays*)
Ingest the data into the marshall feeder survey table

Key Arguments

- `withinLastDays` – within the last number of days. *Default: 50*

insert_into_transientBucket (*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)
- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.
2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

2.2.4 images *(class)*

class images (*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.images.images`

cacher for the ATLAS image stamps

Key Arguments

- `log` – logger
- `settings` – the settings dictionary

- `dbConn` – the marshall database connection.

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a images object, use the following:

```
from marshallEngine.feeders.atlas import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

Methods

`cache`([limit])	*cache the image for the requested survey*

cache (*limit=1000*)
cache the image for the requested survey

Key Arguments

- `limit` – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.5 data *(class)*

class data
Bases: `object`

This baseclass for the feeder survey data imports

Usage

Todo:

- create a frankenstein template for importer

To create a new survey data ingester create a new class using this class as the baseclass:

```
from ..data import data as basedata
class data(basedata):
    ....
```

Methods

`clean_up`()	*A few tasks to finish off the ingest*
`get_csv_data`(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
`insert_into_transientBucket`([...])	*insert objects/detections from the feeder survey table into the transientbucket*

clean_up()
A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
```

(continues on next page)

(continued from previous page)

```
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

`insert_into_transientBucket`(*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)
- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.
2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

2.2.6 images *(class)*

`class images`
Bases: `object`

The base class for the feeder image cachers

Usage

To create a new survey image cacher create a new class using this class as the baseclass:

```
from ..images import images as baseimages
class images(baseimages):
    ....
```

Methods

cache([limit])	*cache the image for the requested survey*

cache (*limit=1000*)
cache the image for the requested survey

Key Arguments

- `limit` – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.7 data *(class)*

class data (*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.data.data`

Import the PanSTARRS transient data into the marshall database

Key Arguments

- `log` – logger
- `dbConn` – the marshall database connection
- `settings` – the settings dictionary

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a data object, use the following:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
).ingest(withinLastDays=withInLastDay)
```

Methods

`clean_up`()	*A few tasks to finish off the ingest*
`get_csv_data`(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
`ingest`(withinLastDays)	*Ingest the data into the marshall feeder survey table*
`insert_into_transientBucket`([...])	*insert objects/detections from the feeder survey table into the transientbucket*

clean_up()
A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
```

(continues on next page)

(continued from previous page)

```
    dbConn=dbConn
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

ingest(*withinLastDays*)
Ingest the data into the marshall feeder survey table

Key Arguments

- `withinLastDays` – within the last number of days. *Default: 50*

insert_into_transientBucket(*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)
- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.
2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

2.2.8 images *(class)*

class images(*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.images.images`

cacher for the panstarrs image stamps

Key Arguments

- `log` – logger
- `settings` – the settings dictionary

- `dbConn` – the marshall database connection.

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a images object, use the following:

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

Methods

`cache`([limit])	*cache the image for the requested survey*

`cache` (*limit=1000*)
cache the image for the requested survey

Key Arguments

- `limit` – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.9 data *(class)*

`class data` (*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.data.data`

Import the tns transient data into the marshall database

Key Arguments

- `log` – logger
- `dbConn` – the marshall database connection
- `settings` – the settings dictionary

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a data object, use the following:

```
from marshallEngine.feeders.tns.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
).ingest()
```

Methods

clean_up()	*A few tasks to finish off the ingest*
get_csv_data(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
ingest([withinLastDays])	*Ingest the data into the marshall feeder survey table*
insert_into_transientBucket([...])	*insert objects/detections from the feeder survey table into the transientbucket*

clean_up()
A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

`ingest`(*withinLastDays=False*)
Ingest the data into the marshall feeder survey table

Key Arguments

- `withinLastDays` – note this will be handle by the transientNamer import to the database

`insert_into_transientBucket`(*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)
- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.
2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

2.2.10 images *(class)*

class images(*log*, *dbConn*, *settings=False*)
Bases: marshallEngine.feeders.images.images

cacher for the tns image stamps

Key Arguments

- log – logger
- settings – the settings dictionary
- dbConn – the marshall database connection.

Usage

To setup your logger, settings and database connections, please use the fundamentals package (see tutorial here).

To initiate a images object, use the following:

```
from marshallEngine.feeders.tns import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

Methods

cache([limit])	*cache the image for the requested survey*

cache(*limit=1000*)
cache the image for the requested survey

Key Arguments

- limit – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.11 data *(class)*

class data(*log*, *dbConn*, *settings=False*)
Bases: `marshallEngine.feeders.data.data`

Import the useradded transient data into the marshall database

Key Arguments

- `log` – logger
- `dbConn` – the marshall database connection
- `settings` – the settings dictionary

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a data object, use the following:

```
from marshallEngine.feeders.useradded.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
).ingest(withinLastDays=withInLastDay)
```

Methods

`clean_up`()	*A few tasks to finish off the ingest*
`get_csv_data`(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
`ingest`(withinLastDays)	*Ingest the data into the marshall feeder survey table*
`insert_into_transientBucket`([...])	*insert objects/detections from the feeder survey table into the transientbucket*

clean_up()
A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info

- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

ingest(*withinLastDays*)
Ingest the data into the marshall feeder survey table

Key Arguments

- `withinLastDays` – within the last number of days. *Default: 50*

insert_into_transientBucket(*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)
- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.

2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

2.2.12 images *(class)*

class images(*log*, *dbConn*, *settings=False*)

Bases: `marshallEngine.feeders.images.images`

cacher for the useradded image stamps

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `dbConn` – the marshall database connection.

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a images object, use the following:

```
from marshallEngine.feeders.useradded import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

Methods

`cache`([limit])	*cache the image for the requested survey*

cache(*limit=1000*)

cache the image for the requested survey

Key Arguments

- `limit` – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.13 data *(class)*

class data(*log*, *dbConn*, *settings=False*)

Bases: `marshallEngine.feeders.data.data`

Import the ZTF transient data into the marshall database

Key Arguments

- `log` – logger
- `dbConn` – the marshall database connection
- `settings` – the settings dictionary

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a data object, use the following:

```
from marshallEngine.feeders.ztf.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
).ingest(withinLastDays=withInLastDay)
```

Methods

`clean_up`()	*A few tasks to finish off the ingest*
`get_csv_data`(url[, user, pwd])	*collect the CSV data from a URL with option to supply basic auth credentials*
`ingest`(withinLastDays)	*Ingest the data into the marshall feeder survey table*
`insert_into_transientBucket`([...])	*insert objects/detections from the feeder survey table into the transientbucket*

clean_up()

A few tasks to finish off the ingest

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

get_csv_data(*url*, *user=False*, *pwd=False*)
collect the CSV data from a URL with option to supply basic auth credentials

Key Arguments

- `url` – the url to the csv file
- `user` – basic auth username
- `pwd` – basic auth password

Return

- `csvData` – a list of dictionaries from the csv file

Usage

To get the CSV data for a suvery from a given URL in the marshall settings file run something similar to:

```
from marshallEngine.feeders.panstarrs.data import data
ingester = data(
    log=log,
    settings=settings,
    dbConn=dbConn
)
csvDicts = ingester.get_csv_data(
    url=settings["panstarrs urls"]["3pi"]["summary csv"],
    user=settings["credentials"]["ps1-3pi"]["username"],
    pwd=settings["credentials"]["ps1-3pi"]["password"]
)
```

Note you will also be able to access the data via `ingester.csvDicts`

ingest(*withinLastDays*)
Ingest the data into the marshall feeder survey table

Key Arguments

- `withinLastDays` – within the last number of days. *Default: 50*

insert_into_transientBucket(*importUnmatched=True*, *updateTransientSummaries=True*)
insert objects/detections from the feeder survey table into the transientbucket

Key Arguments

- `importUnmatched` – import unmatched (new) transients into the marshall (not wanted in some circumstances)

- `updateTransientSummaries` – update the transient summaries and lightcurves? Can be True or False, or alternatively a specific transientBucketId

This method aims to reduce crossmatching and load on the database by:

1. automatically assign the transientbucket id to feeder survey detections where the object name is found in the transientbukcet (no spatial crossmatch required). Copy matched feeder survey rows to the transientbucket.
2. crossmatch remaining unique, unmatched sources in feeder survey with sources in the transientbucket. Add associated transientBucketIds to matched feeder survey sources. Copy matched feeder survey rows to the transientbucket.
3. assign a new transientbucketid to any feeder survey source not matched in steps 1 & 2. Copy these unmatched feeder survey rows to the transientbucket as new transient detections.

Return

- None

Usage

```
ingester.insert_into_transientBucket()
```

2.2.14 images *(class)*

class images(*log*, *dbConn*, *settings=False*)

Bases: `marshallEngine.feeders.images.images`

cacher for the ZTF image stamps

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `dbConn` – the marshall database connection.

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a images object, use the following:

```
from marshallEngine.feeders.ztf import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

Methods

cache([limit])	*cache the image for the requested survey*

cache (*limit=1000*)
cache the image for the requested survey

Key Arguments

- `limit` – limit the number of transients in the list so not to piss-off survey owners by downloading everything in one go.

Usage

```
from marshallEngine.feeders.panstarrs import images
cacher = images(
    log=log,
    settings=settings,
    dbConn=dbConn
).cache(limit=1000)
```

2.2.15 update_transient_summaries *(class)*

class update_transient_summaries (*log*, *dbConn*, *settings=False*, *transientBucketId=False*)
Bases: `object`

Update the transient summaries table in the marshall database

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `dbConn` – the marshall database connection
- `transientBucketId` – a single transientBucketId to update transientBucketId. Default *False* (i.e. update all)

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a update_transient_summaries object, use the following:

Todo:

- add a tutorial about `update_transient_summaries` to documentation

```
from marshallEngine.housekeeping import update_transient_summaries
updater = update_transient_summaries(
    log=log,
    settings=settings,
    dbConn=dbConn,
    transientBucketId=False
).update()
```

Methods

`update()`	*Update the transient summaries table in the marshall database*

update()
Update the transient summaries table in the marshall database

Return

- `update_transient_summaries`

Usage

```
from marshallEngine.housekeeping import update_transient_summaries
updater = update_transient_summaries(
    log=log,
    settings=settings,
    dbConn=dbConn
).update()
```

2.2.16 marshall_lightcurves *(class)*

class marshall_lightcurves(*log*, *dbConn*, *settings=False*, *transientBucketIds=[]*)
Bases: `object`

The worker class for the marshall_lightcurves module

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `dbConn` – the database connection for the mrshall
- `transientBucketIds` – the transientBucketId(s) requiring lightcurves to be regenerated. (int or list)

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a marshall_lightcurves object, use the following:

```
from marshallEngine.lightcurves import marshall_lightcurves
lc = marshall_lightcurves(
    log=log,
    dbConn=dbConn,
    settings=settings,
    transientBucketIds=[28421489, 28121353, 4637952, 27409808]
)
lc.plot()
```

Methods

`plot()`	*generate a batch of lightcurves using multiprocessing given their transientBucketIds*

plot()
generate a batch of lightcurves using multiprocessing given their transientBucketIds

Return

- `filepath` – path to the last generated plot file

Usage

```
from marshallEngine.lightcurves import marshall_lightcurves
lc = marshall_lightcurves(
    log=log,
    dbConn=dbConn,
    settings=settings,
    transientBucketIds=[28421489, 28121353, 4637952, 27409808]
)
lc.plot()
```

2.2.17 panstarrs_location_stamps *(class)*

class panstarrs_location_stamps(*log*, *dbConn*, *transientId=None*, *settings=False*)
Bases: `object`

The worker class for the panstarrs_location_stamps module

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `dbConn` – dbConn
- `transientId` – will download for one transient if single ID given. Default *None*

Usage

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a panstarrs_location_stamps object, use the following:

```
from marshallEngine.services import panstarrs_location_stamps
ps_stamp = panstarrs_location_stamps(
    log=log,
    settings=settings,
    dbConn=dbConn,
    transientId=None
).get()
```

Methods

`get`()	*get the panstarrs_location_stamps object*

get()
get the panstarrs_location_stamps object

2.3 Functions

marshallEngine.commonutils.getpackagepath	*Get the root path for this python package*
marshallEngine.feeders.atlas.lightcurve.generate_atlas_lightcurves	generate all atlas FP lightcurves (clipped and stacked)
marshallEngine.feeders.atlas.lightcurve.get_twin	
marshallEngine.feeders.atlas.lightcurve.get_twin_axis	
marshallEngine.feeders.atlas.lightcurve.plot_single_result	*plot single result*
marshallEngine.feeders.atlas.lightcurve.sigma_clip_data	*clean up rouge data from the files by performing some basic clipping*
marshallEngine.feeders.atlas.lightcurve.stack_photometry	*stack the photometry for the given temporal range*

2.3.1 getpackagepath (*function*)

getpackagepath()

Get the root path for this python package

Used in unit testing code

2.3.2 generate_atlas_lightcurves (*function*)

generate_atlas_lightcurves(*dbConn*, *log*, *settings*)
generate all atlas FP lightcurves (clipped and stacked)

Key Arguments

- `dbConn` – mysql database connection
- `log` – logger
- `settings` – settings for the marshall.

```
from marshallEngine.feeders.atlas.lightcurve import generate_atlas_lightcurves
generate_atlas_lightcurves(
    log=log,
    dbConn=dbConn,
    settings=settings
)
```

2.3.3 get_twin (*function*)

get_twin(*ax*, *axis*)

2.3.4 get_twin_axis (*function*)

get_twin_axis(*ax*, *axis*)

2.3.5 plot_single_result (*function*)

plot_single_result(*transientBucketId*, *log*, *fig*, *converter*, *ax*, *settings*)
plot single result

Key Arguments:

```
- `transientBucketId` -- thr transient ID for the source in the database
- `log` -- logger
- `fig` -- the matplotlib figure to use for the plot
- `converter` -- converter to switch mjd to ut-date
- `ax` -- plot axis
- `settings` -- dictionary of settings (from yaml settings file)
```

Return:

```
- `filePath` -- path to the output PNG plot
```

2.3.6 sigma_clip_data (*function*)

sigma_clip_data(*log*, *fpData*, *clippingSigma=2.2*)
clean up rouge data from the files by performing some basic clipping

Key Arguments:

- `fpData` – data dictionary of force photometry
- `clippingSigma` – the level at which to clip flux data

Return:

- `epochs` – sigma clipped and cleaned epoch data

2.3.7 stack_photometry (*function*)

stack_photometry(*log*, *magnitudes*, *binningDays=1.0*)
stack the photometry for the given temporal range

Key Arguments:

```
- `magnitudes` -- dictionary of photometry divided into filter sets
- `binningDays` -- the binning to use (in days)
```

Return:

```
- `summedMagnitudes` -- the stacked photometry
```

2.4 A-Z Index

Modules

marshallEngine.commonutils	common tools used throughout package
marshallEngine.feeders	Import codes for the various transients surveys that 'feed' the marshall inbox
marshallEngine.feeders.atels	import code for the panstarrs survey
marshallEngine.feeders.atlas	import code for the panstarrs survey
marshallEngine.feeders.panstarrs	import code for the panstarrs survey
marshallEngine.feeders.tns	import code for the panstarrs survey
marshallEngine.feeders.useradded	import code for the panstarrs survey
marshallEngine.feeders.ztf	import code for the panstarrs survey
marshallEngine.housekeeping	Housekeeping and bookkeeping actions for the marshall database
marshallEngine.lightcurves	Lightcurve plotting for the Marshall
marshallEngine.services	small services and tools for the marhall
marshallEngine.feeders.atlas.lightcurve	Generate the force-photometry lightcurves for ATLAS sources found in the marshall database
marshallEngine.utKit	Unit testing tools

Classes

marshallEngine.feeders.atels.data	Import the atels transient data into the marshall database
marshallEngine.feeders.atels.images	cacher for the atels image stamps
marshallEngine.feeders.atlas.data	Import the ATLAS transient data into the marshall database
marshallEngine.feeders.atlas.images	cacher for the ATLAS image stamps
marshallEngine.feeders.data	This baseclass for the feeder survey data imports
marshallEngine.feeders.images	The base class for the feeder image cachers
marshallEngine.feeders.panstarrs.data	Import the PanSTARRS transient data into the marshall database
marshallEngine.feeders.panstarrs.images	cacher for the panstarrs image stamps
marshallEngine.feeders.tns.data	Import the tns transient data into the marshall database
marshallEngine.feeders.tns.images	cacher for the tns image stamps
marshallEngine.feeders.useradded.data	Import the useradded transient data into the marshall database
marshallEngine.feeders.useradded.images	cacher for the useradded image stamps
marshallEngine.feeders.ztf.data	Import the ZTF transient data into the marshall database
marshallEngine.feeders.ztf.images	cacher for the ZTF image stamps
marshallEngine.housekeeping.update_transient_summaries	Update the transient summaries table in the marshall database
marshallEngine.lightcurves.marshall_lightcurves	The worker class for the marshall_lightcurves module
marshallEngine.services.panstarrs_location_stamps	The worker class for the panstarrs_location_stamps module

Functions

marshallEngine.commonutils.getpackagepath	*Get the root path for this python package*
marshallEngine.feeders.atlas.lightcurve.generate_atlas_lightcurves	generate all atlas FP lightcurves (clipped and stacked)
marshallEngine.feeders.atlas.lightcurve.get_twin	
marshallEngine.feeders.atlas.lightcurve.get_twin_axis	
marshallEngine.feeders.atlas.lightcurve.plot_single_result	*plot single result*
marshallEngine.feeders.atlas.lightcurve.sigma_clip_data	*clean up rouge data from the files by performing some basic clipping*
marshallEngine.feeders.atlas.lightcurve.stack_photometry	*stack the photometry for the given temporal range*

CHAPTER

THREE

MARSHALL ENGINE RELEASE NOTES

adding tns-marker into the TNS search requests

fixed atlas and panstarrs image stamp to download from new URLs

v1.0.9 - May 7, 2021

- **REFACTOR**: resurrection code optimised to resurface a few more objects
- **REFACTOR**: some MySQL queries rewritten and table indexes added to optimise common queries
- **REFACTOR**: ATLAS forced photometry plotting code refactored to bring plots in-line with those produced by ATLAS forced photometry server. Biggest improvement is the more robust clipping of outlying data points.

v1.0.8 - March 17, 2021

- **REFACTOR**: Occasional check to make sure all akas are set (i.e. more than just for transients discovered in the last 3 weeks).
- **FIXED**: Links to the ASASSN Sky Patrol now added to all TNS reported ASASSN transient names (credentials discoverable in hover-over tool-tip)
- **FIXED**: ATel comments where getting added correctly to associated object but ticket "ATel" drop-up menu was missing some ATel links occasionally.

v1.0.7 - February 20, 2021

- **ENHANCEMENT**: Added cleanup function at end of ingests so objects appear in inbox quicker and akas are updated more frequently
- **REFACTORING**: Reduced the crossmatch radius from 7 to 4 arcsec (it is easier to merge than split transients later on)

v1.0.6 - January 29, 2021

- **FEATURE**: match transients to astronotes (not yet visualised in the webapp)
- **ENHANCEMENT**: HTM indexing added the transientBucketSummaries table so we can spatially crossmatch

v1.0.5 - January 13, 2021

- **REFACTORING**: atel parsing and matching within the marshall database now upgraded

v1.0.4 - January 11, 2021

- **REFACTORING**: some database schema changes for latest version of Sherlock to run.
- **FIXED**: added a function to recalculate sherlock original radii for merged sources. Fixes webapp visualisation.
- **FIXED**: reduced number of ATLAS LC to be generated in a single batch.

v1.0.3 - December 10, 2020

- **REFACTORING**: reading settings from marshall config folder instead of marshallEngine
- **FIXED**: the save location of lightcurve files was resulting in files not being found in webapp

v1.0.2 - November 14, 2020

- **feature** PS2 survey added as a new import (discoveries, lightcurves and stamps)

v1.0.1 - October 20, 2020

- **REFACTORING**: added time filtering on ATLAS summary CSV file (thanks Ken)
- **FIXED**: small fix to panstamp location map downloader

v1.0.0 - May 28, 2020

- Now compatible with Python 3.* *

PYTHON MODULE INDEX

INDEX

P

R

S

T

U