panoptes-utils Documentation

Release 0.2.29

Wilfred Tyler Gee

Oct 21, 2020

Contents

Utility functions for use within the PANOPTES ecosystem and for general astronomical processing.

This library defines a number of modules that contain useful functions as well as a few services.

CHAPTER 1

Getting

See *Docker* for ways to run `panoptes-utils` services without installing to your host computer.

1.1 pip

To install type:

```
pip install panoptes-utils
```

1.2 Docker

Docker containers are available for running the `panoptes-utils` module and associated services, which also serve as the base container for all other PANOPTES related containers.

See the *Docker* page for details.

CHAPTER 2

Using

2.1 Modules

The modules can be used as helper utilities anywhere you would like.

2.2 Services

The services can be run either from a *Docker* image or from the installed script, as described below.

2.2.1 Config Server

A simple config param server. Runs as a Flask microservice that delivers JSON documents in response to requests for config key items.

Can be run from the installed script (defaults to `http://localhost:6563/get-config`):

```
$ panoptes-config-server -h
usage: panoptes-config-server [-h] [--host HOST] [--port PORT] [--public] [--config-
↪file CONFIG_FILE] [--no-save] [--ignore-local] [--debug]

Start the config server for PANOPTES

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host name, defaults to local interface.
  --port PORT           Local port, default 6563
  --public              If server should be public, default False. Note: inside a␣
↪docker container set this to True to expose to host.
  --config-file CONFIG_FILE
                        Config file, default $PANDIR/conf_files/pocs.yaml
  --no-save             Prevent auto saving of any new values.
```

(continues on next page)

(continued from previous page)

```
  --ignore-local        Ignore the local config files, default False. Mostly for␣
↪testing.
  --debug               Debug
```

Or inside a python process:

```
>>> from panoptes.utils.config.server import config_server
>>> from panoptes.utils.config import client

>>> server_process=config_server()

>>> client.get_config('location.horizon')
30.0

>>> server_process.terminate()  # Or just exit notebook/console
```

For more details and usage examples, see the *Config Server*.

CHAPTER 3

Development

3.1 Environment

Most users of `panoptes-utils` who need the full environment will also want the fulle POCS Environment.

3.2 Logging

The `panoptes-utils` module uses loguru for logging, which also serves as the basis for the POCS logger (see *Logger*).

To access the logs for the module, you can import directly from the `logger` module, i.e., `from panoptes.utils.logger import logger`. This is a simple wrapper around `luguru` with no extra configuration:

```
>>> from panoptes.utils import CountdownTimer
>>> # No logs by default
>>> t0 = CountdownTimer(5)
>>> t0.sleep()
False

>>> # Enable the logs
>>> from panoptes.utils.logger import logger
>>> logger.enable('panoptes')

>>> t1 = CountdownTimer(5)
2020-03-04 06:42:50 | DEBUG | panoptes.utils.time:restart:162 - Restarting Timer␣
↪(blocking) 5.00/5.00
>>> t1.sleep()
2020-03-04 06:42:53 | DEBUG | panoptes.utils.time:sleep:183 - Sleeping for 2.43␣
↪seconds
False
```

CHAPTER 4

Contents

4.1 Config Server

The config server is a simple web service that runs either on a local machine or a remote server.

The configuration is a key/value system where the keys and values must be serializable as valid yaml (or json). Configuration can be initially defined in an external yaml file and any values saved to the active server will by default be saved back to a copy of the yaml file.

The module will install the `panoptes-config-server` for command line usage, which defines a number of subcommands for interacting with (and starting) the server.

```
$ panoptes-config-server --help
↪
↪                                                                    _
Usage: panoptes-config-server [OPTIONS] COMMAND [ARGS]...

Options:
  --verbose / --no-verbose  Turn on panoptes logger for utils, default False
  --help                    Show this message and exit.

Commands:
  get  Get an item from the config server.
  run  Runs the config server with command line options.
  set  Set an item in the config server.
```

Each subcommand has its own `--help` command. See below for specific usage.

4.1.1 Starting the config server

Command line

To start the service from the command-line, use `panoptes-config-server run`:

```
$ panoptes-config-server run --help
↪
↪                                                                    _
Usage: panoptes-config-server run [OPTIONS] CONFIG_FILE

  Runs the config server with command line options.

  This function is installed as an entry_point for the module, accessible at
  `panoptes-config-server`.

Options:
  --host TEXT                     The config server IP address or host name,
                                  default 0.0.0.0
  --port TEXT                     The config server port, default 6563
  --save / --no-save              If the set values should be saved
                                  permanently, default True
  --ignore-local / --no-ignore-local
                                  Ignore the local config files, default
                                  False. Mostly for testing.
  --debug / --no-debug
  --help                          Show this message and exit.
```

Python

From python, for instance when running in a jupyter notebook, you can use:

```
>>> from panoptes.utils.config.server import config_server

>>> server_process = config_server()
...
>>> server_process.terminate()  # Or just exit notebook/console
```

4.1.2 Options

ignore_local

By default, local versions of the config files are parsed and replace any default values. For instance, the default config file is `$PANDIR/conf_files/pocs.yaml` but the config server will also look for and parse `$PANDIR/conf_files/pocs_local.yaml`.

This allows for overriding of default entries while still maintaing the originals.

This option can be disabled with the `ignore_local` setting.

Note: Automatic tests run via `pytest` will always ignore local config files unless they are being run with the `--hardware` options.

4.1.3 Using the config server

Python

The server can be queried/set in python:

```
>>> from panoptes.utils.config import client

# Show the entire config item.
>>> client.get_config('location')
{'elevation': 3400.0,
 'flat_horizon': -6.0,
 'focus_horizon': -12.0,
 'gmt_offset': -600.0,
 'horizon': 30,
 'latitude': 19.54,
 'longitude': -155.58,
 'name': 'Mauna Loa Observatory',
 'observe_horizon': -18.0,
 'timezone': 'US/Hawaii'}

# Get just a specific value.
>>> client.get_config('location.horizon')
30.0

# Set to a new value.
>>> client.set_config('location.horizon', 45)
{'location.horizon': 45.0}

# Retrieve new value.
>>> client.get_config('location.horizon')
45.0

# Work with units.
>>> from astropy import units as u
>>> client.set_config('location.horizon', 45 * u.deg)
{'location.horizon': <Quantity 45. deg>}

>>> client.get_config('location.horizon')
<Quantity 45. deg>

>>> client.get_config('location')
{'elevation': 3400.0,
 'flat_horizon': -6.0,
 'focus_horizon': -12.0,
 'gmt_offset': -600.0,
 'horizon': <Quantity 45. deg>,
 'latitude': 19.54,
 'longitude': -155.58,
 'name': 'Mauna Loa Observatory',
 'observe_horizon': -18.0,
 'timezone': 'US/Hawaii'}

# Get the second camera model
>>> client.get_config('cameras.devices[1].model')
'canon_gphoto2'
```

Command-line

The `panoptes-config-server get` command will fetch the requested key (or the entire config if no is provided) and print it out to the console as JSON string.

The `panoptes-config-server set` command will set the value for the given key.

```
$ panoptes-config-server get --key location
{
  "elevation": 3400,
  "flat_horizon": -6,
  "focus_horizon": -12,
  "gmt_offset": -600,
  "horizon": "45.0 deg",
  "latitude": 19.54,
  "longitude": -155.58,
  "name": "Mauna Loa Observatory",
  "observe_horizon": -18,
  "timezone": "US/Hawaii"
}
```

```
$ panoptes-config-server set 'location.horizon' '37 deg'
{'location.horizon': <Quantity 37. deg>}
```

See `panoptes-config-server get --help` and `panoptes-config-server set --help` for more details.

4.2 Docker

The PANOPTES utilities are available as a docker image that can be built locally for testing purposes. We also use containers based off `latest` in the Google Cloud Registry (GCR):

Image name: `panoptes-utils`

Tags: `latest` and `develop`.

4.2.1 Tags

The `panoptes-utils` image comes in two separate flavors, or tags, that serve different purposes.

latest

The `latest` image is typically used to run services or to serve as a foundational layer for other docker images. It includes all the tools required to run the various functions with the `panoptes-utils` module, including a plate-solver (astrometry.net), `sextractor`, etc.

The `latest` image is also used as a base image for the POCS images.

develop

The `develop` image is used for running the automated tests against the `develop` branch. These are run automatically on both GitHub and Travis for all code pushes but can also be run locally while doing development.

4.2.2 Building

To build the test image:

```
docker/setup-local-environment.sh
```

4.2.3 Running

To run the test suite locally:

```
scripts/testing/test-software.sh
```

4.3 License

4.4 Contributors

- Wilfred Tyler Gee <wtylergee@gmail.com>

4.5 Changelog

4.5.1 0.2.29dev

Added

- Added `oh-my-zsh` install file directly to ease some issues with GCP builds. (@wtgee #257)
- Added `source-extractor` to dependencies but with no custom config files. (@wtgee #257)
- Config Server:
  - Option to start a heartbeat or not. (@wtgee #248)

Changed

- Reverting back to `python=3.7` for compatibility w/ GCP notebooks. (@wtgee #255)
- Freezing `astropy<=4.0.1` while we wait for `astroplan` to get pushed. (@wtgee #255)
- Changed the horizon module to use numpy interpolation so we don't need to explicitly install scipy. (@wtgee #248)
- `altaz_to_radec` accepts astropy quantities. (@wtgee #250)
- Downloaded helper script doesn't have `python3` hardcoded. (@wtgee #250)
- Docker Tools (@wtgee #248):
  - Conda environment built from `resources/environment.yaml`. (@wtgee #252)
  - Adds a "developer" dockerfile and compose file to install things for developers. (@wtgee #248)
  - Docker CMD will run ipython. (@wtgee #248)
  - docker-compose file will start a jupyter-lab instance. (@wtgee #248)

Fixed

- Fixed the `oh-my-zsh` path for Docker install. (@wtgee #256)
- Return testing output from docker container, passint exit status. (@wtgee #256)

Removed

- The `stars` module, which has been moved to `panoptes-pipeline`. (@wtgee #251)
- The `metadata` module, which has been moved to `panoptes-pipeline`. (@wtgee #252)
- Docker Tools (@wtgee #248):
  - Remove `source-extractor` from `panoptes-utils` and move to `panoptes-pipeline`. (@wtgee #252)
  - Remove `imagemagick` from `panoptes-utils`. This is used for adding titles to JPGs. (@wtgee #252)
  - Don't install a separate conda environment, just use the base to help reduce image size, complexity. (@wtgee #252)
  - Cleanup unused dependencies. (@wtgee @252)
- Testing:
  - Adios travis! (@wtgee #252)

4.5.2 0.2.28 - 2020-09-15

Added

- Add bit_depth argument to mask_saturated, no longer convert to float64 by default (@AnthonyHorton #244)

Changed

- Single cloudbuild file for both `panoptes-base` and `panoptes-utils`. (#242)
- Add `astropy` channel.
- Remove the miniforge installer from the docker image and clean up build args. (@wtgee #245)
- Changed relative to absolute imports. (@wtgee #246)

4.5.3 0.2.27 - 2020-09-12

Added

- Config server startup controlled via envvars, incorporating `python-dotenv` (@wtgee #241):
  - `PANOPTES_CONFIG_HOST` and `PANOPTES_CONFIG_PORT`
  - `PANOPTES_CONFIG_FILE`

Changed

- Config server updates (@wtgee #241):
  - Config server with project dir mounted can be started via `docker/docker-testing.yaml`.
  - Config server flask instances are run on `gevent` wsgi server instead of development server.
  - `host` and `port` are specified at the top-level command, e.g. `panoptes-config-server --host foobar --port 9999 get`.
  - `host` and `port` respect the above envvars above.
  - Options changed from `auto-save` and `ignore-local` to `save-local` and `load-local`.
  - `run` command adds the‘‘config_server.running=True‘‘ entry to the server.
  - `stop` command added that sets `config_server.running=False` to break loop.
- Testing (@wtgee #241):
  - All testing is started from `scripts/test-software.sh`.
  - The `panoptes-config-server` is started as an external service, not in the pytest conf.
  - Added a `tests/env` file that is used by the docker compose file for setting vars inside the running containers.
  - Config server uses `tests/testing.yaml` for all testing.
  - Testing logs are stored in `./logs`, relative to the project root outside the container.
  - Coverage file is stored in `/var/panoptes/logs`.
  - A `scripts/wait-for-it.sh` script added to check that config server is running properly before starting tests.

Fixed

- `scripts/setup-local-environment.sh` properly uses new base image if requested. (@wtgee #241)
- Docker images: fixed the created `ssh` directory for `$PANUSER`. (@wtgee #240)

4.5.4 0.2.26 - 2020-08-21

This release is mostly cleanup and testing of our autobuild features.

Changed

- Splitting the `panoptes-base` files into separate folder. (#238)
- Consolidate the GitHub Actions for building and publishing a release package. (#239)

Fixed

- Fix Github Actions for building releases. (#238)

4.5.5 0.2.25 - 2020-08-20

Added

- Google Cloud Build of Docker images.
  - `panoptes-base` and `panoptes-utils` are built for each PR as well as on merges to `develop` and `master`. (#237)
- GitHub Actions
  - If a semantically tagged branch is pushed to GH, a release will automatically be generated and a package will be built and sent to PyPi. (#237)

Changed

- Changelog fixes. (#237)

4.5.6 0.2.23 - 2020-08-16

Changed

- Simplified docker docker images. (#227)
  - Consolidation of Dockerfile to support images:
  - `panoptes-base` serves as a base image for all docker services.
  - `panoptes-utils:latest` installs editable `panoptes-utils` module from github `develop` branch.
  - `panoptes-utils:develop` is used for testing and can be built locally with the `docker/setup-local-environment.sh` script.
  - Tests on GH and Travis use the `docker/setup-local-environment.sh` script for building test images.
  - miniforge used to install a `conda` environment with `conda-forge` as default channels. Forces 64bit support.

  - Multi-arch builds are supported on `gcr.io` via the `cloudbuild.yaml` file. Built with `buildx` plugin to docker. Currently `linux/amd64` and `linux/arm64`.
  - Extra zsh plugins in the docker image.
  - Properly disable auto-update of zsh.
  - Cleanup of entrypoint for better loading of environment.

Fixes

- `Pillow` fights.

Removes

- Dependencies: `pyarrow` too hard to build on arm. `hvplot` and `holoviews` not needed in default install.

4.5.7 0.2.22 - 2020-07-25

Changed

- Changed `dir` to `directory` in disk space check. (#226)
- Pass the git folder to the build context when making local docker images. (#226)

4.5.8 0.2.21 - 2020-07-05

Added

- Added `arm64` build for Docker based off `ubuntu` image. (#223)

Changed

- Docker
  - Changed base image to `ubuntu`. (#223)
  - `amd64` and `arm64` images built by default. (#223)
  - Ubuntu has changed `sextractor` to `source-extractor` (yay). (#223)
- Config Server
  - Better parsing of directories entry in config server. (#222)
  - Make config server less noisy. (#222)
- Bump PyYaml to latest for security warning. (#222)
- Remove pendulum because too hard to build on arm processors. (#223)

4.5.9 0.2.20 - 2020-06-09

Moving to python 3.8.

Changed

- **Breaking** Python minimum version changed to `3.8`. (#217)
- Running pytest locally will generate coverage report in terminal. (#218)
- Lots of documentation. (#218)
- Removing the environment section from the readme. (#218)
- Config Server (#217)
  - Better logging.
  - Cleaning up doctests.
  - Removing all dynamic server items from this repo as they are not needed.
  - Wait for config_server to start.
  - Fixing starting within fixture.
  - Config items no longer assume any defaults for either directories or files. A config file name is always required and it should always be an absolute path. (#218)
  - Adding test file for config items. (#218)
  - `panoptes-config-server` re-worked and now includes `run`, `get`, and `set` subcomamnds. (#221)
- Testing (#218)
  - Log files are rotated for each testing run.
  - Fix env vars (mostly need to make sure the `export` option exists in the `env` file.
  - Pytest commands moved to `setup.cfg` instead of `run-tests.sh`
  - Remove old markers
  - Setting `--strict-markers` options.
  - Add `astrometry` marker for tests requiring solve and `theskyx` marker for running alongside TheSkyX.
  - Coverage reports generated in xml and output in terminal.
- Serializers update. (#217)
  - Make the parsing and serializing functions public.
  - Use pendulum for parsing times instead of astropy Time.
  - Better naming of public functions. (#218)

4.5.10 0.2.19 - 2020-06-04

Straight past `0.2.19`.

Changed

- Removed `bin/panoptes-config-server` and created an entry_point in `setup.cfg`. (#212)
- Removed old developer items in favor of those in `panoptes-pocs`. (#212)
- Consolidate docker files, consistent naming with other repos. (#210, #212)

4.5.11 0.2.17 - 2020-05-30

`0.2.16` was released with an error and this is a hotfix.

Added

- Added CR2 file testing to GitHub Actions. (#125, #205)
- A `wait_for_events` generic utility, mostly pulled from POCS. (#92, #206) * Supports single `callback` that can be used for interrupting, custom logging, etc. (#208)

Changed

- Remove the `validate_collection` requirement from the database types, making any collection is now valid. (#204)
- Rearrange some of the `panoptes.utils.database` modules. (#204)

Removed

- Remove `error.InvalidCollection`. (#204)
- Unused items in `conftest.py`. (#204)

4.5.12 0.2.15 - 2020-05-26

Changed

- Convert to pyscaffold. (#198)
  - Proper namespace package `panoptes`.
  - Move items to `src` folder.
  - Fix version number.
  - Fix build.
  - Fix documentation #27.
  - Move all project config to `setup.cfg`.
  - Base Docker image is run by root only.
  - Added a `testing` Dockerfile and cleaned up `latest` and `develop`.

Removed

- **Breaking** Removing all zmq based messaging services. (#200)

4.5.13 0.2.14 - 2020-05-23

Added

- Add snappy decompression for parquet; `pyarrow` instead of `fastparquet` (#193)
- Password-less sudo for panoptes user on dev docker image (#193)
- `get_metadata` has an optional progress bar. (#194)
- Add `bayer.get_stamp_slice` for getting a stamp slice while respecting the superpixel. This was removed awhile ago and has been re-added and improved. (#196) * Adjusting the offsets so the center pixel is always:

```
G2 B
R  G1
```

Bug fixes

- Fix time-based search (#193)
- Fix the build (#197) * Removed `versioneer` in favor of `setuptools-scm` for workin version numbers. * Removed the MANIFEST.in * Added a simple `pyproject.toml`.

Changed

- **Breaking** Only support getting stars directly from the WCS, not the footprint. (#194) * `get_stars_from_footprint` -> `get_stars_from_wcs` * Better logging * Consistent column names for dtypes * Vmag bin comes from sql. * Allow for different RA/Dec column names. * Better catalog match function.
- `sextractor` param changes. (#194)
- **Breaking** `panoptes.utils.logger` -> `panoptes.utils.logger` so we can `from panoptes.utils.logging import logger` (#197)
- **Breaking The `panoptes.utils.data.assets` module was removed and the** `Downloader` class is placed directly within the `scripts/download-data.py` file. (#197)
- The `panopes-utils` module is not installed in editable mode in the `latest` docker image. (#197) * Slight clean up of latest.Dockerfile

4.5.14 0.2.13 - 2020-05-14

Bug fixes

- Fix some passing of options between `get_solve_field` and `solve_field` that was leading to double parameter issues. (#189)

Changed

- The `panoptes.utils.data` functions use static versions of the file rather than firestore. (#192)
- Updated development environment (#191)
- `get_metadata` filter the fields at the parquet level. (#194)

4.5.15 0.2.12 - 2020-04-29

Quick release to get the `panoptes.utils.sources` into the package.

Bug fixes

- `panoptes.utils.sources` not included in package. (#187, #188)

Changed

- Ability to pass credentials to underlying google client functions. (#187)

4.5.16 0.2.11 - 2020-04-29

Added

- **Data**
  - Added basic data access components for getting observation and image metadata. (#178, #181)
  - **Added a `search_observations` function for searching by various criteria. (#181)**
    - Uses anonymous credentials to connect to firestore.
    - Added a basic notebook demonstrating features.
  - Adding `holoviews` and `hvplot` as required dependencies.

Bug fixes

- **FITS Utils fixes:**
  - Fix docstring return types for some functions. (#173)
  - **`fpack`/`funpack` and `get_solve_field` were not properly overwriting FITS files** under certain conditions when an uncompressed file of the same name was present alongside the compressed version. (#175)
  - Properly pass `args` and `kwargs` to `astropy.io.fits.getdata`. (#180)

Changed

- **Docker**
  - Changed developer tag from `dev` to `develop`. (#174)
- **FITS Utils changes (#173):**
  - Uncompressed file is always used for solve because we were occasionally seeing odd errors as described in dstndstn/astrometry.net#182. (#173)
  - **warning** `get_solve_field` will `overwrite` by default.
  - Better log output for solving.
  - Better checking for solved file at end (via `is_celestial`).
  - Cleanup the cleanup of solve files, removing `remove_extras` option.

  - Pass `kwargs` to underlying `writeto` method for `write_fits`. Needed for, e.g. `overwrite`.
  - Allow additional options to be passed to solve field functions without having to override all options. (#180)
  - **Changed default options in `get_solve_field` to use `scale-low` and `scale-high` instead of `radius` (whi** requires an `ra` and `dec`). (#180)
- Changed `bin/panoptes-dev` -> `bin/panoptes-develop` for naming consistency. (#175)
- **Data**
  - **BREAKING** The `panoptes.utils.data.py` has moved into the `panoptes.utils.data` namespace with the relevant existing `Downloader` class placed in the `assets.py` module. (#181)
  - Changed the `get_data` (and images and observations equivalent) to `get_metadata`. (#181)

Removed

FITS Utils removals (#173):

- Removing unused and confusing `improve_wcs`.
- PanLogger class moved to POCS. (#186)

4.5.17 0.2.10 - 2020-04-13

Added

- `get_stars_from_footpr int` can accept a `WCS` directly instead of just the output from `calc_footprint()`. (#164)
- Ability to create different tags for the docker image. The `develop` directory is now used to create a `develop` image and is provided along with `latest`. (#165)
- `get_rgb_backgrounds(return_separate=True)` will now return the `Background2D` objects. (#166)
- Added BigQuery pandas dependencies. (#168)
- Added a developer image at `panoptes-utils:dev`, which is also auto-built along with the `latest` in the cloudbuild. Offers a `jupyter-lab` instance along with a number of plotting modules. Can be easily started via `panoptes-dev`. (#170, #171)

Bug fixes

- `image_id_from_path` and `sequence_id_from_path` can recognize a zero in the `camera_id` and `None` when no match. (#163)
- Fixed the bigquery client param for star lookup. (#164)
- Unquote paths before id matching. (#169)
- Do WCS match for all unmatched sources, not just matched sources. (#172)

Changed

- Docker entrypoint no longer tries to activate service account if `$GOOGLE_APPLICATION_CREDENTIALS` is found. The python client libraries will recognize the env var so this means we can avoid installing `gcloud` utilities just to activate. (#165)
- The `sources` module does not require a BigQuery client to be passed but can start it's own. A warning is given if `$GOOGLE_APPLICATION_CREDENTIALS` is not found. (#167)
- `lookup_point_sources` updates: default vmag range expanded so less false positive matches [4,18). (#168)
- Removed TOC from changelog. (#170)
- Sextractor param changes: (#171) * Threshold for detection changed from 3 pixels to 10 pixels. * Seeing changed from 0.7 arcsec to 15.3 arcsec. (Isn't used.) * Removed `class_star` from sextractor results.

4.5.18 0.2.9 - 2020-03-27

Pointless version bump because of issue with PyPi.

4.5.19 0.2.8 - 2020-03-27

Thanks first-time contributer @preethi524! :tada:

Changed

- Ability to return separate RGB backgrounds. (#162)
- Increase coverage. (#161)

4.5.20 0.2.7 - 2020-03-22 (hotfix)

Added

- Basic serialization of `Exception`. (#160)

Bug fixes

- Add `args` and `kwargs` to `get_rgb_background`. (#160)

4.5.21 0.2.6 - 2020-03-22

Added

- `get_rgb_background` added to the `bayer` module. (#158)
- `getwcs` thin-wrapper added to `fits` module. (#158)
- Added `sources` utils. (#158)

Bug fixes

- Changed scope of test data files to `function`. (#158)

Changed

- Docker
  - Change to `python:3.8-slim-buster` for base image. Only `amd64` support for now. (#155)
  - Simplified docker files. (#155)
  - Switching from Travis to GHA: (#155)
  - Travis builds docker image before testing.
  - Travis doesn't upload coverage.
  - Don't update module inside container during entrypoint.
  - Fixed user permissions for $HOME and $PANDIR. (#155)
  - The docker container only really likes it when user id `1000` is running the system.
  - Remove GCP Cloud SQL proxy support.
  - Installed `sextractor`. (#158)
  - Added `pandas`. (#158)
  - Default `panoptes` user has password `panoptes`. (#158)

Removed

- Docker (#155)
  - Remove anaconda
- Polar alignment utils (#156)

4.5.22 0.2.5 - 2020-03-18

Added

- Github Actions testing and coverage upload. (#145) * Log files for testing are created as an artifact.
- `PanLogger` helper class added. Mostly handles formatting but can also track handlers. (#145)

Bug fixes

- Fixed top-level namespace so we can have other `panoptes` repos. (#150, fixes #137)

Changed

- Data files for testing are copied before tests. Allows for reuse of unsolved fits file. (#144)
- Fix astrometry data file directories in Docker images. (#144)

Removed

- The docker image no longer updates `panoptes-utils` when using `run-tests.sh`. (#145)

4.5.23 0.2.4 - 2020-03-11

Changed

- Disallow zipped packages, which also interfere with namespace (#142)

Removed

- `photutils` dependency for rectangular apertures in the `show_stamps` method.

4.5.24 0.2.3 - 2020-03-08

Small point release to correct namespace and remove some bloat.

Changed

- Fixed top-level namespace so we can have other `panoptes` repos. (#137)

Removed

- Dependencies that will be deprecated soon and are causing bloat: `photutils`, `scikit-image`. (#138)

Changed

- Fixed top-level namespace so we can have other `panoptes` repos (#137, #150).

4.5.25 0.2.2 - 2020-03-05

Mostly some cleanup from the `v0.2.0` release based on integrating all the changes into POCS.

Bug fixes

- Misc bugs introduced as part of last release, including to `download-data.py` script.
- Custom exceptions now properly pass `kwargs` through to parent (#135).

Changed

- New script for downloading data, `scripts/download-data.py`. This helped resolve some issues with the relative imports introduced in `v0.2.0` and is cleaner. (#129)
- All dependencies are smashed into one "feature" in `setup.py` to make `pip-tools` work well. This will fix the docker image problems introduced in `v0.2.1`. (#136)

Removed

- The `get_root_logger` and associated tests (#134).

4.5.26 0.2.0 - 2020-03-04

[0.2.0] - 2020-03-04 First big overhaul of the repository. Pulls in features that were duplicated or scattered across POCS and PIAA. Removes a lot of code that wasn't being used or was otherwise clutter. Overhauls the logging system to use [`loguru`](https://github.com/Delgan/loguru) so things are simplified. Updates to documentation.

Added

- Config Server
- See the description in the [README](README.md)
- Versioneer for version strings (#123).
- Read the docs config (#123).

Bug fixes

- IERS Mirror (#65, #67)

Changed

- Docker updates
- See #68 and #75 for list.
- Logging:
- Switch to loguru. This simplifies our logging system. Also gives us access to the `trace` (lower than `debug`, good for hardware and other debug we don't need to see during operation) and `success` (higher than `info`) levels, which would be nice to start implementing. (#123)
- Consistent use of relative imports. (#123)
- Documentation updates. (#97, #119, #120, #123)
- Repo cleanup. (#97, #123)
- Using GitHub Actions for testing. (#100, #101)
- Using pip-tools for dependency management.

4.5.27 0.1.0 - 2020-03-04

Changes and cleanup on the way to a (more) stable release. See `0.2.0` for list of changes.

4.5.28 0.0.8 - 2019-06-29

Bringing things in line with updates to `POCS` for docker and `panoptes-utils` use.

Added

- Serial handlers move to panoptes-utils from POCS.
- Tests and coverage.
- `improve_wcs` (moved from PIAA).
- `~utils.fits.getdata` to match other fits convenience functions, allowing for fpack files.

Bug fixes

- Serialization fixes.
  - Use our serialization everywhere (e.g. messaging)
  - Closes #panoptes/POCS/issues/818
  - Closes #panoptes/POCS/issues/103

Changed

- Setup/Install:
  - Scripts are renamed to have `panoptes` prefix.
  - Scripts are installed as part of setup.
  - Script improvements to make more robust and portable.
- Docker Updates:
  - Don't use anaconda.
- Testing:
  - Overhaul of config_server use in testing.
  - Testing config file is separated from any regular config files.
- Logging:
  - Silence some 3rd party logs.

4.5.29 0.0.7 - 2019-05-26

Added

- Added bayer utilities. :camera:
- Added Cloud SQL utilities. :cloud:

Changed

- **Breaking** Changed namespace so no underscores, i.e. `from panoptes.utils import time`.
- Docker updates:
  - Use slim python images and not anaconda on amd64.
  - Adding zsh as default shell along with some customizations.

  - Entrypoint script properly authenticates to google cloud if possible.
  - Added amd64 only build scripts.

4.5.30 0.0.6 - 2019-04-29

Added

- Docker containers created:
  - `panoptes-base` for base OS and system packages, including astrometry.net and friends.
  - `panoptes-utils` for container containing base utilities.
  - Script for building containers in GCR.
- Consistent JSON and YAML serializers.
- Configuration Server (Flask/JSON microservice).

Changed

- **Minimum Python version is 3.6**
- Default PanDB type is changed to `memory`.
- Documentation updates.
- Bux fixes and code improvements.

4.5.31 0.0.5 - 2019-04-09

Added

- Added a change log. Yay.

Changed

- Drop `orjson` and revert to `json` for the JSON serializers.

The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.0.0/), and this project adheres to [Semantic Versioning](https://semver.org/spec/v2.0.0.html).

4.6 panoptes

4.6.1 panoptes package

Subpackages

panoptes.utils package

Subpackages

panoptes.utils.config package

Submodules

panoptes.utils.config.cli module

panoptes.utils.config.client module

panoptes.utils.config.client.**get_config**(*key=None*, *host=None*, *port=None*, *endpoint='get-config'*, *parse=True*, *default=None*, *verbose=True*)

Get a config item from the config server.

Return the config entry for the given `key`. If `key=None` (default), return the entire config.

Nested keys can be specified as a string, as per scalpl.

Examples:

```
>>> get_config(key='name')
'Testing PANOPTES Unit'

>>> get_config(key='location.horizon')
<Quantity 30. deg>

>>> # With no parsing, the raw string (including quotes) is returned.
>>> get_config(key='location.horizon', parse=False)
'"30.0 deg"'
>>> get_config(key='cameras.devices[1].model')
'canon_gphoto2'

>>> # Returns `None` if key is not found.
>>> foobar = get_config(key='foobar')
>>> foobar is None
True

>>> # But you can supply a default.
>>> get_config(key='foobar', default='baz')
'baz'

>>> # Can use Quantities as well
>>> from astropy import units as u
>>> get_config(key='foobar', default=42 * u.meter)
<Quantity 42. m>
```

Notes

By default all calls to this function will log at the *trace* level because there are some calls (e.g. during POCS operation) that will be quite noisy.

Setting *verbose=True* changes those to *debug* log levels for an individual call.

Parameters

- **key** (*str*) – The key to update, see Examples in *get_config()* for details.
- **host** (*str, optional*) – The config server host. First checks for PANOPTES_CONFIG_HOST env var, defaults to 'localhost'.

- **port** (*str or int, optional*) – The config server port. First checks for PANOPTES_CONFIG_HOST env var, defaults to 6563.
- **endpoint** (*str, optional*) – The relative url endpoint to use for getting the config items, default 'get-config'. See *server_is_running()* for example of usage.
- **parse** (*bool, optional*) – If response should be parsed by *panoptes.utils.serializers.from_json()*, default True.
- **default** (*str, optional*) – The config server port, defaults to 6563.
- **verbose** (*bool, optional*) – Determines the output log level, defaults to True (i.e. *debug* log level). See notes for details.

Returns The corresponding config entry.

Return type dict

Raises Exception – Raised if the config server is not available.

panoptes.utils.config.client.**server_is_running**()
Thin-wrapper to check server.

panoptes.utils.config.client.**set_config**(*key*, *new_value*, *host=None*, *port=None*, *parse=True*)
Set config item in config server.

Given a *key* entry, update the config to match. The *key* is a dot accessible string, as given by scalpl. See Examples in *get_config()* for details.

Examples:

```
>>> from astropy import units as u

>>> # Can use astropy units.
>>> set_config('location.horizon', 35 * u.degree)
{'location.horizon': <Quantity 35. deg>}

>>> get_config(key='location.horizon')
<Quantity 35. deg>

>>> # String equivalent works for 'deg', 'm', 's'.
>>> set_config('location.horizon', '30 deg')
{'location.horizon': <Quantity 30. deg>}
```

Parameters

- **key** (*str*) – The key to update, see Examples in *get_config()* for details.
- **new_value** (*scalar|object*) – The new value for the key, can be any serializable object.
- **host** (*str, optional*) – The config server host. First checks for PANOPTES_CONFIG_HOST env var, defaults to 'localhost'.
- **port** (*str or int, optional*) – The config server port. First checks for PANOPTES_CONFIG_HOST env var, defaults to 6563.
- **parse** (*bool, optional*) – If response should be parsed by *panoptes.utils.serializers.from_json()*, default True.

Returns The updated config entry.

Return type dict

Raises Exception – Raised if the config server is not available.

panoptes.utils.config.helpers module

panoptes.utils.config.helpers.**load_config**(*config_files=None*, *parse=True*, *load_local=True*)

Load configuration information.

Note: This function is used by the config server and normal config usage should be via a running config server.

This function supports loading of a number of different files. If no options are passed to `config_files` then the default `$PANDIR/conf_files/pocs.yaml` will be loaded.

`config_files` is a list and loaded in order, so the second entry will overwrite any values specified by similarly named keys in the first entry.

`config_files` should be specified by an absolute path, which can exist anywhere on the filesystem.

Local versions of files can override built-in versions and are automatically loaded if they exist alongside the specified config path. Local files have a `<>_local.yaml` name, where `<>` is the built-in file.

Given the following path:

```
/path/to/dir
|- my_conf.yaml
|- my_conf_local.yaml
```

You can do a `load_config('/path/to/dir/my_conf.yaml')` and both versions of the file will be loaded, with the values in the local file overriding the non-local. Typically the local file would also be ignored by `git`, etc.

For example, the `panoptes.utils.config.server.config_server` will always save values to a local version of the file so the default settings can always be recovered if necessary.

Local files can be ignored (mostly for testing purposes or for recovering default values) with the `load_local=False` parameter.

Parameters

- **config_files** (*list, optional*) – A list of files to load as config, see Notes for details of how to specify files.
- **parse** (*bool, optional*) – If the config file should attempt to create objects such as dates, astropy units, etc.
- **load_local** (*bool, optional*) – If local files should be used, see Notes for details.

Returns A dictionary of config items.

Return type dict

panoptes.utils.config.helpers.**parse_config_directories**(*directories*, *must_exist=False*)

Parse the config dictionary for common objects.

Given a *base* entry that corresponds to the absolute path of a directory, prepend the *base* to all other relative directory entries.

If *must_exist=True*, then only update entry if the corresponding directory exists on the filesystem.

```
>>> dirs_config = dict(base='/var/panoptes', foo='bar', baz='bam')
>>> # If the relative dir doesn't exist but is required, return as is.
>>> parse_config_directories(dirs_config, must_exist=True)
{'base': '/var/panoptes', 'foo': 'bar', 'baz': 'bam'}

>>> # Default is to return anyway.
>>> parse_config_directories(dirs_config)
{'base': '/var/panoptes', 'foo': '/var/panoptes/bar', 'baz': '/var/panoptes/bam'}

>>> # If 'base' is not a valid absolute directory, return all as is.
>>> dirs_config = dict(base='panoptes', foo='bar', baz='bam')
>>> parse_config_directories(dirs_config, must_exist=False)
{'base': 'panoptes', 'foo': 'bar', 'baz': 'bam'}
```

Parameters

- **directories** (*dict*) – The dictionary of directory information. Usually comes from the "directories" entry in the config.
- **must_exist** (*bool*) – Only parse directory if it exists on the filesystem, default False.

Returns The same directory but with relative directories resolved.

Return type dict

panoptes.utils.config.helpers.**save_config**(*path*, *config*, *overwrite=True*)

Save config to local yaml file.

Parameters

- **path** (*str*) – Path to save, can be relative or absolute. See Notes in `load_config`.
- **config** (*dict*) – Config to save.
- **overwrite** (*bool, optional*) – True if file should be updated, False to generate a warning for existing config. Defaults to True for updates.

Returns If the save was successful.

Return type bool

Raises `FileExistsError` – If the local path already exists and `overwrite=False`.

panoptes.utils.config.server module

class panoptes.utils.config.server.**CustomJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `flask.json.JSONEncoder`

default(*obj*)

Custom serialization of each object.

This method will call *panoptes.utils.serializers.serialize_object()* for each object.

Parameters **obj** (*any*) – The object to serialize.

panoptes.utils.config.server.**config_server**(*config_file*, *host=None*, *port=None*, *load_local=True*, *save_local=False*, *auto_start=True*, *access_logs=None*, *error_logs='logger'*)

Start the config server in a separate process.

A convenience function to start the config server.

Parameters

- **config_file** (*str or None*) – The absolute path to the config file to load. Checks for PANOPTES_CONFIG_FILE env var and fails if not provided.
- **host** (*str, optional*) – The config server host. First checks for PANOPTES_CONFIG_HOST env var, defaults to 'localhost'.
- **port** (*str or int, optional*) – The config server port. First checks for PANOPTES_CONFIG_HOST env var, defaults to 6563.
- **load_local** (*bool, optional*) – If local config files should be used when loading, default True.
- **save_local** (*bool, optional*) – If setting new values should auto-save to local file, default False.
- **auto_start** (*bool, optional*) – If server process should be started automatically, default True.
- **access_logs** ('default' or *logger* or *File*-like or None, optional) – Controls access logs for the gevent WSGIServer. The *default* string will cause access logs to go to stderr. The string *logger* will use the panoptes logger. A File-like will write to file. The default *None* will turn off all access logs.
- **error_logs** ('default' or 'logger' or *File*-like or None, optional) – Same as *access_logs* except we use our *logger* as the default.

Returns The process running the config server.

Return type multiprocessing.Process

panoptes.utils.config.server.**get_config_entry**()

Get config entries from server.

Endpoint that responds to GET and POST requests and returns configuration item corresponding to provided key or entire configuration. The key entries should be specified in dot-notation, with the names corresponding to the entries stored in the configuration file. See the scalpl documentation for details on the dot-notation.

The endpoint should receive a JSON document with a single key named `"key"` and a value that corresponds to the desired key within the configuration.

For example, take the following configuration:

```
{
    'location': {
        'elevation': 3400.0,
        'latitude': 19.55,
        'longitude': 155.12,
    }
}
```

To get the corresponding value for the elevation, pass a JSON document similar to:

```
'{"key": "location.elevation"}'
```

Returns The json string for the requested object if object is found in config. Otherwise a json string with `status` and `msg` keys will be returned.

Return type str

panoptes.utils.config.server.**heartbeat**()
A simple echo service to be used for a heartbeat.

Defaults to looking for the 'config_server.running' bool value, although a different *key* can be specified in the POST.

panoptes.utils.config.server.**reset_config**()
Reset the configuration.

An endpoint that accepts a POST method. The json request object must contain the key `reset` (with any value).

The method will reset the configuration to the original configuration files that were used, skipping the local (and saved file).

Note: If the server was originally started with a local version of the file, those will be skipped upon reload. This is not ideal but hopefully this method is not used too much.

Returns A json string object containing the keys `success` and `msg` that indicate success or failure.

Return type str

panoptes.utils.config.server.**set_config_entry**()
Sets an item in the config.

Endpoint that responds to GET and POST requests and sets a configuration item corresponding to the provided key.

The key entries should be specified in dot-notation, with the names corresponding to the entries stored in the configuration file. See the scalpl documentation for details on the dot-notation.

The endpoint should receive a JSON document with a single key named `"key"` and a value that corresponds to the desired key within the configuration.

For example, take the following configuration:

```
{
    'location': {
        'elevation': 3400.0,
        'latitude': 19.55,
        'longitude': 155.12,
    }
}
```

To set the corresponding value for the elevation, pass a JSON document similar to:

```
'{"location.elevation": "1000 m"}'
```

Returns If method is successful, returned json string will be a copy of the set values. On failure, a json string with `status` and `msg` keys will be returned.

Return type str

Module contents

panoptes.utils.database package

Submodules

panoptes.utils.database.base module

class panoptes.utils.database.base.**AbstractPanDB**(*db_name=None*, ***kwargs*)
Bases: object

__init__(*db_name=None*, ***kwargs*)
Init base class for db instances.

Parameters **db_name** – Name of the database, typically 'panoptes' or 'panoptes_testing'.

clear_current(*type*)
Clear the current record of a certain type

Parameters **type** (*str*) – The type of entry in the current collection that should be cleared.

find(*collection*, *obj_id*)
Find an object by it's identifier.

Parameters

- **collection** (*str*) – Collection to search for object.
- **obj_id** (*ObjectID|str*) – Record identifier returned earlier by insert or insert_current.

Returns Object matching identifier or None.

Return type dict|None

get_current(*collection*)
Returns the most current record for the given collection

Parameters **collection** (*str*) – Name of the collection to get most current from

Returns Current object of the collection or None.

Return type dict|None

insert(*collection*, *obj*)
Insert an object into the collection provided.

The *obj* to be stored in a collection should include the *type* and *date* metadata as well as a *data* key that contains the actual object data. If these keys are not provided then *obj* will be wrapped in a corresponding object that does contain the metadata.

Parameters

- **collection** (*str*) – Name of valid collection within the db.
- **obj** (*dict or str*) – Object to be inserted.

Returns

identifier of inserted record in *collection*. Returns None if unable to insert into the collection.

Return type str

insert_current(*collection*, *obj*, *store_permanently=True*)
Insert an object into both the *current* collection and the collection provided.

Parameters

- **collection** (*str*) – Name of valid collection within the db.
- **obj** (*dict or str*) – Object to be inserted.
- **store_permanently** (*bool*) – Whether to also update the collection, defaults to True.

Returns

identifier of inserted record. If *store_permanently* is True, will be the identifier of the object in the *collection*, otherwise will be the identifier of object in the *current* collection. These may or may not be the same. Returns None if unable to insert into the collection.

Return type str

class panoptes.utils.database.base.**PanDB**
Bases: object

Simple class to load the appropriate DB type based on the config.

We don't actually create instances of this class, but instead create an instance of the 'correct' type of db.

classmethod permanently_erase_database(*db_type*, *db_name*, *storage_dir=None*, *really=False*, *dangerous=False*, **args*, ***kwargs*)
Permanently delete the contents of the identified database.

panoptes.utils.database.base.**create_storage_obj**(*collection*, *data*, *obj_id*)
Wrap the data in a dict along with the id and a timestamp.

panoptes.utils.database.base.**get_db_class**(*module_name='file'*)
Load the main DB class for the module of the given name.

Note: This is used by the *PanDB* constructor to determine the correct database type. Normal DB instantiation should be done via the *PanDB()* class with the desired *db_type* parameter set. See example in *PanDB* below.

Parameters **module_name** (*str*) – Name of module, one of: *file* (default), 'memory'.

Returns An instance of the db class for the correct database type.

Return type *panoptes.utils.database.PanDB*

Raises Exception – If an unsupported database type string is passed.

panoptes.utils.database.file module

class panoptes.utils.database.file.**PanFileDB**(*db_name='panoptes'*, *storage_dir='json_store'*, ***kwargs*)
Bases: *panoptes.utils.database.base.AbstractPanDB*

Stores collections as files of JSON records.

__init__(*db_name='panoptes'*, *storage_dir='json_store'*, ***kwargs*)
Flat file storage for json records.

This will simply store each json record inside a file corresponding to the type. Each entry will be stored in a single line.

Parameters

- **db_name** (*str, optional*) – Name of the database containing the collections.
- **storage_dir** (*str, optional*) – The name of the directory in $PANDIR where the database files will be stored. Default is *json_store* for backwards compatibility.

clear_current(*record_type*)
Clears the current record of the given type.

Parameters **record_type** (*str*) – The record type, e.g. 'weather', 'environment', etc.

find(*collection*, *obj_id*)
Find an object by it's identifier.

Parameters

- **collection** (*str*) – Collection to search for object.
- **obj_id** (*ObjectID|str*) – Record identifier returned earlier by insert or insert_current.

Returns Object matching identifier or None.

Return type dict|None

get_current(*collection*)
Returns the most current record for the given collection

Parameters **collection** (*str*) – Name of the collection to get most current from

Returns Current object of the collection or None.

Return type dict|None

insert(*collection*, *obj*)
Insert an object into the collection provided.

The *obj* to be stored in a collection should include the *type* and *date* metadata as well as a *data* key that contains the actual object data. If these keys are not provided then *obj* will be wrapped in a corresponding object that does contain the metadata.

Parameters

- **collection** (*str*) – Name of valid collection within the db.
- **obj** (*dict or str*) – Object to be inserted.

Returns

identifier of inserted record in *collection*. Returns None if unable to insert into the collection.

Return type str

insert_current(*collection*, *obj*, *store_permanently=True*)
Insert an object into both the *current* collection and the collection provided.

Parameters

- **collection** (*str*) – Name of valid collection within the db.
- **obj** (*dict or str*) – Object to be inserted.
- **store_permanently** (*bool*) – Whether to also update the collection, defaults to True.

Returns

identifier of inserted record. If *store_permanently* is True, will be the identifier of the object in the *collection*, otherwise will be the identifier of object in the *current* collection. These may or may not be the same. Returns None if unable to insert into the collection.

Return type str

classmethod permanently_erase_database(*db_name*, *storage_dir='json_store'*)

panoptes.utils.database.memory module

class panoptes.utils.database.memory.**PanMemoryDB**(***kwargs*)
Bases: *panoptes.utils.database.base.AbstractPanDB*

In-memory store of serialized objects.

We serialize the objects in order to test the same code path used when storing in an external database.

active_dbs = <WeakValueDictionary>

clear_current(*entry_type*)
Clear the current record of a certain type

Parameters **type** (*str*) – The type of entry in the current collection that should be cleared.

find(*collection*, *obj_id*)
Find an object by it's identifier.

Parameters

- **collection** (*str*) – Collection to search for object.
- **obj_id** (*ObjectID|str*) – Record identifier returned earlier by insert or insert_current.

Returns Object matching identifier or None.

Return type dict|None

get_current(*collection*)
Returns the most current record for the given collection

Parameters **collection** (*str*) – Name of the collection to get most current from

Returns Current object of the collection or None.

Return type dict|None

classmethod get_or_create(*db_name=None*, ***kwargs*)
Returns the named db, creating if needed.

This method exists because PanDB gets called multiple times for the same database name. With mongo or a file store where the storage is external from the instance, that is not a problem, but with PanMemoryDB the instance is the store, so the instance must be shared.

insert(*collection*, *obj*)
Insert an object into the collection provided.

The *obj* to be stored in a collection should include the *type* and *date* metadata as well as a *data* key that contains the actual object data. If these keys are not provided then *obj* will be wrapped in a corresponding object that does contain the metadata.

Parameters

- **collection** (*str*) – Name of valid collection within the db.

- **obj** (*dict or str*) – Object to be inserted.

Returns

identifier of inserted record in *collection*. Returns None if unable to insert into the collection.

Return type str

insert_current(*collection*, *obj*, *store_permanently=True*)

Insert an object into both the *current* collection and the collection provided.

Parameters

- **collection** (*str*) – Name of valid collection within the db.
- **obj** (*dict or str*) – Object to be inserted.
- **store_permanently** (*bool*) – Whether to also update the collection, defaults to True.

Returns

identifier of inserted record. If *store_permanently* is True, will be the identifier of the object in the *collection*, otherwise will be the identifier of object in the *current* collection. These may or may not be the same. Returns None if unable to insert into the collection.

Return type str

classmethod permanently_erase_database(**args*, ***kwargs*)

Module contents

panoptes.utils.images package

Submodules

panoptes.utils.images.bayer module

panoptes.utils.images.bayer.**get_pixel_color**(*x*, *y*)

Given a zero-indexed x,y position, return the corresponding color.

Note: See *get_rgb_data()* for a description of the RGGB pattern.

Returns one of 'R', 'G1', 'G2', 'B'

Return type str

panoptes.utils.images.bayer.**get_rgb_background**(*fits_fn*, *box_size=(84, 84)*, *filter_size=(3, 3)*, *camera_bias=0*, *estimator='mean'*, *interpolator='zoom'*, *sigma=5*, *iters=5*, *exclude_percentile=100*, *return_separate=False*, **args*, ***kwargs*)

Get the background for each color channel.

Most of the options are described in the *photutils.Background2D* page: https://photutils.readthedocs.io/en/stable/background.html#d-background-and-noise-estimation

```
>>> from panoptes.utils.images import fits as fits_utils
>>> fits_fn = getfixture('solved_fits_file')
```

```
>>> data = fits_utils.getdata(fits_fn)
>>> data.mean()
2236.816...
```

```
>>> rgb_back = get_rgb_background(fits_fn)
>>> rgb_back.mean()
2202.392...
```

```
>>> rgb_backs = get_rgb_background(fits_fn, return_separate=True)
>>> rgb_backs[0]
<photutils.background.background_2d.Background2D...>
>>> {color:data.background_rms_median for color, data in zip('rgb', rgb_backs)}
{'r': 20.566..., 'g': 32.787..., 'b': 23.820...}
```

Parameters

- **fits_fn** (*str*) – The filename of the FITS image.
- **box_size** (*tuple, optional*) – The box size over which to compute the 2D-Background, default (84, 84).
- **filter_size** (*tuple, optional*) – The filter size for determining the median, default (3, 3).
- **camera_bias** (*int, optional*) – The built-in camera bias, default 0. A zero camera bias means the bias will be considered as part of the background.
- **estimator** (*str, optional*) – The estimator object to use, default 'median'.
- **interpolator** (*str, optional*) – The interpolater object to user, default 'zoom'.
- **sigma** (*int, optional*) – The sigma on which to filter values, default 5.
- **iters** (*int, optional*) – The number of iterations to sigma filter, default 5.
- **exclude_percentile** (*int, optional*) – The percentage of the data (per channel) that can be masked, default 100 (i.e. all).
- **return_separate** (*bool, optional*) – If the function should return a separate array for color channel, default False.
- ***args** – Description
- ****kwargs** – Description

Returns

Either a single numpy array representing the entire background, or a list of masked numpy arrays in RGB order. The background for each channel has full interploation across all pixels, but the mask covers them.

Return type 'numpy.array'|list

panoptes.utils.images.bayer.**get_rgb_data**(*data*, *separate_green=False*)

Get the data split into separate channels for RGB.

data can be a 2D (*W x H*) or 3D (*N x W x H*) array where W=width and H=height of the data, with N=number of frames.

The return array will be a *3 x W x H* or *3 x N x W x H* array.

The Bayer array defines a superpixel as a collection of 4 pixels set in a square grid:

```
R G
G B
```

ds9 and other image viewers define the coordinate axis from the lower left corner of the image, which is how a traditional x-y plane is defined and how most images would expect to look when viewed. This means that the *(0, 0)* coordinate position will be in the lower left corner of the image.

When the data is loaded into a *numpy* array the data is flipped on the vertical axis in order to maintain the same indexing/slicing features. This means the the `(0, 0)` coordinate position is in the upper-left corner of the array when output. When plotting this array one can use the `origin='lower'` option to view the array as would be expected in a normal image although this does not change the actual index.

Image dimensions:

```
---------------------------
x | width  | i | columns |  5208
y | height | j | rows    |  3476
```

Bayer pattern as seen in ds9:

```
                          x / j

            0     1    2     3 ... 5204 5205 5206 5207
          ----------------------------------------------
      3475 |  R   G1    R    G1        R   G1    R   G1
      3474 | G2    B   G2     B       G2    B   G2    B
      3473 |  R   G1    R    G1        R   G1    R   G1
      3472 | G2    B   G2     B       G2    B   G2    B
         . |
y / i    . |
         . |
         3 |  R   G1    R    G1        R   G1    R   G1
         2 | G2    B   G2     B       G2    B   G2    B
         1 |  R   G1    R    G1        R   G1    R   G1
         0 | G2    B   G2     B       G2    B   G2    B
```

The RGGB super-pixels thus start in the upper-left.

Bayer pattern as seen in a numpy array:

```
                          x / j

            0     1    2     3 ... 5204 5205 5206 5207
          ----------------------------------------------
         0 | G2    B   G2     B       G2    B   G2    B
         1 |  R   G1    R    G1        R   G1    R   G1
         2 | G2    B   G2     B       G2    B   G2    B
         3 |  R   G1    R    G1        R   G1    R   G1
         . |
y / i    . |
         . |
      3472 | G2    B   G2     B       G2    B   G2    B
      3473 |  R   G1    R    G1        R   G1    R   G1
      3474 | G2    B   G2     B       G2    B   G2    B
      3475 |  R   G1    R    G1        R   G1    R   G1
```

Here the RGGB super-pixels are flipped upside down.

In both cases the data is in the following format:

```
    | row (y) |  col (x)
--------------| ------
 R  |  odd i, |  even j
 G1 |  odd i, |   odd j
 G2 | even i, |  even j
 B  | even i, |   odd j
```

And a mask can therefore be generated as:

```
bayer[1::2, 0::2] = 1 # Red
bayer[1::2, 1::2] = 1 # Green
bayer[0::2, 0::2] = 1 # Green
bayer[0::2, 1::2] = 1 # Blue
```

panoptes.utils.images.bayer.**get_rgb_masks**(*data*, *separate_green=False*)

Get the RGGB Bayer pattern for the given data.

Note: See *get_rgb_data()* for a description of the RGGB pattern.

Parameters

- **data** (*np.array*) – An array of data representing an image.
- **separate_green** (*bool, optional*) – If the two green channels should be separated, default False.

Returns A 3-tuple of numpy arrays of *bool* type.

Return type tuple(np.array, np.array, np.array)

panoptes.utils.images.bayer.**get_stamp_slice**(*x*, *y*, *stamp_size=(14, 14)*, *ignore_superpixel=False*)

Get the slice around a given position with fixed Bayer pattern.

Given an x,y pixel position, get the slice object for a stamp of a given size but make sure the first position corresponds to a red-pixel. This means that x,y will not necessarily be at the center of the resulting stamp.

```
>>> from panoptes.utils.images import bayer
>>> # Make a super-pixel as represented in numpy (see full stamp below).
>>> superpixel = np.array(['G2', 'B', 'R', 'G1']).reshape(2, 2)
>>> superpixel
array([['G2', 'B'],
       ['R', 'G1']], dtype='<U2')
>>> # Tile it into a 5x5 grid of super-pixels, i.e. a 10x10 stamp.
>>> stamp0 = np.tile(superpixel, (5, 5))
>>> stamp0
array([['G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1'],
       ['G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1'],
       ['G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1'],
       ['G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1'],
```

(continues on next page)

(continued from previous page)

```
       ['G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1', 'R', 'G1']],
      dtype='<U2')
>>> stamp1 = np.arange(100).reshape(10, 10)
>>> stamp1
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
>>> x = 7
>>> y = 5
>>> pixel_index = (y, x)  # y=rows, x=columns
>>> stamp0[pixel_index]
'G1'
>>> stamp1[pixel_index]
57
>>> slice0 = bayer.get_stamp_slice(x, y, stamp_size=(6, 6))
>>> slice0
(slice(2, 8, None), slice(4, 10, None))
>>> stamp0[slice0]
array([['G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1'],
       ['G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1'],
       ['G2', 'B', 'G2', 'B', 'G2', 'B'],
       ['R', 'G1', 'R', 'G1', 'R', 'G1']], dtype='<U2')
>>> stamp1[slice0]
array([[24, 25, 26, 27, 28, 29],
       [34, 35, 36, 37, 38, 39],
       [44, 45, 46, 47, 48, 49],
       [54, 55, 56, 57, 58, 59],
       [64, 65, 66, 67, 68, 69],
       [74, 75, 76, 77, 78, 79]])
```

The original index had a value of *57*, which is within the center superpixel.

Notice that the resulting stamp has a super-pixel in the center and is bordered on all sides by a complete super-pixel. This is required by default and an invalid size

We can use *ignore_superpixel=True* to get an odd-sized stamp.

```
>>> slice1 = bayer.get_stamp_slice(x, y, stamp_size=(5, 5), ignore_
↪superpixel=True)
>>> slice1
(slice(3, 8, None), slice(5, 10, None))
>>> stamp0[slice1]
array([['G1', 'R', 'G1', 'R', 'G1'],
       ['B', 'G2', 'B', 'G2', 'B'],
       ['G1', 'R', 'G1', 'R', 'G1'],
       ['B', 'G2', 'B', 'G2', 'B'],
       ['G1', 'R', 'G1', 'R', 'G1']], dtype='<U2')
```

(continues on next page)

(continued from previous page)

```
>>> stamp1[slice1]
array([[35, 36, 37, 38, 39],
       [45, 46, 47, 48, 49],
       [55, 56, 57, 58, 59],
       [65, 66, 67, 68, 69],
       [75, 76, 77, 78, 79]])
```

This puts the requested pixel in the center but does not offer any guarantees about the RGGB pattern.

Parameters

- **x** (*float*) – X pixel position.
- **y** (*float*) – Y pixel position.
- **stamp_size** (*tuple, optional*) – The size of the cutout, default (14, 14).
- **ignore_superpixel** (*bool*) – If superpixels should be ignored, default False.

Returns A slice object for the data.

Return type *slice*

panoptes.utils.images.cr2 module

panoptes.utils.images.cr2.**cr2_to_fits**(*cr2_fname*, *fits_fname=None*, *overwrite=False*, *headers={}*, *fits_headers={}*, *remove_cr2=False*, ***kwargs*)

Convert a CR2 file to FITS

This is a convenience function that first converts the CR2 to PGM via ~cr2_to_pgm. Also adds keyword headers to the FITS file.

Note: The intermediate PGM file is automatically removed

Parameters

- **cr2_fname** (*str*) – Name of the CR2 file to be converted.
- **fits_fname** (*str, optional*) – Name of the FITS file to output. Default is *None*, in which case the *cr2_fname* is used as the base.
- **overwrite** (*bool, optional*) – Overwrite existing FITS, default False.
- **headers** (*dict, optional*) – Header data added to the FITS file.
- **fits_headers** (*dict, optional*) – Header data added to the FITS file without filtering.
- **remove_cr2** (*bool, optional*) – If CR2 should be removed after processing, default False.
- ****kwargs** – Description

Returns The full path to the generated FITS file.

Return type str

panoptes.utils.images.cr2.**cr2_to_pgm**(*cr2_fname*, *pgm_fname=None*, *overwrite=True*, **args*, ***kwargs*)

Convert CR2 file to PGM

Converts a raw Canon CR2 file to a netpbm PGM file via *dcraw*. Assumes *dcraw* is installed on the system

Note: This is a blocking call

Parameters

- **{str} -- Name of CR2 file to convert** (*cr2_fname*) –
- **{dict} -- Additional keywords to pass to script** (***kwargs*) –

Keyword Arguments

- **{str} -- Name of PGM file to output, if None** (*pgm_fname*) – use same name as CR2 (default: {None})
- **{str} -- Path to installed dcraw (default** (*dcraw*) – {'dcraw'})
- **{bool} -- A bool indicating if existing PGM should be overwritten** (*overwrite*) – (default: {True})

Returns str – Filename of PGM that was created

panoptes.utils.images.cr2.**read_exif**(*fname*, *exiftool='exiftool'*)

Read the EXIF information

Gets the EXIF information using exiftool

Note: Assumes the *exiftool* is installed

Parameters **{str} -- Name of file** (*fname*) –

Keyword Arguments **{str} -- Location of exiftool (default** (*exiftool*) – {'/usr/bin/exiftool'})

Returns dict – Dictonary of EXIF information

panoptes.utils.images.cr2.**read_pgm**(*fname*, *byteorder='>'*, *remove_after=False*)

Return image data from a raw PGM file as numpy array.

Note: Format Spec: http://netpbm.sourceforge.net/doc/pgm.html Source: http://stackoverflow.com/questions/7368739/numpy-and-16-bit-pgm

Note: This is correctly processed as a Big endian even though the CR2 itself marks it as a Little endian. See the notes in Source page above as well as the comment about significant bit in the Format Spec

Parameters

- **fname** (*str*) – Filename of PGM to be converted
- **byteorder** (*str*) – Big endian

- **remove_after** (*bool*) – Delete fname file after reading, defaults to False.
- **overwrite** (*bool*) – overwrite existing PGM or not, defaults to True

Returns The raw data from the PGMx

Return type numpy.array

panoptes.utils.images.fits module

panoptes.utils.images.fits.**fpack**(*fits_fname*, *unpack=False*, *overwrite=True*)

Compress/Decompress a FITS file

Uses *fpack* (or *funpack* if *unpack=True*) to compress a FITS file

Parameters

- **fits_fname** (*{str}*) – Name of a FITS file that contains a WCS.
- **unpack** (*{bool}, optional*) – file should decompressed instead of compressed, default False.

Returns Filename of compressed/decompressed file.

Return type str

panoptes.utils.images.fits.**funpack**(**args*, ***kwargs*)

Unpack a FITS file.

Note: This is a thin-wrapper around the ~fpack function with the *unpack=True* option specified. See ~fpack documentation for details.

Parameters

- ***args** – Arguments passed to ~fpack.
- ****kwargs** – Keyword arguments passed to ~fpack.

Returns Path to uncompressed FITS file.

Return type str

panoptes.utils.images.fits.**get_solve_field**(*fname*, *replace=True*, *overwrite=True*, *timeout=30*, ***kwargs*)

Convenience function to wait for *solve_field* to finish.

This function merely passes the *fname* of the image to be solved along to *solve_field*, which returns a subprocess.Popen object. This function then waits for that command to complete, populates a dictonary with the EXIF informaiton and returns. This is often more useful than the raw *solve_field* function.

Example:

```
>>> from panoptes.utils.images import fits as fits_utils
```

```
>>> # Get our fits filename.
>>> fits_fn = getfixture('unsolved_fits_file')
```

```
>>> # Perform the solve.
>>> solve_info = fits_utils.get_solve_field(fits_fn)
```

```
>>> # Show solved filename.
>>> solve_info['solved_fits_file']
'.../unsolved.fits'
```

```
>>> # Pass a suggested location.
>>> ra = 15.23
>>> dec = 90
>>> radius = 5 # deg
>>> solve_info = fits_utils.solve_field(fits_fn, ra=ra, dec=dec, radius=radius)
```

```
>>> # Pass kwargs to `solve-field` program.
>>> solve_kwargs = {'--pnm': '/tmp/awesome.bmp', '--overwrite': True}
>>> solve_info = fits_utils.get_solve_field(fits_fn, **solve_kwargs, skip_
↪solved=False)
>>> assert os.path.exists('/tmp/awesome.bmp')
```

Parameters

- **fname** (*{str}*) – Name of FITS file to be solved.
- **replace** (*bool, optional*) – Saves the WCS back to the original file, otherwise output base filename with *.new* extension. Default True.
- **overwrite** (*bool, optional*) – Clobber file, default True. Required if *replace=True*.
- **timeout** (*int, optional*) – The timeout for solving, default 30 seconds.
- ****kwargs** (*{dict}*) – Options to pass to *solve_field* should start with –.

Returns Keyword information from the solved field.

Return type dict

panoptes.utils.images.fits.**get_wcsinfo**(*fits_fname*, ***kwargs*)

Returns the WCS information for a FITS file.

Uses the *wcsinfo* astrometry.net utility script to get the WCS information from a plate-solved file.

Parameters

- **fits_fname** (*{str}*) – Name of a FITS file that contains a WCS.
- ****kwargs** – Args that can be passed to wcsinfo.

Returns Output as returned from *wcsinfo*.

Return type dict

Raises error.InvalidCommand – Raised if *wcsinfo* is not found (part of astrometry.net)

panoptes.utils.images.fits.**getdata**(*fn*, **args*, ***kwargs*)

Get the FITS data.

Small wrapper around *astropy.io.fits.getdata* to auto-determine the FITS extension. This will return the data associated with the image.

```
>>> fits_fn = getfixture('solved_fits_file')
>>> d0 = getdata(fits_fn)
>>> d0
array([[2215, 2169, 2200, ..., 2169, 2235, 2168],
       [2123, 2191, 2133, ..., 2212, 2127, 2217],
```

(continues on next page)

(continued from previous page)

```
       [2208, 2154, 2209, ..., 2159, 2233, 2187],
       ...,
       [2120, 2201, 2120, ..., 2209, 2126, 2195],
       [2219, 2151, 2199, ..., 2173, 2214, 2166],
       [2114, 2194, 2122, ..., 2202, 2125, 2204]], dtype=uint16)
>>> d1, h1 = getdata(fits_fn, header=True)
>>> (d0 == d1).all()
True
>>> h1['FIELD']
'KIC 8462852'
```

Parameters

- **fn** (*str*) – Path to FITS file.
- ***args** – Passed to *astropy.io.fits.getdata*.
- ****kwargs** – Passed to *astropy.io.fits.getdata*.

Returns The FITS data.

Return type *np.ndarray*

panoptes.utils.images.fits.**getheader**(*fn*, **args*, ***kwargs*)

Get the FITS header.

Small wrapper around *astropy.io.fits.getheader* to auto-determine the FITS extension. This will return the header associated with the image. If you need the compression header information use the astropy module directly.

```
>>> fits_fn = getfixture('tiny_fits_file')
>>> os.path.basename(fits_fn)
'tiny.fits'
>>> header = getheader(fits_fn)
>>> header['IMAGEID']
'PAN001_XXXXXX_20160909T081152'
```

```
>>> # Works with fpacked files
>>> fits_fn = getfixture('solved_fits_file')
>>> os.path.basename(fits_fn)
'solved.fits.fz'
>>> header = getheader(fits_fn)
>>> header['IMAGEID']
'PAN001_XXXXXX_20160909T081152'
```

Parameters

- **fn** (*str*) – Path to FITS file.
- ***args** – Passed to *astropy.io.fits.getheader*.
- ****kwargs** – Passed to *astropy.io.fits.getheader*.

Returns The FITS header for the data.

Return type *astropy.io.fits.header.Header*

panoptes.utils.images.fits.**getval**(*fn*, **args*, ***kwargs*)

Get a value from the FITS header.

Small wrapper around *astropy.io.fits.getval* to auto-determine the FITS extension. This will return the value from the header associated with the image (not the compression header). If you need the compression header information use the astropy module directly.

```
>>> fits_fn = getfixture('tiny_fits_file')
>>> getval(fits_fn, 'IMAGEID')
'PAN001_XXXXXX_20160909T081152'
```

Parameters **fn** (*str*) – Path to FITS file.

Returns Value from header (with no type conversion).

Return type str or float

panoptes.utils.images.fits.**getwcs**(*fn*, **args*, ***kwargs*)

Get the WCS for the FITS file.

Small wrapper around *astropy.wcs.WCS*.

```
>>> from panoptes.utils.images import fits as fits_utils
>>> fits_fn = getfixture('solved_fits_file')
>>> wcs = fits_utils.getwcs(fits_fn)
>>> wcs.is_celestial
True
>>> fits_fn = getfixture('unsolved_fits_file')
>>> wcs = fits_utils.getwcs(fits_fn)
>>> wcs.is_celestial
False
```

Parameters

- **fn** (*str*) – Path to FITS file.
- ***args** – Passed to *astropy.io.fits.getheader*.
- ****kwargs** – Passed to *astropy.io.fits.getheader*.

Returns The World Coordinate System information.

Return type *astropy.wcs.WCS*

panoptes.utils.images.fits.**solve_field**(*fname*, *timeout=15*, *solve_opts=None*, **args*, ***kwargs*)

Plate solves an image.

Note: This is a low-level wrapper around the underlying *solve-field* program. See *get_solve_field* for more typical usage and examples.

Parameters

- **fname** (*str, required*) – Filename to solve in .fits extension.
- **timeout** (*int, optional*) – Timeout for the solve-field command, defaults to 60 seconds.
- **solve_opts** (*list, optional*) – List of options for solve-field.

panoptes.utils.images.fits.**update_observation_headers**(*file_path*, *info*)

Update FITS headers with items from the Observation status.

```
>>> # Check the headers
>>> from panoptes.utils.images import fits as fits_utils
>>> fits_fn = getfixture('unsolved_fits_file')
>>> # Show original value
>>> fits_utils.getval(fits_fn, 'FIELD')
'KIC 8462852'
```

```
>>> info = {'field_name': 'Tabbys Star'}
>>> update_observation_headers(fits_fn, info)
>>> # Show new value
>>> fits_utils.getval(fits_fn, 'FIELD')
'Tabbys Star'
```

Parameters

- **file_path** (*str*) – Path to a FITS file.
- **info** (*dict*) – The return dict from *pocs.observatory.Observation.status*, which includes basic information about the observation.

panoptes.utils.images.fits.**write_fits**(*data*, *header*, *filename*, *exposure_event=None*, ***kwargs*)

Write FITS file to requested location.

```
>>> from panoptes.utils.images import fits as fits_utils
>>> data = np.random.normal(size=100)
>>> header = { 'FILE': 'delete_me', 'TEST': True }
>>> filename = str(getfixture('tmpdir').join('temp.fits'))
>>> fits_utils.write_fits(data, header, filename)
>>> assert os.path.exists(filename)
```

```
>>> fits_utils.getval(filename, 'FILE')
'delete_me'
>>> data2 = fits_utils.getdata(filename)
>>> assert np.array_equal(data, data2)
```

Parameters

- **data** (*array_like*) – The data to be written.
- **header** (*dict*) – Dictionary of items to be saved in header.
- **filename** (*str*) – Path to filename for output.
- **exposure_event** (*None|`threading.Event`, optional*) – A *threading.Event* that can be triggered when the image is written.
- **kwargs** (*dict*) – Options that are passed to the *astropy.io.fits.PrimaryHDU.writeto* method.

panoptes.utils.images.focus module

panoptes.utils.images.focus.**focus_metric**(*data*, *merit_function='vollath_F4'*, ***kwargs*)

Compute the focus metric.

Computes a focus metric on the given data using a supplied merit function. The merit function can be passed either as the name of the function (must be defined in this module) or as a callable object. Additional keyword arguments for the merit function can be passed as keyword arguments to this function.

Parameters

- **data** (*numpy array*) –
- **merit_function** (*str/callable*) – panoptes.utils.images) or a callable object.

Returns result of calling merit function on data

Return type scalar

panoptes.utils.images.focus.**vollath_F4**(*data*, *axis=None*)

Compute F4 focus metric

Computes the F_4 focus metric as defined by Vollath (1998) for the given 2D numpy array. The metric can be computed in the y axis, x axis, or the mean of the two (default).

Parameters

- **data** (*numpy array*) –
- **axis** (*str, optional, default None*) – be 'Y'/'y', 'X'/'x' or None, which will calculate the F4 value for both axes and return the mean.

Returns Calculated F4 value for y, x axis or both

Return type float64

panoptes.utils.images.plot module

panoptes.utils.images.plot.**add_colorbar**(*axes_image*, *size='5%'*, *pad=0.05*, *orientation='vertical'*)

Add a colorbar to the image.

This is a simple convenience function to add a colorbar to a plot generated by *matplotlib.pyplot.imshow*.

Parameters **axes_image** (*matplotlib.image.AxesImage*) – A matplotlib AxesImage.

panoptes.utils.images.plot.**add_pixel_grid**(*ax1*, *grid_height*, *grid_width*, *show_axis_labels=True*, *show_superpixel=False*, *major_alpha=0.5*, *minor_alpha=0.25*)

panoptes.utils.images.plot.**animate_stamp**(*d0*)

panoptes.utils.images.plot.**get_palette**(*cmap='inferno'*)

Get a palette for drawing.

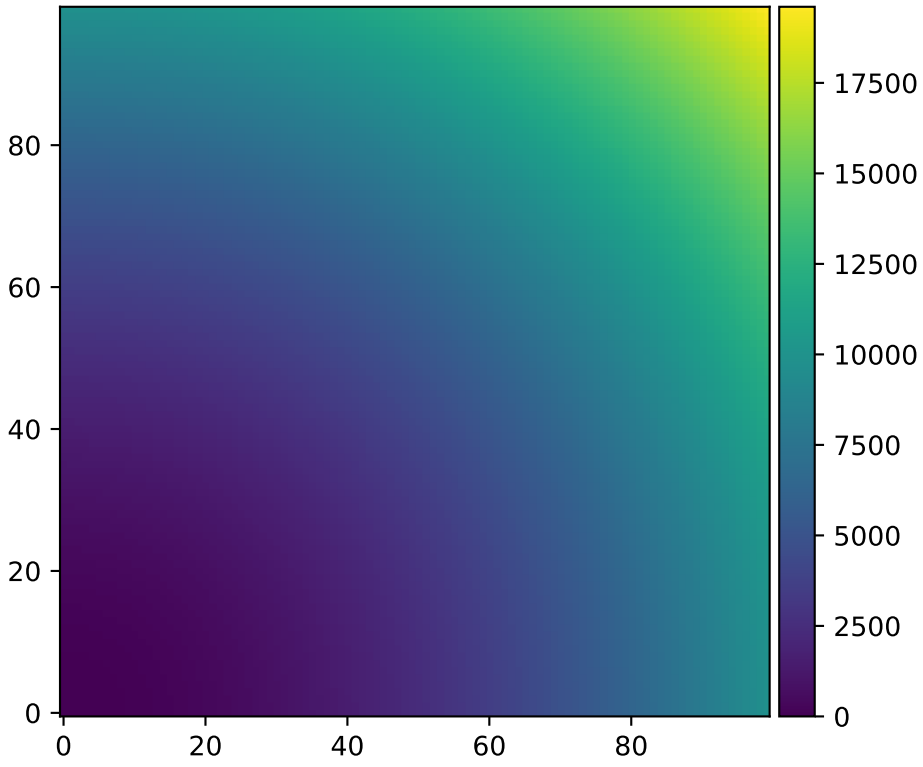
Returns a copy of the colormap palette with bad pixels marked.

Parameters **cmap** (*str, optional*) – Colormap to use, default 'inferno'.

Returns The colormap.

Return type *matplotlib.cm*

panoptes.utils.images.plot.**show_stamps**(*pscs*, *frame_idx=None*, *stamp_size=11*, *aperture_position=None*, *show_residual=False*, *stretch=None*, *save_name=None*, *show_max=False*, *show_pixel_grid=False*, ***kwargs*)


0
20
40
60
80
0
20
40
60
80
0
2500
5000
7500
10000
12500
15000
17500

Module contents

panoptes.utils.images.**crop_data**(*data*, *box_width=200*, *center=None*, *data_only=True*, *wcs=None*, ***kwargs*)

Return a cropped portion of the image

Shape is a box centered around the middle of the data

Parameters

- **data** (*numpy.array*) – Array of data.
- **box_width** (*int, optional*) – Size of box width in pixels, defaults to 200px.
- **center** (*tuple(int, int), optional*) – Crop around set of coords, default to image center.
- **data_only** (*bool, optional*) – If True (default), return only data. If False return the *Cutout2D* object.
- **wcs** (*None|`astropy.wcs.WCS`, optional*) – A valid World Coordinate System (WCS) that will be cropped along with the data if provided.

Returns

A clipped (thumbnailed) version of the data if *data_only=True*, otherwise a *astropy.nddata.Cutout2D* object.

Return type np.array

panoptes.utils.images.**make_pretty_image**(*fname*, *title=None*, *timeout=15*, *img_type=None*, *link_path=None*, ***kwargs*)

Make a pretty image.

This will create a jpg file from either a CR2 (Canon) or FITS file.

Notes

See `scripts/cr2_to_jpg.sh` for CR2 process.

Parameters

- **fname** (*str*) – The path to the raw image.
- **title** (*None|str, optional*) – Title to be placed on image, default None.
- **timeout** (*int, optional*) – Timeout for conversion, default 15 seconds.
- **img_type** (*None|str, optional*) – Image type of fname, one of '.cr2' or '.fits'. The default is *None*, in which case the file extension of fname is used.
- **link_path** (*None|str, optional*) – Path to location that image should be sym-linked. The directory must exist.
- **{dict} -- Additional arguments to be passed to external script.** (***kwargs*) –

Returns str – Filename of image that was created.

Deleted Parameters:

link_latest (bool, optional): If the pretty picture should be linked to `link_path`, default False.

panoptes.utils.images.**make_timelapse**(*directory*, *fn_out=None*, *glob_pattern='20[1-9][0-9]*T[0-9]*.jpg'*, *overwrite=False*, *timeout=60*, ***kwargs*)

Create a timelapse.

A timelapse is created from all the images in given `directory`

Parameters

- **directory** (*str*) – Directory containing image files.
- **fn_out** (*str, optional*) – Full path to output file name, if not provided, defaults to *directory* basename.
- **glob_pattern** (*str, optional*) – A glob file pattern of images to include, default '20[1-9][0-9]*T[0-9]*.jpg', which corresponds to the observation images but excludes any pointing images. The pattern should be relative to the local directory.
- **overwrite** (*bool, optional*) – Overwrite timelapse if exists, default False.
- **timeout** (*int*) – Timeout for making movie, default 60 seconds.
- ****kwargs** (*dict*) –

Returns Name of output file

Return type str

Raises

- `error.InvalidSystemCommand` – Raised if ffmpeg command is not found.
- `FileExistsError` – Raised if fn_out already exists and overwrite=False.

panoptes.utils.images.**mask_saturated**(*data*, *saturation_level=None*, *threshold=0.9*, *bit_depth=None*, *dtype=None*)

Convert data to a masked array with saturated values masked.

Parameters

- **data** (*array_like*) – The numpy data array.
- **saturation_level** (*scalar, optional*) – The saturation level. If not given then the saturation level will be set to threshold times the maximum pixel value.
- **threshold** (*float, optional*) – The fraction of the maximum pixel value to use as the saturation level, default 0.9.
- **bit_depth** (*astropy.units.Quantity or int, optional*) – The effective bit depth of the data. If given the maximum pixel value will be assumed to be 2**bit_depth, otherwise an attempt will be made to infer the maximum pixel value from the data type of the data. If data is not an integer type the maximum pixel value cannot be inferred and an IllegalValue exception will be raised.
- **dtype** (*numpy.dtype, optional*) – The requested dtype for the masked array. If not given the dtype of the masked array will be same as data.

Returns The masked numpy array.

Return type numpy.ma.array

Raises

- `error.IllegalValue` – Raised if bit_depth is an astropy.units.Quantity object but the units are not compatible with either bits or bits/pixel.

- `error.IllegalValue` – Raised if neither saturation level or bit_depth are given, and data has a non integer data type.

panoptes.utils.serial_handlers package

Submodules

panoptes.utils.serial_handlers.protocol_arduinosimulator module

Provides a simple simulator for telemetry_board.ino or camera_board.ino.

We use the pragma "no cover" in several places that happen to never be reached or that would only be reached if the code was called directly, i.e. not in the way it is intended to be used.

class panoptes.utils.serial_handlers.protocol_arduinosimulator.**ArduinoSimulator**(*message*, *relay_queue*, *json_queue*, *chunk_size*, *stop*)

Bases: `object`

Simulates the serial behavior of the PANOPTES Arduino sketches.

The RS-232 connection is simulated with an input and output queue of bytes. This class provides a run function which can be called from a Thread to execute. Every two seconds while running it will generate another json output line, and then send that to the json_queue in small chunks at a rate similar to 9600 baud, the rate used by our Arduino sketches.

__init__(*message*, *relay_queue*, *json_queue*, *chunk_size*, *stop*)

Parameters

- **message** – The message to be sent (millis and report_num will be added).
- **relay_queue** – The queue.Queue instance from which relay command bytes are read and acted upon. Elements are of type bytes.
- **json_queue** – The queue.Queue instance to which json messages (serialized to bytes) are written at ~9600 baud. Elements are of type bytes (i.e. each element is a sequence of bytes of length up to chunk_size).
- **chunk_size** – The number of bytes to write to json_queue at a time.
- **stop** – a threading.Event which is checked to see if run should stop executing.

generate_next_message(*now*)
Append the next message to the pending bytearray and scheduled the next message.

generate_next_message_bytes(*now*)
Generate the next message (report) from the simulated Arduino.

handle_pending_relay_bytes()
Process complete relay commands.

output_next_chunk(*now*)
Output one chunk of pending json bytes.

read_relay_queue_until(*next_time*)
Read and process relay queue bytes until time for the next action.

run()
Produce messages periodically and emit their bytes at a limited rate.

class panoptes.utils.serial_handlers.protocol_arduinosimulator.**FakeArduinoSerialHandler**(*ar
**k

Bases: *panoptes.utils.serial_handlers.NoOpSerial*

close()
Close port immediately.

flush()
Write the buffered data to the output device.

We interpret that here as waiting until the simulator has taken all of the entries from the queue.

in_waiting
The number of input bytes available to read immediately.

is_config_ok
Does the caller ask for the correct serial device config?

open()
Open port.

Raises SerialException if the port cannot be opened.

out_waiting
The number of bytes in the output buffer.

read(*size=1*)
Read size bytes.

If a timeout is set it may return fewer characters than requested. With no timeout it will block until the requested number of bytes is read.

Parameters **size** – Number of bytes to read.

Returns Bytes read from the port, of type 'bytes'.

readline()
Read and return one line from the simulator.

This override exists just to support logging of the line.

reset_input_buffer()
Flush input buffer, discarding all it's contents.

reset_output_buffer()
Clear output buffer.

Aborts the current output, discarding all that is in the output buffer.

write(*data*)
Write the bytes data to the port.

Parameters **data** – The data to write (bytes or bytearray instance).

Returns Number of bytes written.

Raises SerialTimeoutException – In case a write timeout is configured for the port and the time is exceeded.

panoptes.utils.serial_handlers.protocol_arduinosimulator.**Serial**
alias of *panoptes.utils.serial_handlers.protocol_arduinosimulator.FakeArduinoSerialHandler*

panoptes.utils.serial_handlers.protocol_buffers module

class panoptes.utils.serial_handlers.protocol_buffers.**BuffersSerial**(**args*, ***kwargs*)

Bases: *panoptes.utils.serial_handlers.NoOpSerial*

in_waiting
The number of input bytes available to read immediately.

read(*size=1*)
Read size bytes.

If a timeout is set it may return fewer characters than requested. With no timeout it will block until the requested number of bytes is read.

Parameters **size** – Number of bytes to read.

Returns Bytes read from the port, of type 'bytes'.

Raises SerialTimeoutException – In case a write timeout is configured for the port and the time is exceeded.

write(*data*)

Parameters **data** – The data to write.

Returns Number of bytes written.

Raises SerialTimeoutException – In case a write timeout is configured for the port and the time is exceeded.

panoptes.utils.serial_handlers.protocol_buffers.**GetWBufferValue**()
Returns an immutable bytes object with the value of the w buffer.

panoptes.utils.serial_handlers.protocol_buffers.**ResetBuffers**(*read_data=None*)

panoptes.utils.serial_handlers.protocol_buffers.**Serial**
alias of *panoptes.utils.serial_handlers.protocol_buffers.BuffersSerial*

panoptes.utils.serial_handlers.protocol_buffers.**SetRBufferValue**(*data*)
Sets the r buffer to data (a bytes object).

panoptes.utils.serial_handlers.protocol_hooked module

panoptes.utils.serial_handlers.protocol_hooked.**ExampleSerialClassForUrl**(*url*)
Implementation of serial_class_for_url called by serial.serial_for_url.

Returns the url, possibly modified, and a factory function to be called to create an instance of a SerialBase sub-class (or at least behaves like it). You can return a class as that factory function, as calling a class creates an instance of that class.

serial.serial_for_url will call that factory function with None as the port parameter (the first), and after creating the instance will assign the url to the port property of the instance.

Returns A tuple (url, factory).

panoptes.utils.serial_handlers.protocol_hooked.**serial_class_for_url**(*url*)
Implementation of serial_class_for_url called by serial.serial_for_url.

Returns the url, possibly modified, and a factory function to be called to create an instance of a SerialBase sub-class (or at least behaves like it). You can return a class as that factory function, as calling a class creates an instance of that class.

serial.serial_for_url will call that factory function with None as the port parameter (the first), and after creating the instance will assign the url to the port property of the instance.

Returns A tuple (url, factory).

panoptes.utils.serial_handlers.protocol_no_op module

Module contents

The protocol_*.py files in this package are based on PySerial's file test/handlers/protocol_test.py, modified for different behaviors. The call serial.serial_for_url("XYZ://") looks for a class Serial in a file named protocol_XYZ.py in this package (i.e. directory).

class panoptes.utils.serial_handlers.**NoOpSerial**(**args*, ***kwargs*)
Bases: serial.serialutil.SerialBase

No-op implementation of PySerial's SerialBase.

Provides no-op implementation of various methods that SerialBase expects to have implemented by the sub-class. Can be used as is for a /dev/null type of behavior.

close()
Close port immediately.

in_waiting
The number of input bytes available to read immediately.

open()
Open port.

Raises SerialException if the port cannot be opened.

read(*size=1*)
Read size bytes.

If a timeout is set it may return fewer characters than requested. With no timeout it will block until the requested number of bytes is read.

Parameters **size** – Number of bytes to read.

Returns Bytes read from the port, of type 'bytes'.

Raises SerialTimeoutException – In case a write timeout is configured for the port and the time is exceeded.

reset_input_buffer()
Remove any accumulated bytes from the device.

reset_output_buffer()
Remove any accumulated bytes not yet sent to the device.

write(*data*)

Parameters **data** – The data to write.

Returns Number of bytes written.

Raises SerialTimeoutException – In case a write timeout is configured for the port and the time is exceeded.

panoptes.utils.social package

Submodules

panoptes.utils.social.slack module

class panoptes.utils.social.slack.**SocialSlack**(***kwargs*)
Bases: object

Social Messaging sink to output to Slack.

send_message(*msg*, *timestamp*)

panoptes.utils.social.twitter module

Module contents

Submodules

panoptes.utils.error module

exception panoptes.utils.error.**ArduinoDataError**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

PanError raised when there is something very wrong with Arduino information.

exception panoptes.utils.error.**BadConnection**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

PanError raised when a connection is bad

exception panoptes.utils.error.**BadSerialConnection**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

PanError raised when serial command is bad

exception panoptes.utils.error.**CameraNotFound**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.NotFound*

Camera cannot be imported

exception panoptes.utils.error.**DomeNotFound**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.NotFound*

Dome device not found.

exception panoptes.utils.error.**GoogleCloudError**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

Errors related to google cloud

exception panoptes.utils.error.**IllegalValue**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*, ValueError

Errors from trying to hardware parameters to values not supported by a particular model

exception panoptes.utils.error.**InvalidCommand**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

PanError raised if a system command does not run

exception panoptes.utils.error.**InvalidConfig**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

PanError raised if config file is invalid

exception panoptes.utils.error.**InvalidDeserialization**(*msg='Problem deserializing'*, ***kwargs*)
Bases: *panoptes.utils.error.PanError*

Error for serialization errors

exception panoptes.utils.error.**InvalidMountCommand**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

PanError raised if attempting to send command that doesn't exist

exception panoptes.utils.error.**InvalidObservation**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.NotFound*

PanError raised if a field is invalid.

exception panoptes.utils.error.**InvalidSerialization**(*msg='Problem Serializing'*, ***kwargs*)
Bases: *panoptes.utils.error.PanError*

Error for serialization errors

exception panoptes.utils.error.**InvalidSystemCommand**(*msg='Problem running system command'*, ***kwargs*)
Bases: *panoptes.utils.error.PanError*

Error for a system level command malfunction

exception panoptes.utils.error.**MountNotFound**(*msg='Mount Not Found'*, ***kwargs*)
Bases: *panoptes.utils.error.NotFound*

Mount cannot be import

exception panoptes.utils.error.**NoObservation**(*msg='No valid observations found.'*, ***kwargs*)
Bases: *panoptes.utils.error.PanError*

Generic no Observation

exception panoptes.utils.error.**NotFound**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

Generic not found class

exception panoptes.utils.error.**NotSupported**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*, NotImplementedError

Errors from trying to use hardware features not supported by a particular model

exception panoptes.utils.error.**PanError**(*msg=None*, *exit=False*)
Bases: Exception

Base class for Panoptes errors

exit_program(*msg=None*)
Kills running program

exception panoptes.utils.error.**SolveError**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.NotFound*

Camera cannot be imported

exception panoptes.utils.error.**TheSkyXError**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.PanError*

Errors from TheSkyX

exception panoptes.utils.error.**TheSkyXKeyError**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.TheSkyXError*

Errors from TheSkyX because bad key passed

exception panoptes.utils.error.**TheSkyXTimeout**(*msg=None*, *exit=False*)
Bases: *panoptes.utils.error.TheSkyXError*

Errors from TheSkyX because bad key passed

exception panoptes.utils.error.**Timeout**(*msg='Timeout waiting for event'*, ***kwargs*)
Bases: *panoptes.utils.error.PanError*

Error called when an event times out

panoptes.utils.horizon module

class panoptes.utils.horizon.**Horizon**(*obstructions=[]*, *default_horizon=30*)
Bases: object

A simple class to define some coordinate points.

Accepts a list of lists where each list consists of two points corresponding to an altitude (0-90) and an azimuth (0-360). If azimuth is a negative number (but greater than -360) then 360 will be added to put it in the correct range.

The list are points that are obstruction points beyond the default horizon.

__init__(*obstructions=[]*, *default_horizon=30*)
Create a list of horizon obstruction points.

Each item in the *obstructions* list should be two or more points, where each point is an *[Alt, Az]* coordinate.

Example

An example *obstruction_point* list:

```
[
[[40, 30], [40, 75]],    # From azimuth 30° to 75° there is an
                         # obstruction that is at 40° altitude
[[50, 180], [40, 200]], # From azimuth 180° to 200° there is
                         # an obstruction that slopes from 50°
                         # to 40° altitude
]
```

Parameters

- **obstructions** (*list(list(list)), optional*) – A list of obstructions where each obstruction consists of a set of lists. The individual lists are alt/az pairs. Defaults to empty list in which case the *default_horizon* defines a flat horizon.
- **default_horizon** (*float, optional*) – A default horizon to be used whenever there is no obstruction.

panoptes.utils.library module

panoptes.utils.library.**load_c_library**(*name*, *path=None*, *mode=0*, ***kwargs*)
Utility function to load a shared/dynamically linked library (.so/.dylib/.dll).

The name and location of the shared library can be manually specified with the library_path argument, otherwise the ctypes.util.find_library function will be used to try to locate based on library_name.

Parameters

- **name** (*str*) – name of the library (without 'lib' prefix or any suffixes, e.g. 'fli').
- **path** (*str, optional*) – path to the library e.g. '/usr/local/lib/libfli.so'.
- **mode** (*int, optional*) – mode in which to load the library, see dlopen(3) man page for details. Should be one of ctypes.RTLD_GLOBAL, ctypes.RTLD_LOCAL, or ctypes.DEFAULT_MODE. Default is ctypes.DEFAULT_MODE.

Returns ctypes.CDLL

Raises

- pocs.utils.error.NotFound – raised if library_path not given & find_library fails to locate the library.
- OSError – raises if the ctypes.CDLL loader cannot load the library.

panoptes.utils.library.**load_module**(*module_name*)
Dynamically load a module.

```
>>> from panoptes.utils.library import load_module
>>> error = load_module('panoptes.utils.error')
>>> error.__name__
'panoptes.utils.error'
>>> error.__package__
'panoptes.utils'
```

Parameters **module_name** (*str*) – Name of module to import.

Returns an imported module name

Return type module

Raises error.NotFound – If module cannot be imported.

panoptes.utils.logging module

panoptes.utils.rs232 module

Provides SerialData, a PySerial wrapper.

class panoptes.utils.rs232.**SerialData**(*port=None*, *baudrate=115200*, *name=None*, *timeout=2.0*, *open_delay=0.0*, *retry_limit=5*, *retry_delay=0.5*, ***kwargs*)
Bases: object

SerialData wraps a PySerial instance for reading from and writing to a serial device.

Because POCS is intended to be very long running, and hardware may be turned off when unused or to force a reset, this wrapper may or may not have an open connection to the underlying serial device. Note that for

most devices, is_connected will return true if the device is turned off/unplugged after a connection is opened; the code will only discover there is a problem when we attempt to interact with the device.

```
>>> import serial

# Register our serial simulators
>>> serial.protocol_handler_packages.append('panoptes.utils.serial_handlers')
>>> from panoptes.utils.serial_handlers import protocol_buffers as pb

# Import our serial utils
>>> from panoptes.utils.rs232 import SerialData

# Connect to our fake buffered device
>>> device_listener = SerialData(port='buffers://')

# Note: A manual reset is currently required because implementation is not␣
↪complete.
# See https://github.com/panoptes/POCS/issues/758 for details.
>>> pb.ResetBuffers()
>>> device_listener.is_connected
True

>>> device_listener.port
'buffers://'

# Device sends event
>>> pb.SetRBufferValue(b'emit event')

# Listen for event
>>> device_listener.read()
'emit event'

>>> device_listener.write('ack event')
9
>>> pb.GetWBufferValue()
b'ack event'

# Remove custom handlers
>>> serial.protocol_handler_packages.remove('panoptes.utils.serial_handlers')
```

__init__(*port=None*, *baudrate=115200*, *name=None*, *timeout=2.0*, *open_delay=0.0*, *retry_limit=5*, *retry_delay=0.5*, ***kwargs*)

Create a SerialData instance and attempt to open a connection.

The device need not exist at the time this is called, in which case is_connected will be false.

Parameters

- **port** – The port (e.g. /dev/tty123 or socket://host:port) to which to open a connection.
- **baudrate** – For true serial lines (e.g. RS-232), sets the baud rate of the device.
- **name** – Name of this object. Defaults to the name of the port.
- **timeout** (*float, optional*) – Timeout in seconds for both read and write. Defaults to 2.0.
- **open_delay** – Seconds to wait after opening the port.
- **retry_limit** – Number of times to try readline() calls in read().
- **retry_delay** – Delay between readline() calls in read().

Raises ValueError – If the serial parameters are invalid (e.g. a negative baudrate).

connect()
If disconnected, then connect to the serial port.

Raises error.BadSerialConnection if unable to open the connection.

disconnect()
Closes the serial connection.

Raises error.BadSerialConnection if unable to close the connection.

get_and_parse_reading(*retry_limit=5*)
Reads a line of JSON text and returns the decoded value, along with the current time.

Parameters **retry_limit** – Number of lines to read in an attempt to get one that parses as JSON.

Returns A pair (tuple) of (timestamp, decoded JSON line). The timestamp is the time of completion of the readline operation.

get_reading()
Reads and returns a line, along with the timestamp of the read.

Returns A pair (tuple) of (timestamp, line). The timestamp is the time of completion of the readline operation.

is_connected
True if serial port is open, False otherwise.

port
Name of the port.

read(*retry_limit=None*, *retry_delay=None*)
Reads next line of input using readline.

If no response is given, delay for retry_delay and then try to read again. Fail after retry_limit attempts.

read_bytes(*size=1*)
Reads size bytes from the serial port.

If a read timeout is set on self.ser, this may return less characters than requested. With no timeout it will block until the requested number of bytes is read.

Parameters **size** – Number of bytes to read.

Returns Bytes read from the port.

reset_input_buffer()
Clear buffered data from connected port/device.

Note that Wilfred reports that the input from an Arduino can seriously lag behind realtime (e.g. 10 seconds), and that clear_buffer may exist for that reason (i.e. toss out any buffered input from a device, and then read the next full line, which likely requires tossing out a fragment of a line).

write(*value*)
Write value (a string) after encoding as bytes.

write_bytes(*data*)
Write data of type bytes.

panoptes.utils.rs232.**get_serial_port_info**()
Returns the serial ports defined on the system.

Returns: a list of PySerial's ListPortInfo objects. See: https://github.com/pyserial/pyserial/blob/master/serial/tools/list_ports_common.py

panoptes.utils.serializers module

class panoptes.utils.serializers.**StringYAML**(*_kw=<object object>*, *typ=None*, *pure=False*, *output=None*, *plug_ins=None*)

Bases: `ruamel.yaml.main.YAML`

dump(*data*, *stream=None*, ***kwargs*)

YAML class that can dump to a string.

By default the YAML parser doesn't serialize directly to a string. This class is a small wrapper to output StreamIO as a string if no stream is provided.

See https://yaml.readthedocs.io/en/latest/example.html#output-of-dump-as-a-string.

Note: This class should not be used directly but instead is instantiated as part of the yaml convenience methods below.

Parameters

- **data** (*object*) – An object, usually dict-like.
- **stream** (*None* | stream, optional) – A stream object to write the YAML. If default *None*, return value as string.
- ****kwargs** – Keywords passed to the *dump* function.

Returns The serialized object string.

Return type *str*

panoptes.utils.serializers.**deserialize_all_objects**(*obj*)

Recursively parse the incoming object for various data types.

This will currently attempt to parse and return, in the following order:

If `obj` is a dict with exactly two keys named `unit` and `value`, then attempt to parse into a valid `astropy.unit.Quantity`.

A boolean.

A *datetime.datetime* object as parsed by *dateutil.parser.parse*.

If a string ending with any of `['m', 'deg', 's']`, an `astropy.unit.Quantity`

Note: See the *to/from_json/yaml* methods, which use this function.

Parameters **obj** (*dict* or *str* or *object*) – Object to check for quantities.

Returns Same as *obj* but with objects converted to quantities.

Return type *dict*

panoptes.utils.serializers.**from_json**(*msg*)

Convert a JSON string into a Python object.

Astropy quanitites will be converted from a `{"value": val, "unit": unit}` format. Additionally, the following units will be converted if the value ends with the exact string:

- deg
- m
- s

Time-like values are *not* parsed, however see example below.

Examples:

```
>>> from panoptes.utils.serializers import from_json
>>> config_str = '{"name":"Mauna Loa","elevation":{"value":3397.0,"unit":"m"}}'
>>> from_json(config_str)
{'name': 'Mauna Loa', 'elevation': <Quantity 3397. m>}

# Invalid values will be returned as is.
>>> from_json('{"horizon":{"value":42.0,"unit":"degr"}}')
{'horizon': {'value': 42.0, 'unit': 'degr'}}

# The following will convert if final string:
>>> from_json('{"horizon": "42.0 deg"}')
{'horizon': <Quantity 42. deg>}

>>> from_json('{"elevation": "1000 m"}')
{'elevation': <Quantity 1000. m>}

>>> from_json('{"readout_time": "10 s"}')
{'readout_time': <Quantity 10. s>}

# Be careful with short unit names in extended format!
>>> horizon = from_json('{"horizon":{"value":42.0,"unit":"d"}}')
>>> horizon['horizon']
<Quantity 42. d>
>>> horizon['horizon'].decompose()
<Quantity 3628800. s>

>>> from panoptes.utils import current_time
>>> time_str = to_json({"current_time": current_time().datetime})
>>> from_json(time_str)['current_time']
2019-04-08T06:43:28.232406

>>> from astropy.time import Time
>>> Time(from_json(time_str)['current_time'])
<Time object: scale='utc' format='isot' value=2019-04-08T06:43:28.232>
```

Parameters **msg** (*str*) – The JSON string representation of the object.

Returns The loaded object.

Return type *dict*

panoptes.utils.serializers.**from_yaml**(*msg*, *parse=True*)

Convert a YAML string into a Python object.

This is a thin-wrapper around *ruamel.YAML.load* that also parses the results looking for *astropy.units.Quantity* objects.

Comments are preserved as long as the object remains YAML (lost on conversion to JSON, for example).

See *from_json* for examples of astropy unit parsing.

Examples

Note how comments in the YAML are preserved.

```
>>> config_str = '''name: Testing PANOPTES Unit
... pan_id: PAN000
...
... location:
...   latitude: 19.54 deg
...   longitude: -155.58 deg
...   name: Mauna Loa Observatory  # Can be anything
... '''

>>> config = from_yaml(config_str)
>>> config['location']['latitude']
<Quantity 19.54 deg>

>>> yaml_config = to_yaml(config)
>>> yaml_config
''' name: Testing PANOPTES Unit
... pan_id: PAN000  # CHANGE NAME
...
... location:
...   latitude: 19.54 deg
...   longitude: value: -155.58 deg
...   name: Mauna Loa Observatory  # Can be anything
... '''
>>> yaml_config == config_str
True
```

Parameters

- **msg** (*str*) – The YAML string representation of the object.
- **parse** (*bool*) – If objects should be parsed via *_parse_all_objects*, default True.

Returns

The ordered dict representing the YAML string, with appropriate object deserialization.

Return type *collections.OrderedDict*

panoptes.utils.serializers.**serialize_all_objects**(*obj*)

Iterate the `obj` items and serialize each value.

Note: See the *to/from_json/yaml* methods, which use this function.

Parameters **obj** (*dict*) – The dictionary object to be iterated.

Returns The same as `obj` but with the values serialized.

Return type dict

panoptes.utils.serializers.**serialize_object**(*obj*)

Serialize the given object.

This is a custom serializer function used by `to_json` to serialize individual objects. Also called in a loop by `serialize_all_objects`.

```
>>> from panoptes.utils.serializers import serialize_object
>>> from dateutil.parser import parse as date_parse
>>> from astropy import units as u
```

```
>>> serialize_object(42 * u.meter)
'42.0 m'
```

```
>>> party_time = date_parse('1999-12-31 11:59:59')
>>> type(party_time)
 <class 'datetime.datetime'>
>>> serialize_object(party_time)
'1999-12-31T11:59:59.000'
```

Note: See the *to/from_json/yaml* methods, which use this function.

Parameters **obj** (*any*) – The object to be serialized.

Returns:

panoptes.utils.serializers.**to_json**(*obj*, *filename=None*, *append=True*, ***kwargs*)

Convert a Python object to a JSON string.

Will handle *datetime* objects as well as *astropy.unit.Quantity* objects. Astropy quantities will be converted to a dict: *{"value": val, "unit": unit}*.

Examples:

```
>>> from panoptes.utils.serializers import to_json
>>> from astropy import units as u
>>> config = { "name": "Mauna Loa", "elevation": 3397 * u.meter }
>>> to_json(config)
'{"name": "Mauna Loa", "elevation": "3397.0 m"}'

>>> to_json({"numpy_array": np.arange(10)})
'{"numpy_array": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}'

>>> from panoptes.utils import current_time
>>> to_json({"current_time": current_time()})
'{"current_time": "2019-04-08 22:19:28.402198"}'
```

Parameters

- **obj** (*object*) – The object to be converted to JSON, usually a dict.
- **filename** (*str*, optional) – Path to file for saving.
- **append** (*bool*, optional) – Append to *filename*, default True. Setting False will clobber the file.

- ****kwargs** – Keyword arguments passed to *json.dumps*.

Returns The JSON string representation of the object.

Return type *str*

panoptes.utils.serializers.**to_yaml**(*obj*, ***kwargs*)

Serialize a Python object to a YAML string.

This will properly serialize the following:

- *datetime.datetime*
- *astropy.time.Time*
- *astropy.units.Quantity*

Examples

Also see the examples *from_yaml*.

```
>>> import os
>>> os.environ['POCSTIME'] = '1999-12-31 23:49:49'
>>> from panoptes.utils import current_time
>>> t0 = current_time()
>>> t0
<Time object: scale='utc' format='iso' value=1999-12-31 23:49:49.000>

>>> to_yaml({'astropy time -> astropy time': t0})
"astropy time -> astropy time: '1999-12-31T23:49:49.000'\n"

>>> to_yaml({'datetime -> astropy time': t0.datetime})
"datetime -> astropy time: '1999-12-31T23:49:49.000'\n"

>>> # Can pass a `stream` parameter to save to file
>>> with open('temp.yaml', 'w') as f:
...     to_yaml({'my_object': 42}, stream=f)
```

Parameters

- **obj** (*dict*) – The object to be converted to be serialized.
- ****kwargs** – Arguments passed to *ruamel.yaml.dump*. See Examples.

Returns The YAML string representation of the object.

Return type *str*

panoptes.utils.theskyx module

class panoptes.utils.theskyx.**TheSkyX**(*host='localhost'*, *port=3040*, *connect=True*, **args*, ***kwargs*)

Bases: object

A socket connection for communicating with TheSkyX

connect()

Sets up serial connection

is_connected

read(*timeout=5*)

write(*value*)

panoptes.utils.time module

class panoptes.utils.time.**CountdownTimer**(*duration*)
Bases: object

Simple timer object for tracking whether a time duration has elapsed.

Parameters **duration** (*int or float or astropy.units.Quantity*) – Amount of time to before time expires. May be numeric seconds or an Astropy time duration (e.g. 1 * u.minute).

expired()
Return a boolean, telling if the timeout has expired.

Returns If timer has expired.

Return type bool

restart()
Restart the timed duration.

sleep(*max_sleep=None*)
Sleep until the timer expires, or for max_sleep, whichever is sooner.

Parameters **max_sleep** – Number of seconds to wait for, or None.

Returns True if slept for less than time_left(), False otherwise.

time_left()
Return how many seconds are left until the timeout expires.

Returns Number of seconds remaining in timer, zero if `is_non_blocking=True`.

Return type int

panoptes.utils.time.**current_time**(*flatten=False*, *datetime=False*, *pretty=False*)
Convenience method to return the "current" time according to the system.

Note: If the `$POCSTIME` environment variable is set then this will return the time given in the variable. This is used for setting specific times during testing. After checking the value of POCSTIME the environment variable will also be incremented by one second so that subsequent calls to this function will generate monotonically increasing times.

Operation of POCS from $POCS/bin/pocs_shell will clear the POCSTIME variable.

```
>>> os.environ['POCSTIME'] = '1999-12-31 23:59:59'
>>> party_time = current_time(pretty=True)
>>> party_time
'1999-12-31 23:59:59'

# Next call is one second later when using $POCSTIME.
>>> y2k = current_time(pretty=True)
>>> y2k
'2000-01-01 00:00:00'
```

Note: The time returned from this function is **not** timezone aware. All times are UTC.

```
>>> from panoptes.utils import current_time
>>> current_time()
<Time object: scale='utc' format='datetime' value=2018-10-07 22:29:03.009873>

>>> current_time(datetime=True)
datetime.datetime(2018, 10, 7, 22, 29, 26, 594368)

>>> current_time(pretty=True)
'2018-10-07 22:29:51'
```

Returns Object representing now.

Return type astropy.time.Time

panoptes.utils.time.**flatten_time**(*t*)

Given an astropy time, flatten to have no extra chars besides integers.

```
>>> from astropy.time import Time
>>> from panoptes.utils import flatten_time
>>> t0 = Time('1999-12-31 23:59:59')
>>> t0.isot
'1999-12-31T23:59:59.000'

>>> flatten_time(t0)
'19991231T235959'
```

Parameters **t** (*astropy.time.Time*) – The time to be flattened.

Returns The flattened string representation of the time.

Return type str

panoptes.utils.time.**wait_for_events**(*events*, *timeout=600*, *sleep_delay=<Quantity 5. s>*, *callback=None*)

Wait for event(s) to be set.

This method will wait for a maximum of *timeout* seconds for all of the *events* to complete.

Checks every *sleep_delay* seconds for the events to be set.

If provided, the *callback* will be called every *sleep_delay* seconds. The callback should return *True* to continue waiting otherwise *False* to interrupt the loop and return from the function.

```
>>> import time
>>> import threading
>>> from panoptes.utils.time import wait_for_events
>>> # Create some events, normally something like taking an image.
>>> event0 = threading.Event()
>>> event1 = threading.Event()

>>> # Wait for 30 seconds but interrupt after 1 second by returning False from␣
↪callback.
>>> def interrupt_cb(): time.sleep(1); return False
```

(continues on next page)

(continued from previous page)

```
>>> # The function will return False if events are not set.
>>> wait_for_events([event0, event1], timeout=30, callback=interrupt_cb)
False

>>> # Timeout will raise an exception.
>>> wait_for_events([event0, event1], timeout=1)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File ".../panoptes-utils/src/panoptes/utils/time.py", line 254, in wait_for_
↪events
panoptes.utils.error.Timeout: Timeout: Timeout waiting for generic event

>>> # Set the events in another thread for normal usage.
>>> def set_events(): time.sleep(1); event0.set(); event1.set()
>>> threading.Thread(target=set_events).start()
>>> wait_for_events([event0, event1], timeout=30)
True
```

Parameters

- **events** (list(*threading.Event*)) – An Event or list of Events to wait on.
- **timeout** (*float|`astropy.units.Quantity`*) – Timeout in seconds to wait for events, default 600 seconds.
- **sleep_delay** (*float, optional*) – Time in seconds between event checks.
- **callback** (*callable*) – A periodic callback that should return *True* to continue waiting or *False* to interrupt the loop. Can also be used for e.g. custom logging.

Returns True if events were set, False otherwise.

Return type bool

Raises `error.Timeout` – Raised if events have not all been set before *timeout* seconds.

panoptes.utils.utils module

class panoptes.utils.utils.**DelaySigTerm**

Bases: contextlib.ContextDecorator

Supports delaying SIGTERM during a critical section.

This allows one to avoid having SIGTERM interrupt a critical block of code, such as saving to a database.

Example

with DelaySigTerm(): db.WriteCurrentRecord(record)

panoptes.utils.utils.**altaz_to_radec**(*alt=None*, *az=None*, *location=None*, *obstime=None*, ***kwargs*)

Convert alt/az degrees to RA/Dec SkyCoord.

```
>>> from panoptes.utils import altaz_to_radec
>>> from astropy.coordinates import EarthLocation
>>> from astropy import units as u
```

(continues on next page)

(continued from previous page)

```
>>> keck = EarthLocation.of_site('Keck Observatory')
...
```

```
>>> altaz_to_radec(alt=75, az=180, location=keck, obstime='2020-02-02T20:20:02.02
↪')
<SkyCoord (ICRS): (ra, dec) in deg
    (281.78..., 4.807...)>
```

```
>>> # Can use quantities or not.
>>> alt = 4500 * u.arcmin
>>> az = 180 * u.degree
>>> altaz_to_radec(alt=alt, az=az, location=keck, obstime='2020-02-02T20:20:02.02
↪')
<SkyCoord (ICRS): (ra, dec) in deg
    (281.78..., 4.807...)>
```

```
>>> # Will use current time if none given.
>>> altaz_to_radec(alt=35, az=90, location=keck)
<SkyCoord (ICRS): (ra, dec) in deg
    (..., ...)>
```

```
>>> # Must pass a `location` instance.
>>> altaz_to_radec()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  ...
    assert location is not None
AssertionError
```

Parameters

- **alt** (*astropy.units.Quantity or scalar*) – Altitude.
- **az** (*astropy.units.Quantity or scalar*) – Azimuth.
- **location** (*astropy.coordinates.EarthLocation, required*) – A valid location.
- **obstime** (*None, optional*) – Time for object, defaults to *current_time*

Returns Coordinates corresponding to the AltAz.

Return type astropy.coordinates.SkyCoord

panoptes.utils.utils.**get_free_space**(*directory=None*)

Return the amoung of freespace in gigabytes for given directory.

```
>>> from panoptes.utils import get_free_space
>>> get_free_space()
<Quantity ... Gbyte>
```

```
>>> get_free_space(directory='/')
<Quantity ... Gbyte>
```

Parameters **directory** (*str, optional*) – Path to directory. If None defaults to $PANDIR.

Returns The number of gigabytes avialable in folder.

Return type astropy.units.Quantity

panoptes.utils.utils.**get_quantity_value**(*quantity*, *unit=None*)

Thin-wrapper around the *astropy.units.Quantity.to_value* method.

If passed something other than a Quantity will simply return the original object.

```
>>> from astropy import units as u
>>> from panoptes.utils import get_quantity_value
```

```
>>> get_quantity_value(60 * u.second)
60.0
```

```
>>> # Can convert between units.
>>> get_quantity_value(60 * u.minute, unit='second')
3600.0
```

```
>>> get_quantity_value(60 * u.minute, unit=u.second)
3600.0
```

```
>>> get_quantity_value(60)
60
```

Parameters

- **quantity** (*astropy.units.Quantity or scalar*) – Quantity to extract numerical value from.
- **unit** (*astropy.units.Unit, optional*) – unit to convert to.

Returns numerical value of the Quantity after conversion to the specified unit.

Return type float

panoptes.utils.utils.**image_id_from_path**(*path*)

Return the *image_id* from the given path or uri.

```
>>> from panoptes.utils import image_id_from_path
>>> path = 'gs://panoptes-raw-images/PAN012/ee04d1/20190820T111638/
↪20190820T122447.fits'
>>> image_id_from_path(path)
'PAN012_ee04d1_20190820T122447'
```

```
>>> path = 'nothing/to/match'
>>> image_id_from_path(path)
```

Parameters **path** (*str*) – A path or uri for a file.

Returns The image id in the form "<unit_id>_<camera_id>_<image_id>"

Return type str

panoptes.utils.utils.**listify**(*obj*)

Given an object, return a list.

Always returns a list. If obj is None, returns empty list, if obj is list, just returns obj, otherwise returns list with obj as single member.

If a *dict* object is passed then this function will return a list of *only* the values.

```
>>> listify(42)
[42]
>>> listify('foo')
['foo']
>>> listify(None)
[]
>>> listify(['a'])
['a']

>>> my_dict = dict(a=42, b='foo')
>>> listify(my_dict)
[42, 'foo']
>>> listify(my_dict.values())
[42, 'foo']
>>> listify(my_dict.keys())
['a', 'b']
```

Returns You guessed it.

Return type list

panoptes.utils.utils.**moving_average**(*data_set*, *periods=3*)

Moving average.

Parameters

- **data_set** (*numpy.array*) – An array of values over which to perform the moving average.
- **periods** (*int, optional*) – Number of periods.

Returns An array of the computed averages.

Return type *numpy.array*

panoptes.utils.utils.**sequence_id_from_path**(*path*)

Return the *sequence_id* from the given path or uri.

```
>>> from panoptes.utils import sequence_id_from_path
>>> path = 'gs://panoptes-raw-images/PAN012/ee04d1/20190820T111638/
↪20190820T122447.fits'
>>> sequence_id_from_path(path)
'PAN012_ee04d1_20190820T111638'
```

```
>>> path = 'nothing/to/match'
>>> sequence_id_from_path(path)
```

Parameters **path** (*str*) – A path or uri for a file.

Returns The image id in the form "<unit_id>_<camera_id>_<sequence_id>"

Return type str

panoptes.utils.utils.**string_to_params**(*opts*)

Parses a single string into parameters that can be passed to a function.

A user of the *peas_shell* can supply positional and keyword arguments to the command being called, however the *Cmd* module that is used for the shell does not parse these options but instead passes this as a single string. This utility method does some simple parsing of that string and returns a list of positional parameters and a dictionary of keyword arguments. A keyword argument is considered anything that contains an equal sign (e.g. *exptime=30*). Any leading – to a keyword argument will be stripped during parsing.

A list of items can be passed by specifying the keyword argument multiple times.

Note: This function will attempt to parse keyword values as floats if possible. If a string is required include a single quote around the value, e.g. *param='42'* will keep the value as the string *'42'*.

```
>>> from panoptes.utils import string_to_params
>>> args, kwargs = string_to_params("parg1 parg2 key1=a_str key2=2 key2='2' key3=03")
>>> args
['parg1', 'parg2']
>>> kwargs
{'key1': 'a_str', 'key2': [2.0, '2'], 'key3': 3.0}
>>> isinstance(kwargs['key2'][0], float)
True
>>> isinstance(kwargs['key2'][1], str)
True
>>> kwargs['key2'][1] == '2'
True
```

```
>>> args, kwargs = string_to_params('--key1=val1 --key1-2=val1-2')
>>> kwargs
{'key1': 'val1', 'key1-2': 'val1-2'}
```

Parameters **opts** (*str*) – A single string containing everything beyond the actual command that is called.

Returns Returns a list of positional parameters and a dictionary of keyword arguments. These correspond to the *args and **kwargs that a typical function would receive.

Return type tuple(list, dict)

Module contents

Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W