POCS Documentation

Release unknown

Project PANOPTES

Jun 01, 2020

Contents

CHAPTER 1

PANOPTES Observatory Control System

Warning: The recent *v0.7.0* release of POCS is not backwards compatible. If you are one of the folks running that software, please either do a reinstall of your system using the instructions below or see our forum for advice.

1.1 Overview

PANOPTES is an open source citizen science project that is designed to find transiting exoplanets with digital cameras. The goal of PANOPTES is to establish a global network of of robotic cameras run by amateur astronomers schools in order to monitor, as continuously as possible, a very large number of stars. For more general information about the project, including the science case and resources for interested individuals, see the about page.

POCS (PANOPTES Observatory Control System) is the main software driver for the PANOPTES unit, responsible for high-level control of the unit. This repository also contains a number of scripts for running a full instance of POCS.

1.2 Getting Started

POCS is designed to control a fully constructed PANOPTES unit. Additionally, POCS can be run with simulators when hardware is not present or when the system is being developed.

For information on building a PANOPTES unit, see the main PANOPTES website and join the community forum.

To get started with POCS there are three easy steps:

1. **Setup** POCS on the computer you will be using for your unit or for development.
2. **Test** your POCS setup by running our testing script
3. **Start using POCS!**

See below for more details.

Note: We rely heavily on much of the code in *panoptes-utils*.

See https://github.com/panoptes/panoptes-utils

1.3 Install

1.3.1 POCS Environment

If you are running a PANOPTES unit then you will most likely want the entire PANOPTES environment.

There is a bash shell script that will attempt to install an entire working POCS system on your computer. Some folks even report that it works on a Mac.

To test the script, open a terminal and enter:

```
curl -L https://install.projectpanoptes.org | bash
```

Or using *wget*:

```
wget -O - https://install.projectpanoptes.org | bash
```

1.3.2 POCS Module

If you want just the POCS module, for instance if you want to override it in your own OCS (see Huntsman-POCS for an example), then install via *pip*:

```
pip install panoptes-pocs
```

If you want the extra features, such as Google Cloud Platform connectivity, then use the extras options:

```
pip install "panoptes-pocs[google]"
```

1.4 Test POCS

See the Testing section of the Contributing guide.

1.5 Links

- PANOPTES Homepage: https://projectpanoptes.org
- PANOPTES Data Explorer: https://www.panoptes-data.net
- Community Forum: https://forum.projectpanoptes.org
- Source Code: https://github.com/panoptes/POCS

CHAPTER 2

Contents

2.1 License

2.2 Contributors

- Wilfred Tyler Gee <wtylergee@gmail.com>
- Josh Walawender <joshwalawender@users.noreply.github.com>
- James Synge <jamessynge@users.noreply.github.com>
- Demezhan Marikov <demezhan1998@gmail.com>
- Anthony Horton <anthony.horton@mq.edu.au>

- Brendan Orenstein <brendan.orenstein@students.mq.edu.au>
- Mike Butterfield <github@mikebutterfield.com>
- TaylahB <taylah.beard@students.mq.edu.au>
- James Synge <james.synge@gmail.com>
- jermainegug <32515601+jermainegug@users.noreply.github.com>
- blackflip14 <cdkrogers@yahoo.com>
- danjampro <danjampro@sky.com>
- Sushant Mehta <mehtasushant05@gmail.com>
- kmeagle1515 <46345142+kmeagle1515@users.noreply.github.com>
- Dan Proole <danjampro@sky.com>
- Jenny Tong <mimming@google.com>
- Kate Storey-Fisher <ksf@google.com>
- Lee Spitler <lee.spitler@mq.edu.au>
- Luca <jeremylan@users.noreply.github.com>
- Sean Marquez <capsulecorplab@gmail.com>
- lucasholucasho <lucasho2340@gmail.com>
- megwill4268 <megan.will@ymail.com>

2.3 CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

2.3.1 [0.7.4] - 2020-05-31

Note that we skipped *0.7.2* and *0.7.3*.

Bug fixes

- Package name is *panoptes-pocs* for namespace consistency. (#971)
- README changed to rst. (#971)

2.3.2 [0.7.1] - 2020-05-31

If you thought 9 months between releases was a long time, how about 18 months! :) This version has a lot of breaking changes and is not backwards compatible with previous versions. The release is a (big) stepping stone on the way to `0.8.0` and (eventually!) a `1.0.0`.

The entire repo has been redesigned to support docker images. This comes with a number of changes, including the refactoring of many items into the panoptes-utils repo.

There are a lot of changes included in this release, highlights below:

Warning: This changelog is likely missing some things. The release was large. Too large. If you think something might be working different that just might be true. Check the forums.

Added

- Storing an explicit `safety` collection in the database.
- Configuration file specific for testing rather than relying on `pocs.yaml`.
- **Convenience scripts for running tests inside docker container:** `scripts/testing/test-software.sh`
- GitHub Actions for testing and coverage upload.

Changed

- Docker as default. (#951).
- **Weather items have moved to aag-weather.**
  - Two docker containers run from the `aag-weather` image and have a `docker/docker-compose-aag.yaml` file to start.
- **Config items related to the configuration system have been moved to the Config Server in `panoptes-utils` repo.**
  - The main interface for POCS related items is through `self.get_config`, which can take a key and a default, e.g. `self.get_config('mount.horizon', default='30 deg')`.
  - Test writing is affected and is currently more difficult than would be ideal. An updated test writing document will be following this release.
- **Logging has changed to loguru and has been greatly simplified:**
  - `get_root_logger` has been replaced by `get_logger`.
- **The `per-run` logs have been removed and have been replaced by two logs files:**
  - **`$PANDIR/logs/panoptes.log`: Log file meant for watching on the** command line (via `tail`) or for otherwise human-readable logs. Rotated daily at 11:30 am. Only the previous days' log is retained.
  - **`$PANDIR/logs/panoptes_YYYYMMDD.log`: Log file meant for archive** or information gathering. Stored in JSON format for ingestion into log analysis service. Rotated daily at 11:30 and stored in a compressed file for 7 days. Future updates will add option to upload to google servers.
- `loguru` provides two new log levels
  - `trace`: one level below `debug`.
  - `success`: one level above `info`.
- **Breaking Mount: unparking has been moved from the** `ready` to the `slewing` state. This fixes a problem where after waiting 10 minutes for observation check, the mount would move from park to home to park without checking weather safety.
- Documentation updates.
- Lots of conversions to `f-strings`.

- Renamed codecov configuration file to be compliant.
- Switch to pyscaffold for package maintenance.
- **"Waiting" method changes:**
  - *sleep* has been renamed to *wait*.
- All *status()* methods have been converted to properties that return a useful dict.
- Making proper abstractmethods.
- Documentation updates where found.
- Many log and f-string fixes.
- *pocs.config_port* property available publicly.
- horizon check for state happens directly in *run*.

Removed

- Cleanup of any stale or unused code.
- All `mongo` related code.
- **Consolidate configration files: `.pycodestyle.cfg`, `.coveragerc`** into `setup.cfg`.
- **Weather related items. These have been moved to `aag-weather <https://github.com/panoptes/aag-weather>`__.**
- **All notebook tutorials in favor of `panoptes-tutorials <https://github.com/panoptes/panoptes-tutorials>`__.**
- Remove all old install and startup scripts.

2.3.3 [0.6.2] - 2018-09-27

One week between releases is a lot better than 9 months! ;) Some small but important changes mark this release including faster testing times on local machines. Also a quick release to remove some of the CloudSQL features (but see the shiny new Cloud Functions over in the panoptes-network repo!).

Fixed

- Cameras
- **Use unit_id for sequence and image ids. Important for processing** consistency [#613].
- State Machine

Changed

- Camera
- Remove camera creation from Observatory [#612].
- Smarter event waiting [#625].
- **More cleanup, especially path names and pretty images [#610, #613,** #614, #620].
- Mount

- Testing
- Caching some of the build dirs [#611].
- **Only use Mongo DB type during local testing - Local testing with** 1/3rd the wait! [#616].
- Google Cloud [#599]
- Storage improvements [#601].

Added

- Misc
- CountdownTimer utility [#625].

Removed

- Google Cloud [#599]
- Reverted some of the CloudSQL connectivity [#652]
- Cameras
- Remove spline smoothing focus [#621].

2.3.4 [0.6.1] - 2018-09-20

Lots of changes in this release. In particular we've pushed through a

lot of changes | (especially with the help of @jamessynge) to make the development process a lot | smoother. This has in turn contribute to the quality of the codebase.

Too long between releases but even more exciting improvements to come! Next up is tackling the events notification system, which will let us start having some vastly improved UI features.

Below is a list of some of the changes.

Thanks to first-time contributors: @jermainegug @jeremylan as well as contributions from many folks over at https://github.com/AstroHuntsman/huntsman-pocs.

Fixed

- Cameras
- Fix for DATE-OBS fits header [#589].
- Better property settings for DSLRs [#589].
- Pretty image improvements [#589].
- Autofocus improvements for SBIG/Focuser [#535].
- Primary camera updates [#614, 620].
- Many bug fixes [#457, #589].
- State Machine

- Many fixes [#509, #518].

Changed

- Mount
- **POCS Shell: Hitting `Ctrl-c` will complete movement through states** [#590].
- Pointing updates, including `auto_correct` [#580].
- **Tracking mode updates (fixes for Northern Hemisphere only!)** [#549].
- Serial interaction improvements [#388, #403].
- Shutdown improvements [#407, #421].
- Dome
- Changes from May Huntsman commissioning run [#535]
- Messaging
- Better and consistent topic terminology [#593, #605].
- Anticipation of coming events.
- Misc
- Default to rereading the fields file for targets [#488].
- Timelapse updates [#523, #591].

Added

- Cameras
- Basic scripts for bias and dark frames.
- Add support for Optec FocusLynx based focus controllers [#512].
- Pretty images from FITS files. Thanks @jermainegug! [#538].
- Testing
- pyflakes testing support for bug squashing! :bettle: [#596].
- pycodestyle for better code! [#594].
- Threads instead of process [#468].
- Fix coverage & Travis config for concurrency [#566].
- Google Cloud [#599]
- Added instructions for authentication [#600].
- Add a `pan_id` to units for GCE interaction[#595].
- Adding Google CloudDB interaction [#602].
- Sensors
- Much work on arduinos and sensors [#422].
- Misc
- Startup scripts for easier setup [#475].

- Install scripts for Ubuntu 18.04 [#585].
- New database type: mongo, file, memory [#414].
- **Twitter! Slack! Social median interactions. Hooray! Thanks** @jeremylan! [#522]

2.3.5 [0.6.0] - 2017-12-30

Changed

- **Enforce 100 character limit for code** 159.
- **Using root-relative module imports** 252.
- **`Observatory` is now a parameter for a POCS instance** 195.
- **Better handling of simulator types** 200.
- Log improvements:
- **Separate files for each level and new naming scheme** 165.
- **Reduced log format** 254.
- **Better reusing of logger** 192.
- **Single shared MongoClient connection** 228.
- **Improvements to build process** 176, 166.
- **State machine location more flexible** 209, 219
- **Testing improvments** 249.
- Updates to many wiki pages.
- Misc bug fixes and improvements.

Added

- **Merge PEAS into POCS** 169.
- **Merge PACE into POCS** 167.
- **Support added for testing of serial devices** 164, 180.
- **Basic dome support** 231, 248.
- **Polar alignment helper functions moved from PIAA** 265.

Removed

- **Remove threading support from rs232.SerialData** 148.

2.3.6 [0.5.1] - 2017-12-02

Added

- First real release!

- Working POCS features:
- mount (iOptron)
- cameras (DSLR, SBIG)
- focuer (Birger)
- scheduler (simple)
- Relies on separate repositories PEAS and PACE
- Automated testing with travis-ci.org
- Code coverage via codecov.io
- Basic install scripts

2.4 panoptes

2.4.1 panoptes package

Subpackages

panoptes.peas package

Submodules

panoptes.peas.remote_sensors module

class panoptes.peas.remote_sensors.**RemoteMonitor**(*endpoint_url=None*, *sensor_name=None*, **args*, ***kwargs*)

Bases: object

Does a pull request on an endpoint to obtain a JSON document.

capture(*store_result=True*)

Read JSON from endpoint url and capture data.

Note: Currently this doesn't do any processing or have a callback.

Returns Dictionary of sensors keyed by sensor name.

Return type sensor_data (dict)

disconnect()

panoptes.peas.sensors module

class panoptes.peas.sensors.**ArduinoSerialMonitor**(*sensor_name=None*, *auto_detect=False*, **args*, ***kwargs*)

Bases: object

Monitors the serial lines and tries to parse any data recevied as JSON.

Checks for the *camera_box* and *computer_box* entries in the config and tries to connect. Values are updated in the database.

capture(*store_result=True*)
Helper function to return serial sensor info.

Reads each of the connected sensors. If a value is received, attempts to parse the value as json.

Returns Dictionary of sensors keyed by sensor name.

Return type sensor_data (dict)

disconnect()

panoptes.peas.sensors.**auto_detect_arduino_devices**(*comports=None*)

panoptes.peas.sensors.**detect_board_on_port**(*port*)
Open a port and determine which type of board its producing output.

Returns: (name, serial_reader) if we can read a line of JSON from the port, parse it and find a 'name' attribute in the top-level object. Else returns None.

panoptes.peas.sensors.**find_arduino_devices**()
Find devices (paths or URLs) that appear to be Arduinos.

Returns //host:port arduino_simulator://?board=camera).

Return type a list of strings; device paths (e.g. /dev/ttyACM1) or URLs (e.g. rfc2217

Module contents

panoptes.pocs package

Subpackages

panoptes.pocs.camera package

Subpackages

panoptes.pocs.camera.simulator package

Submodules

panoptes.pocs.camera.simulator.dslr module

Module contents

panoptes.pocs.camera.simulator_sdk package

Submodules

panoptes.pocs.camera.simulator_sdk.ccd module

Module contents

Submodules

panoptes.pocs.camera.camera module

panoptes.pocs.camera.canon_gphoto2 module

panoptes.pocs.camera.fli module

panoptes.pocs.camera.libasi module

panoptes.pocs.camera.libfli module

panoptes.pocs.camera.libfliconstants module

panoptes.pocs.camera.sbig module

panoptes.pocs.camera.sbigudrv module

panoptes.pocs.camera.sdk module

panoptes.pocs.camera.zwo module

Module contents

panoptes.pocs.dome package

Submodules

panoptes.pocs.dome.abstract_serial_dome module

panoptes.pocs.dome.astrohaven module

panoptes.pocs.dome.bisque module

panoptes.pocs.dome.protocol_astrohaven_simulator module

panoptes.pocs.dome.simulator module

Module contents

panoptes.pocs.filterwheel package

Submodules

panoptes.pocs.filterwheel.filterwheel module

panoptes.pocs.scheduler.dispatch module

panoptes.pocs.scheduler.field module

panoptes.pocs.scheduler.observation module

panoptes.pocs.scheduler.scheduler module

Module contents

panoptes.pocs.sensors package

Submodules

panoptes.pocs.sensors.arduino_io module

class panoptes.pocs.sensors.arduino_io.**ArduinoIO**(*board*, *serial_data*, *db*)
Bases: object

Supports reading from and writing to Arduinos.

The readings (python dictionaries) are recorded in a PanDB collection in the following form:

‘‘‘

{ ‘name’: self.board, ‘timestamp’: t, ‘data’: reading

}

’’’

connect()
Connect to the port.

disconnect()
Disconnect from the port.

Will re-open automatically if reading or writing occurs.

get_reading()
Reads and returns a single reading.

handle_command(*msg*)
Handle one relay command.

TODO(jamessynge): Add support for ‘set_relay’, where we look up the relay name in self._last_reading to confirm that it exists on this device.

handle_commands()
Read and process commands for up to 1 second.

Returns when there are no more commands available from the command subscriber, or when a second has passed. The interval is 1 second because we expect at least 2 seconds between reports, and also expect that it probably doesn’t take more than 1 second for each report to be read. We could make this configurable, or could dynamically adjust, such as by polling for input.

handle_reading(*reading*)
Saves a reading as the last_reading and writes to output_queue.

read_and_record()
Try to get the next reading and, if successful, record it.

Write the reading to the appropriate PanDB collections and to the appropriate message topic.

If there is an interruption in success in reading from the device, we announce (log) the start and end of that situation.

reconnect()
Disconnect from and connect to the serial port.

This supports handling a SerialException, such as when the USB bus is reset.

run()
Main loop for recording data and reading commands.

This only ends if an Exception is unhandled or if a 'shutdown' command is received. The most likely exception is from SerialData.get_and_parse_reading() in the event that the device disconnects from USB.

stop_running

write(*text*)
Writes text (a string) to the port.

Returns: the number of bytes written.

panoptes.pocs.sensors.arduino_io.**auto_detect_arduino_devices**(*ports=None*)
Returns a list of tuples of (board_name, port).

panoptes.pocs.sensors.arduino_io.**detect_board_on_port**(*port*)
Determine which type of board is attached to the specified port.

Returns: Name of the board (e.g. 'camera_board') if we can read a line of JSON from the port, parse it and find a 'name' attribute in the top-level object. Else returns None.

panoptes.pocs.sensors.arduino_io.**get_arduino_ports**()
Find ports (device paths or URLs) that appear to be Arduinos.

Returns //host:port arduino_simulator://?board=camera).

Return type a list of strings; device paths (e.g. /dev/ttyACM1) or URLs (e.g. rfc2217

panoptes.pocs.sensors.arduino_io.**open_serial_device**(*port*, *serial_config=None*, ***kwargs*)
Creates an rs232.SerialData for port, assumed to be an Arduino.

Default parameters are provided when creating the SerialData instance, but may be overridden by serial_config or kwargs.

Parameters

- **serial_config** – dictionary (or None) with serial settings from config file, suitable for passing to SerialData or to a PySerial instance.
- ****kwargs** – Any other parameters to be passed to SerialData. These have higher priority than the serial_config parameter.

Module contents

panoptes.pocs.state package

Subpackages

panoptes.pocs.state.states package

Subpackages

panoptes.pocs.state.states.default package

Submodules

panoptes.pocs.state.states.default.analyzing module

panoptes.pocs.state.states.default.analyzing.**on_enter**(*event_data*)

panoptes.pocs.state.states.default.housekeeping module

panoptes.pocs.state.states.default.housekeeping.**on_enter**(*event_data*)

panoptes.pocs.state.states.default.observing module

panoptes.pocs.state.states.default.observing.**on_enter**(*event_data*)
Take an observation image.

This state is responsible for taking the actual observation image.

panoptes.pocs.state.states.default.parked module

panoptes.pocs.state.states.default.parked.**on_enter**(*event_data*)

panoptes.pocs.state.states.default.parking module

panoptes.pocs.state.states.default.parking.**on_enter**(*event_data*)

panoptes.pocs.state.states.default.pointing module

panoptes.pocs.state.states.default.ready module

panoptes.pocs.state.states.default.ready.**on_enter**(*event_data*)
Once in the *ready* state our unit has been initialized successfully. The next step is to schedule something for the night.

panoptes.pocs.state.states.default.scheduling module

panoptes.pocs.state.states.default.scheduling.**on_enter**(*event_data*)
In the *scheduling* state we attempt to find a field using our scheduler. If field is found, make sure that the field is up right now (the scheduler should have taken care of this). If observable, set the mount to the field and calls *start_slewing* to begin slew.

If no observable targets are available, *park* the unit.

panoptes.pocs.state.states.default.sleeping module

panoptes.pocs.state.states.default.sleeping.**on_enter**(*event_data*)

panoptes.pocs.state.states.default.slewing module

panoptes.pocs.state.states.default.slewing.**on_enter**(*event_data*)
Once inside the slewing state, set the mount slewing.

panoptes.pocs.state.states.default.tracking module

panoptes.pocs.state.states.default.tracking.**on_enter**(*event_data*)
The unit is tracking the target. Proceed to observations.

Module contents

Module contents

Submodules

panoptes.pocs.state.machine module

class panoptes.pocs.state.machine.**PanStateMachine**(*state_machine_table*, ***kwargs*)
Bases: transitions.core.Machine

A finite state machine for PANOPTES.

The state machine guides the overall action of the unit.

after_state(*event_data*)
Called after each state.

Parameters **event_data** (*transitions.EventData*) – Contains information about the event

before_state(*event_data*)
Called before each state.

Parameters **event_data** (*transitions.EventData*) – Contains information about the event

check_safety(*event_data=None*)
Checks the safety flag of the system to determine if safe.

This will check the weather station as well as various other environmental aspects of the system in order to determine if conditions are safe for operation.

Note: This condition is called by the state machine during each transition

Parameters

- **event_data** (*transitions.EventData*) – carries information about the event if

- **from the state machine.** (*called*) –

Returns Latest safety flag

Return type bool

goto_next_state()
Make a transition to the next state.

Each state is responsible for setting the *next_state* property based off the logic that happens inside the state. This method will look up the transition method to reach the next state and call that method.

If no transition method is defined for whatever is set as *next_state* then the *park* method will be called.

Returns If state was successfully changed.

Return type bool

classmethod load_state_table(*state_table_name='simple_state_table'*)
Loads the state table

Parameters **state_table_name** (*str*) – Name of state table. Corresponds to file name in *$POCS/resources/state_table/* directory or to absolute path if starts with "/". Default 'simple_state_table'.

Returns Dictionary with *states* and *transitions* keys.

Return type dict

mount_is_initialized(*event_data*)
Transitional check for mount.

This is used as a conditional check when transitioning between certain states.

mount_is_tracking(*event_data*)
Transitional check for mount.

This is used as a conditional check when transitioning between certain states.

next_state

run(*exit_when_done=False*, *run_once=False*)
Runs the state machine loop

This runs the state machine in a loop. Setting the machine property *is_running* to False will stop the loop.

Parameters

- **exit_when_done** (*bool, optional*) – If True, the loop will exit when *do_states* has become False, otherwise will wait (default)
- **run_once** (*bool, optional*) – If the machine loop should only run one time, defaults to False to loop continuously.

stop_states()
Stops the machine loop on the next iteration

Module contents

panoptes.pocs.tests package

Subpackages

panoptes.pocs.tests.bisque package

Submodules

panoptes.pocs.tests.bisque.test_dome module

panoptes.pocs.tests.bisque.test_mount module

panoptes.pocs.tests.bisque.test_run module

Module contents

panoptes.pocs.tests.serial_handlers package

Submodules

panoptes.pocs.tests.serial_handlers.protocol_buffers module

panoptes.pocs.tests.serial_handlers.protocol_hooked module

panoptes.pocs.tests.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.pocs.tests.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 subclass. 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.

Submodules

panoptes.pocs.tests.test_astrohaven_dome module

panoptes.pocs.tests.test_base module

panoptes.pocs.tests.test_base_scheduler module

panoptes.pocs.tests.test_camera module

panoptes.pocs.tests.test_codestyle module

panoptes.pocs.tests.test_constraints module

panoptes.pocs.tests.test_dispatch_scheduler module

panoptes.pocs.tests.test_dome_simulator module

panoptes.pocs.tests.test_field module

panoptes.pocs.tests.test_filterwheel module

panoptes.pocs.tests.test_focuser module

panoptes.pocs.tests.test_images module

panoptes.pocs.tests.test_ioptron module

panoptes.pocs.tests.test_mount module

panoptes.pocs.tests.test_mount_simulator module

panoptes.pocs.tests.test_observation module

panoptes.pocs.tests.test_observatory module

panoptes.pocs.tests.test_pocs module

panoptes.pocs.tests.test_rs232 module

panoptes.pocs.tests.test_scheduler module

panoptes.pocs.tests.test_state_machine module

Module contents

Submodules

panoptes.pocs.base module

panoptes.pocs.core module

panoptes.pocs.hardware module

Information about hardware supported by Panoptes.

panoptes.pocs.hardware.**get_all_names**(*all_names=['camera', 'dome', 'mount', 'night', 'weather'], without=[]*)

Returns the names of all the categories of hardware that POCS supports.

Note that this doesn't extend to the Arduinos for the telemetry and camera boards, for which no simulation is supported at this time.

panoptes.pocs.hardware.**get_simulator_names**(*simulator=None, kwargs=None, config=None*)

Returns the names of the simulators to be used in lieu of hardware drivers.

Note that returning a list containing 'X' doesn't mean that the config calls for a driver of type 'X'; that is up to the code working with the config to create drivers for real or simulated hardware.

This funciton is intended to be called from PanBase or similar, which receives kwargs that may include simulator, config or both. For example:

get_simulator_names(config=self.config, kwargs=kwargs)

Or: get_simulator_names(simulator=simulator, config=self.config)

The reason this function doesn't just take **kwargs as its sole arg is that we need to allow for the case where the caller is passing in simulator (or config) twice, once on its own, and once in the kwargs (which won't be examined). Python doesn't permit a keyword argument to be passed in twice.

Parameters

- **simulator** – An explicit list of names of hardware to be simulated (i.e. hardware drivers to be replaced with simulators).

- **kwargs** – The kwargs passed in to the caller, which is inspected for an arg called 'simulator'.
- **config** – Dictionary created from pocs.yaml or similar.

Returns List of names of the hardware to be simulated.

panoptes.pocs.images module

panoptes.pocs.observatory module

Module contents

Module contents

2.5 CONTRIBUTING GUIDE

Please see the code of conduct for our playground rules and follow them during all your contributions.

2.5.1 Getting Started

We prefer that all changes to POCS have an associated GitHub Issue in the project that explains why it is needed. This allows us to debate the best approach to address the issue before folks spend a lot of time writing code. If you are unsure about a possible contribution to the project, please contact the project owners about your idea; of course, an issue is a good way to do this.

2.5.2 Pull Request Process

Note: This is a summary of the process. See the POCS wiki for more info.

- Pre-requisites
- Ensure you have a github account.
- Setup ssh access for github.
- If the change you wish to make is not already an Issue in the project, please create one specifying the need.

Process

1. Create a fork of the repository via github (button in top-right).
2. Clone your fork to your local system:

```
cd $PANDIR
git clone git@github.com:YOUR-GITHUB-NAME/POCS.git
```

3. Set the "upstream" branch to `panoptes` and fetch the upstream changes:

```
cd POCS
git remote add upstream https://github.com/panoptes/POCS.git
git fetch upstream
```

4. Use a topic branch within your fork to make changes. All of our repositories have a default branch of `develop` when you first clone them, but your work should be in a separate branch (see note below). Your branch should be based off of the `upstream/develop` branch.

   Create a branch with a descriptive name, e.g.:

```
git checkout -b new-camera-simulator upstream/develop
git checkout -b issue-28 upstream/develop
```

5. Ensure that your code meets this project's standards (see Testing and Code Formatting below).
6. Run the testing suite locally to ensure that all tests are passing. See Testing below.
7. Submit a pull request to the repository, be sure to reference the issue number it addresses.

Note: See "A successful Git branching model" for details on how the repository is structured.

2.5.3 Code Formatting

- All Python should use PEP 8 Standards
- Line length is set at 100 characters instead of 80.
- It is recommended to have your editor auto-format code whenever you save a file rather than attempt to go back and change an entire file all at once. There are many plugins that exist for this.
- You can also use yapf (Yet Another Python Formatter) for which POCS includes a style file (.style.yapf). For example:

```
# cd to the root of your workspace.
cd $(git rev-parse --show-toplevel)
# Format the modified python files in your workspace.
yapf -i $(git diff --name-only | egrep '\.py$')``
```

- Do not leave in commented-out code or unnecessary whitespace.
- Variable/function/class and file names should be meaningful and descriptive.
- File names should be lower case and underscored, not contain spaces. For example, `my_file.py` instead of `My File.py`.
- Define any project specific terminology or abbreviations you use in the file you use them.

2.5.4 Log Messages

Use appropriate logging:

- DEBUG (i.e. `self.logger.debug()`) should attempt to capture all run*time information.
- INFO (i.e. `self.logger.info()`) should be used sparingly and meant to convey information to a person actively watching a running unit.

- WARNING (i.e. `self.logger.warning()`) should alert when something does not go as expected but operation of unit can continue.
- ERROR (i.e. `self.logger.error()`) should be used at critical levels when operation cannot continue.
- The logger supports variable information without the use of the `format` method.
- There is a `say` method available on the main `POCS` class that is meant to be used in friendly manner to convey information to a user. This should be used only for personable output and is typically displayed in the "chat box"of the PAWS website. These messages are also sent to the INFO level logger.

Logging examples:

Note: These are meant to illustrate the logging calls and are not necessarily indicative of real operation

```
self.say("I'm all ready to go, first checking the weather")

self.logger.info(f'PANOPTES unit initialized: {self.name}')

self.logger.debug("Setting up weather station")

self.logger.warning(f'Problem getting wind safety: {e!r}')

self.logger.debug(f'Rain: {is_raining} Clouds: {is_cloudy} Dark: {is_dark} Temp:
↪{temp:.02f}')

self.logger.error('Unable to connect to AAG Cloud Sensor, cannot continue')
```

Viewing log files

- You typically want to follow an active log file by using `tail -F` on the command line.

```
tail -F $PANDIR/logs/panoptes.log
```

2.5.5 Test POCS

POCS comes with a testing suite that allows it to test that all of the software works and is installed correctly. Running the test suite by default will use simulators for all of the hardware and is meant to test that the software works correctly. Additionally, the testing suite can be run with various flags to test that attached hardware is working properly.

Software Testing

There are a few scenarios where you want to run the test suite:

1. You are getting your unit ready and want to test software is installed correctly.
2. You are upgrading to a new release of software (POCS, its dependencies or the operating system).
3. You are helping develop code for POCS and want test your code doesn't break something.

Testing your installation

In order to test your installation you should have followed all of the steps above for getting your unit ready. To run the test suite, you will need to open a terminal and navigate to the `$POCS` directory.

```
cd $POCS

# Run the software testing
scripts/testing/test-software.sh
```

Note: The test suite will give you some warnings about what is going on and give you a chance to cancel the tests (via `Ctrl-c`).

It is often helpful to view the log output in another terminal window while the test suite is running:

```
# Follow the log file
tail -F $PANDIR/logs/panoptes.log
```

Testing your code changes

Note: This step is meant for people helping with software development.

The testing suite will automatically be run against any code committed to our github repositories. However, the test suite should also be run locally before pushing to github. This can be done either by running the entire test suite as above or by running an individual test related to the code you are changing. For instance, to test the code related to the cameras one can run:

```
pytest -xv pocs/tests/test_camera.py
```

Here the `-x` option will stop the tests upon the first failure and the `-v` makes the testing verbose. Note that some tests might require additional software. This software is installed in the docker image, which is used by the `test-software.sh` script above), but is **not** used when calling `pytest` directly. For instance, anything requiring plate solving needs `astrometry.net` installed.

Any new code should also include proper tests. See below for details.

Writing tests

All code changes should include tests. We strive to maintain a high code coverage and new code should necessarily maintain or increase code coverage. For more details see the Writing Tests page.

Hardware Testing

Hardware testing uses the same testing suite as the software testing but with additional options passed on the command line to signify what hardware should be tested.

The options to pass to `pytest` is `--with-hardware`, which accepts a list of possible hardware items that are connected. This list includes `camera`, `mount`, and `weather`. Optionally you can use `all` to test a fully connected unit.

Warning: The hardware tests do not perform safety checking of the weather or dark sky. The `weather` test mentioned above tests if a weather station is connected but does not test the safety conditions. It is assumed that hardware testing is always done with direct supervision.

```
# Test an attached camera
pytest --with-hardware=camera

# Test an attached camera and mount
pytest --with-hardware=camera,mount

# Test a fully connected unit
pytest --with-hardware=all
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search
- *CONTRIBUTING GUIDE*

Python Module Index

p

Index

S

W