
panoptes-pocs Documentation

Release 0.0.post1.dev50+g2a8b019

Project PANOPTES

Apr 16, 2024

CONTENTS

1	Welcome to POCS documentation!	1
2	Project PANOPTES	3
3	POCS	5
4	Contents	9
5	Indices and tables	109
	Python Module Index	111
	Index	113

WELCOME TO POCS DOCUMENTATION!

PROJECT PANOPTES

PANOPTES is an open source citizen science project designed to find [transiting exoplanets](#) with digital cameras. The goal of PANOPTES is to establish a global network of robotic cameras run by amateur astronomers and schools (or anyone!) 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 [project overview](#).

POCS (PANOPTES Observatory Control System) is the main software driver for a PANOPTES unit, responsible for high-level control of the unit.

For more information, see the full documentation at: <https://pocs.readthedocs.io>.

3.1 Install

3.1.1 POCS Environment

If you are running a PANOPTES unit then you will most likely want an entire PANOPTES environment, which includes the necessary tools for operation of a complete unit.

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

To install POCS via the script, open a terminal and enter (you may be prompted for your `sudo` password):

```
curl -fsSL https://install.projectpanoptes.org > install-pocs.sh
bash install-pocs.sh
```

Or using `wget`:

```
wget -qO- https://install.projectpanoptes.org > install-pocs.sh
bash install-pocs.sh
```

The install script will ask a few questions at the beginning of the process. If you are unsure of the answer the default is probably okay.

In addition to installing POCS, the install script will create the Config Server and Power Monitor services, which will automatically be restarted upon reboot of the computer.

3.1.2 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,focuser,testing]"
```

Running POCS

POCS requires a few things to properly run:

1. A `panoptes-utils` `config-server` running to provide dynamic configuration.
2. An Observatory instance that has details about the location of a POCS unit (real or simulated), which hardware is available, etc.

A minimal working example with a simulated Observatory would be:

```
import os
from panoptes.utils.config.server import config_server
from panoptes.pocs.core import POCS

os.environ['PANDIR'] = '/var/panoptes'
conf_server = config_server('conf_files/pocs.yaml')
I 01-20 01:01:10.886 Starting panoptes-config-server with config_file='conf_files/pocs.
→yaml'
S 01-20 01:01:10.926 Config server Loaded 17 top-level items
I 01-20 01:01:10.928 Config items saved to flask config-server
I 01-20 01:01:10.934 Starting panoptes config server with localhost:6563

pocs = POCS.from_config(simulators=['all'])
I 01-20 01:01:20.408 Initializing PANOPTES unit - Generic PANOPTES Unit - Mauna Loa
→Observatory
I 01-20 01:01:20.419 Making a POCS state machine from panoptes
I 01-20 01:01:20.420 Loading state table: panoptes
S 01-20 01:01:20.485 Unit says: Hi there!
W 01-20 01:01:20.494 Scheduler not present
W 01-20 01:01:20.495 Cameras not present
W 01-20 01:01:20.496 Mount not present
I 01-20 01:01:20.497 Scheduler not present, cannot get current observation.

pocs.initialize()
W 01-20 01:01:28.386 Scheduler not present
W 01-20 01:01:28.388 Cameras not present
W 01-20 01:01:28.389 Mount not present
S 01-20 01:01:28.390 Unit says: Looks like we're missing some required hardware.
Out[10]: False
```

For a more realistic usage, see the full documentation at: <https://pocs.readthedocs.io>.

For actually deploying a PANOPTES unit, refer to the [Operating Guider](#).

Developing POCS

See [Coding in PANOPTES](#)

Testing

To test the software, you can use the standard `pytest` tool from the root of the directory.

By default all tests will be run. If you want to run one specific test, give the specific filename as an argument to `pytest`:

```
pytest tests/test_mount.py
```

3.2 Links

- PANOPTES Homepage: <https://www.projectpanoptes.org>
- Forum: <https://forum.projectpanoptes.org>
- Documentation: <https://pocs.readthedocs.io>
- Source Code: <https://github.com/panoptes/POCS>

CONTENTS

4.1 panoptes

4.1.1 panoptes namespace

Subpackages

panoptes.pocs package

Subpackages

panoptes.pocs.camera package

Subpackages

panoptes.pocs.camera.gphoto package

Submodules

panoptes.pocs.camera.gphoto.base module

class panoptes.pocs.camera.gphoto.base.**AbstractGPhotoCamera**(*arg, **kwargs)

Bases: *AbstractCamera*, *ABC*

Abstract camera class that uses gphoto2 interaction.

Parameters

config (*Dict*) – Config key/value pairs, defaults to empty dict.

command(cmd: *List[str] | str*, check_exposing: *bool* = *True*)

Run gphoto2 command.

connect()

property cooling_power

Get current power level of the camera's image sensor cooling system (typically as a percentage of the maximum).

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

get_command_result(*timeout: float = 10*) → List[str] | None

Get the output from the command.

Accepts a *timeout* param for communicating with the process.

Returns a list of strings corresponding to the output from the gphoto2 camera or *None* if no command has been specified.

get_property(*prop: str*) → str

Gets a property from the camera

classmethod gphoto_file_download(*port: str, filename_pattern: str, only_new: bool = True*)

Downloads (newer) files from the camera on the given port using the filename pattern.

property is_exposing

True if an exposure is currently under way, otherwise False.

load_properties() → dict

Load properties from the camera.

Reads all the configuration properties available via gphoto2 and returns as dictionary.

process_exposure(*metadata, **kwargs*)

Converts the CR2 to FITS then processes image.

set_properties(*prop2index: Dict[str, int] = None, prop2value: Dict[str, str] = None*)

Sets a number of properties all at once, by index or value.

Parameters

- **prop2index** (*dict* or *None*) – A dict with keys corresponding to the property to be set and values corresponding to the index option.
- **prop2value** (*dict* or *None*) – A dict with keys corresponding to the property to be set and values corresponding to the literal value.

set_property(*prop: str, val: str | int, is_value: bool = False, is_index: bool = False*)

Set a property on the camera.

Parameters

- **prop** (*str*) – The property to set.
- **val** (*str*, *int*) – The value to set the property to.
- **is_value** (*bool*) – If True, then the value is a literal value. Default False.
- **is_index** (*bool*) – If True, then the value is an index. Default False.

Raises

ValueError – If the property is not found.

classmethod start_tether(*port, filename_pattern: str = '%Y%m%dT%H%M%S.%C'*)

Start a tether for gphoto2 auto-download on given port using filename pattern.

property target_temperature

Get current value of the target temperature for the camera's image sensor cooling control.

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property temperature

Get current temperature of the camera's image sensor.

Note: this only needs to be implemented for cameras which can provided this information, e.g. those with cooled image sensors.

property uid: str

A six-digit serial number for the camera

panoptes.pocs.camera.gphoto.canon module

```
class panoptes.pocs.camera.gphoto.canon.Camera(readout_time: float = 1.0, file_extension: str = 'cr2',
                                                connect: bool = True, *args, **kwargs)
```

Bases: [AbstractGPhotoCamera](#)

property bit_depth

ADC bit depth.

connect()

Connect to Canon DSLR.

Gets the serial number from the camera and sets various settings.

property egain

Image sensor gain in e-/ADU as reported by the camera.

```
classmethod get_shutterspeed_index(seconds: float, return_minimum: bool = False)
```

Looks up the appropriate shutterspeed setting for the given seconds.

If the given seconds does not match a set shutterspeed, the 'bulb' setting is returned.

panoptes.pocs.camera.gphoto.remote module

```
class panoptes.pocs.camera.gphoto.remote.Camera(endpoint: AnyHttpUrl = 'http://localhost:6565',
                                                *args, **kwargs)
```

Bases: [Camera](#)

A remote gphoto2 camera class.

```
command(cmd, endpoint: AnyHttpUrl = None)
```

Run the gphoto2 command remotely.

This assumes the remote camera service is running at the endpoint specified on the camera object or passed to the method.

```
get_command_result(timeout: float = 10) → List[str] | None
```

Get the output from the remote camera service.

property is_exposing

True if an exposure is currently under way, otherwise False.

Module contents

panoptes.pocs.camera.simulator package

Submodules

panoptes.pocs.camera.simulator.ccd module

```
class panoptes.pocs.camera.simulator.ccd.Camera(name='Simulated SDK camera', driver=<class
'panoptes.pocs.camera.simulator.ccd.SDKDriver'>,
target_temperature=<Quantity 0. deg_C>, *args,
**kwargs)
```

Bases: [AbstractSDKCamera](#), [Camera](#), [ABC](#)

connect()

Connect to camera simulator

The simulator merely marks the *connected* property.

property cooling_enabled

Get current status of the camera's image sensor cooling system (enabled/disabled).

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property cooling_power

Get current power level of the camera's image sensor cooling system (typically as a percentage of the maximum).

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property target_temperature

Get current value of the target temperature for the camera's image sensor cooling control.

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property temperature

Get current temperature of the camera's image sensor.

Note: this only needs to be implemented for cameras which can provided this information, e.g. those with cooled image sensors.

```
class panoptes.pocs.camera.simulator.ccd.SDKDriver(library_path=None, **kwargs)
```

Bases: [AbstractSDKDriver](#)

get_SDK_version()

Get the version of the SDK

get_devices()

Get connected device UIDs and corresponding device nodes/handles/IDs.

panoptes.pocs.camera.simulator.dslr module

class panoptes.pocs.camera.simulator.dslr.**Camera**(name='Simulated Camera', *args, **kwargs)

Bases: [AbstractCamera](#)

property bit_depth

ADC bit depth.

connect()

Connect to camera simulator

The simulator merely marks the *connected* property.

property gain

Image sensor gain in e-/ADU as reported by the camera.

take_observation(observation, headers=None, filename=None, *args, **kwargs)

Take an observation

Gathers various header information, sets the file path, and calls

take_exposure. Also creates a *threading.Event* object and a *threading.Thread* object. The Thread calls *process_exposure* after the exposure had completed and the Event is set once *process_exposure* finishes.

Parameters

- **observation** (*Observation*) – Object describing the observation
- **headers** (*dict* or *Header*, *optional*) – Header data to be saved along with the file.
- **filename** (*str*, *optional*) – pass a filename for the output FITS file to override the default file naming system.
- **blocking** (*bool*) – If method should wait for observation event to be complete before returning, default False.
- ****kwargs** (*dict*) – Optional keyword arguments (*exptime*, *dark*)

Returns

The metadata from the event.

Return type

dict

Module contents

Submodules

panoptes.pocs.camera.camera module

class panoptes.pocs.camera.camera.**AbstractCamera**(name='Generic Camera', model='simulator', port=None, primary=False, *args, **kwargs)

Bases: [PanBase](#)

Base class for all cameras.

filter_type

Type of filter attached to camera. If a filterwheel is present this will return the `filterwheel.current_filter` property, otherwise it will return the value of the `filter_type` keyword argument, or if that argument was not given it will query the camera driver, e.g. 'M' for unfiltered monochrome camera, 'RGGB' for Bayer matrix colour camera.

Type`str`**focuser**

Focuser for the camera, default None.

Type``panoptes.pocs.focuser.AbstractFocuser`|None`**filter_wheel**

Filter wheel for the camera, default None.

Type``panoptes.pocs.filterwheel.AbstractFilterWheel`|None`**uid**

Unique identifier of the camera.

Type`str`**is_primary**

If this camera is the primary camera for the system, default False.

Type`bool`**model**

The model of camera, such as 'gphoto2', 'sbig', etc. Default 'simulator'.

Type`str`**name**

Name of the camera, default 'Generic Camera'.

Type`str`**port**

The port the camera is connected to, typically a usb device, default None.

Type`str`**temperature**

Current temperature of the image sensor.

Type`astropy.units.Quantity`**target_temperature**

image sensor cooling target temperature.

Type

astropy.units.Quantity

temperature_tolerance

tolerance for image sensor temperature.

Type

astropy.units.Quantity

cooling_enabled

True if image sensor cooling is active.

Type

bool

cooling_power

Current image sensor cooling power level in percent.

Type

astropy.unit.Quantity

egain

Image sensor gain in e-/ADU as reported by the camera.

Type

astropy.units.Quantity

gain

The gain setting of the camera (ZWO cameras only).

Type

int

bitdepth

ADC bit depth in bits.

Type

astropy.units.Quantity

image_type

Image format of the camera, e.g. 'RAW16', 'RGB24' (ZWO cameras only).

Type

str

timeout

max time to wait after exposure before TimeoutError.

Type

astropy.units.Quantity

readout_time

approximate time to readout the camera after an exposure.

Type

float

file_extension

file extension used by the camera's image data, e.g. 'fits'

Type

str

library_path

path to camera library, e.g. `‘/usr/local/lib/libfli.so’` (SBIG, FLI, ZWO)

Type

`str`

properties

A collection of camera properties as read from the camera.

Type

`dict`

is_connected

True if camera is connected.

Type

`bool`

is_cooled_camera

True if camera has image sensor cooling capability.

Type

`bool`

is_temperature_stable

True if image sensor temperature is stable.

Type

`bool`

is_exposing

True if an exposure is currently under way, otherwise False.

Type

`bool`

is_ready

True if the camera is ready to take an exposure.

Type

`bool`

can_take_internal_darks

True if the camera can take internal dark exposures.

Type

`bool`

Notes

The port parameter is not used by SBIG or ZWO cameras, and is deprecated for FLI cameras. For these cameras serial_number should be passed to the constructor instead. For SBIG and FLI this should simply be the serial number engraved on the camera case, whereas for ZWO cameras this should be the 8 character ID string previously saved to the camera firmware. This can be done using ASICAP, or `panoptes.pocs.camera.libasi.ASIDriver.set_ID()`.

autofocus(*seconds=None, focus_range=None, focus_step=None, cutout_size=None, keep_files=None, take_dark=None, merit_function='vollath_F4', merit_function_kwargs=None, mask_dilations=None, coarse=False, make_plots=None, blocking=False, *args, **kwargs*)

Focuses the camera using the specified merit function. Optionally performs a coarse focus to find the approximate position of infinity focus, which should be followed by a fine focus before observing.

Parameters

- **seconds** (*scalar, optional*) – Exposure time for focus exposures, if not specified will use value from config.
- **focus_range** (*2-tuple, optional*) – Coarse & fine focus sweep range, in encoder units. Specify to override values from config.
- **focus_step** (*2-tuple, optional*) – Coarse & fine focus sweep steps, in encoder units. Specify to override values from config.
- **cutout_size** (*int, optional*) – Size of square central region of image to use, default 500 x 500 pixels.
- **keep_files** (*bool, optional*) – If True will keep all images taken during focusing. If False (default) will delete all except the first and last images from each focus run.
- **take_dark** (*bool, optional*) – If True will attempt to take a dark frame before the focus run, and use it for dark subtraction and hot pixel masking, default True.
- **merit_function** (*str/callable, optional*) – Merit function to use as a focus metric, default `vollath_F4`.
- **merit_function_kwargs** (*dict or None, optional*) – Dictionary of additional keyword arguments for the merit function.
- **mask_dilations** (*int, optional*) – Number of iterations of dilation to perform on the saturated pixel mask (determine size of masked regions), default 10
- **coarse** (*bool, optional*) – Whether to perform a coarse focus, otherwise will perform a fine focus. Default False.
- **(bool (make_plots))** – Whether to write focus plots to images folder, default behaviour is to check the focuser `autofocus_make_plots` attribute.
- **optional** – Whether to write focus plots to images folder, default behaviour is to check the focuser `autofocus_make_plots` attribute.
- **blocking** (*bool, optional*) – Whether to block until autofocus complete, default False.

Returns

Event that will be set when autofocusing is complete

Return type

`threading.Event`

Raises

ValueError – If invalid values are passed for any of the focus parameters.

property `bit_depth`

ADC bit depth.

property `can_take_internal_darks`

True if the camera can take internal dark exposures. This will be true of cameras that have an internal mechanical shutter and can be commanded to keep that shutter closed during the exposure. For cameras that either lack a mechanical shutter or lack the option to keep it closed light must be kept out of the camera during dark exposures by other means, e.g. an opaque blank in a filterwheel, a lens cap, etc.

abstract connect()

property cooling_enabled

Get current status of the camera's image sensor cooling system (enabled/disabled).

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property cooling_power

Get current power level of the camera's image sensor cooling system (typically as a percentage of the maximum).

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property egain

Image sensor gain in e-/ADU as reported by the camera.

property exposure_error

Error message from the most recent exposure or None, if there was no error.

property file_extension

File extension for images saved by camera

property filter_type

Image sensor filter type (e.g. 'RGGB') or name of the current filter (e.g. 'g2_3')

get_cutout(*seconds*, *file_path*, *cutout_size*, *keep_file=False*, **args*, ***kwargs*)

Takes an image and returns a thumbnail cutout.

Takes an image, grabs the data, deletes the FITS file and returns a cutout from the centre of the image.

Parameters

- **seconds** (*astropy.units.Quantity*) – exposure time, Quantity or numeric type in seconds.
- **file_path** (*str*) – path to (temporarily) save the image file to.
- **cutout_size** (*int*) – size of the square region of the centre of the image to return.
- **keep_file** (*bool*, *optional*) – if True the image file will be deleted, if False it will be kept.
- ***args** – passed to the *take_exposure* method
- ****kwargs** – passed to the *take_exposure* method

property has_filterwheel

Return True if the camera has a filterwheel, False if not.

property has_focuser

Return True if the camera has a focuser, False if not.

property is_connected

Is the camera available via gphoto2

property is_cooled_camera

True if camera has image sensor cooling capability

property is_exposing

True if an exposure is currently under way, otherwise False.

property is_observing

True if an observation is currently under, otherwise False.

property is_ready

True if camera is ready to start another exposure, otherwise False.

property is_temperature_stable

True if image sensor temperature is stable, False if not.

See also: See *temperature_tolerance* for more information about the temperature stability. An uncooled camera, or cooled camera with cooling disabled, will always return False.

process_exposure(metadata, **kwargs)

Processes the exposure.

Performs the following steps:

1. First checks to make sure that the file exists on the file system.
2. Calls *_process_fits* with the filename and info, which is specific to each camera.
3. Makes pretty images if requested.
4. Records observation metadata if requested.
5. Compress FITS files if requested.
6. Sets the *observation_event*.

If the camera is a primary camera, extract the jpeg image and save metadata to database *current* collection. Saves metadata to *observations* collection for all images.

Parameters

metadata (*dict*) – Header metadata saved for the image.

Raises

FileNotFoundError – If the FITS file isn't at the specified location.

property readiness

Dictionary detailing the readiness of the camera system to take an exposure.

property readout_time

Readout time for the camera in seconds

take_exposure(*seconds=<Quantity 1. s>*, *filename=None*, *metadata=None*, *dark=False*, *blocking=False*, *timeout=<Quantity 10. s>*, **args*, ***kwargs*) → *Thread*

Take an exposure for given number of seconds and saves to provided filename.

Parameters

- **seconds** (*u.second*, *optional*) – Length of exposure.
- **filename** (*str*, *optional*) – Image is saved to this filename.
- **metadata** (*dict*, *optional*) – Add key/value as FITS header. Does not support nested dicts.
- **dark** (*bool*, *optional*) – Exposure is a dark frame, default False. On cameras that support taking dark frames internally (by not opening a mechanical shutter) this will be done, for other cameras the light must be blocked by some other means. In either case

setting `dark` to `True` will cause the *IMAGETYP* FITS header keyword to have value 'Dark Frame' instead of 'Light Frame'. Set `dark` to `None` to disable the *IMAGETYP* keyword entirely.

- **blocking** (*bool*, *optional*) – If `False` (default) returns immediately after starting the exposure, if `True` will block until it completes and file exists.
- **timeout** (*astropy.Quantity*) – The timeout to use for the exposure, default 10 seconds. The timeout gets added to the *seconds* and the *self.readout_time* to get the total timeout for the exposure. If the exposure takes longer than this then a *panoptes.utils.error.Timeout* exception will be raised.

Returns

The readout thread, which joins when readout has finished.

Return type

`threading.Thread`

Raises

- **error.PanError** – If camera is not connected.
- **error.Timeout** – If the exposure takes longer than total *timeout* to complete.

take_observation(*observation*, *headers=None*, *filename=None*, *blocking=False*, ***kwargs*) → `dict`

Take an observation

Gathers various header information, sets the file path, and calls

take_exposure. Also creates a *threading.Event* object and a *threading.Thread* object. The Thread calls *process_exposure* after the exposure had completed and the Event is set once *process_exposure* finishes.

Parameters

- **observation** (*Observation*) – Object describing the observation
- **headers** (*dict* or *Header*, *optional*) – Header data to be saved along with the file.
- **filename** (*str*, *optional*) – pass a filename for the output FITS file to override the default file naming system.
- **blocking** (*bool*) – If method should wait for observation event to be complete before returning, default `False`.
- ****kwargs** (*dict*) – Optional keyword arguments (*exptime*, *dark*)

Returns

The metadata from the event.

Return type

`dict`

property target_temperature

Get current value of the target temperature for the camera's image sensor cooling control.

Note: this only needs to be implemented for cameras which have cooled image sensors, not for those that don't (e.g. DSLRs).

property temperature

Get current temperature of the camera's image sensor.

Note: this only needs to be implemented for cameras which can provided this information, e.g. those with cooled image sensors.

property temperature_tolerance

Get current value of the image sensor temperature tolerance.

If the image sensor temperature differs from the target temperature by more than the temperature tolerance then the temperature is not considered stable (by `is_temperature_stable`) and, for cooled cameras, `is_ready` will report False.

property uid

Return unique identifier for camera.

property waiting_for_readout

True if the most recent readout has not finished. Should be set in `write_fits`

write_fits(data, header, filename)

Write the FITS file.

This is a thin-wrapper around the `fits_utils.write_fits` method that marks the readout as complete.

panoptes.pocs.camera.fli module

```
class panoptes.pocs.camera.fli.Camera(name='FLI Camera', target_temperature=<Quantity 25. deg_C>,
                                       *args, **kwargs)
```

Bases: [`AbstractSDKCamera`](#)

connect()

Connect to FLI camera.

Gets a 'handle', serial number and specs/capabilities from the driver

property cooling_enabled

Current status of the camera's image sensor cooling system (enabled/disabled).

Note: For FLI cameras this is always True, and cannot be set.

property cooling_power

Current power level of the camera's image sensor cooling system (as a percentage of the maximum).

property is_exposing

True if an exposure is currently under way, otherwise False

property target_temperature

Current value of the target temperature for the camera's image sensor cooling control.

Can be set by assigning an `astropy.units.Quantity`.

property temperature

Current temperature of the camera's image sensor.

panoptes.pocs.camera.libasi module

class panoptes.pocs.camera.libasi.ASIDriver(*library_path=None, **kwargs*)

Bases: *AbstractSDKDriver*

close_camera(*camera_ID*)

Close camera with given integer ID

disable_dark_subtract(*camera_ID*)

Disable dark subtraction.

May need to call this as dark current subtraction settings persist in the registry on Windows.

enable_dark_subtract(*camera_ID, dark_file_path*)

Enable dark subtraction (not implemented).

You almost certainly wouldn't want to use this as it only works with images taken in RGB8 format and only with dark frames saved as .BMP files. Far better to do dark subtraction in post-processing.

get_ID(*camera_ID*)

Get string ID from firmware for the camera with given integer ID

The saved ID is an array of 8 unsigned chars for some reason.

get_SDK_version()

Get the version of the ZWO ASI SDK

get_camera_mode(*camera_ID*)

Get current trigger mode for camera with given integer ID.

get_camera_property(*camera_index*)

Get properties of the camera with given index

get_camera_property_by_id(*camera_ID*)

Get properties of the camera with a given integer ID.

get_camera_supported_mode(*camera_ID*)

Get supported trigger modes for camera with given integer ID.

get_control_caps(*camera_ID*)

Gets the details of all the controls supported by the camera with given integer ID

get_control_value(*camera_ID, control_type*)

Gets the value of the control *control_type* from camera with given integer ID

get_devices()

Gets currently connected camera info.

Returns

All currently connected camera serial numbers with corresponding integer camera IDs.

Return type

dict

Notes

If a camera does not have a serial number it will attempt to fall back to string ID. Cameras with neither serial number nor string ID will be left out of the dictionary as they have no unique identifier.

get_dropped_frames(*camera_ID*)

Get the number of dropped frames during video capture.

get_exposure_data(*camera_ID*, *width*, *height*, *image_type*)

Get image data from exposure on camera with given integer ID

get_exposure_status(*camera_ID*)

Get status of current exposure on camera with given integer ID

get_gain_offset(*camera_ID*)

Get pre-setting parameters.

get_num_of_connected_cameras()

Get the count of connected ASI cameras

get_num_of_controls(*camera_ID*)

Gets the number of control types supported by the camera with given integer ID

get_product_ids()

Get product IDs of cameras supported by the SDK.

get_roi_format(*camera_ID*)

Get the ROI size and image format setting for camera with given integer ID

get_serial_number(*camera_ID*)

Get serial number of the camera with given integer ID.

The serial number is an array of 8 unsigned chars, the same as string ID, but it is interpreted differently. It is displayed in ASICAP as a 16 digit hexadecimal number, so we will convert it the same 16 character string representation.

get_start_position(*camera_ID*)

Get position of the upper left corner of the ROI for camera with given integer ID

Parameters

camera_ID (*int*) – integer ID of the camera

Returns

x, y coordinates of the upper left

corner of the ROI. Note, these are in binned pixels.

Return type

(astropy.units.Quantity, astropy.units.Quantity)

get_trigger_output_io_conf(*camera_ID*)

Get external trigger configuration of the camera with given integer ID.

get_video_data(*camera_ID*, *width*, *height*, *image_type*, *timeout*)

Get the image data from the next available video frame

init_camera(*camera_ID*)

Initialise camera with given integer ID

open_camera(*camera_ID*)

Open camera with given integer ID

pulse_guide_off(*camera_ID, direction*)

Turn off PulseGuide on ST4 port of given camera in given direction.

pulse_guide_on(*camera_ID, direction*)

Turn on PulseGuide on ST4 port of given camera in given direction.

send_soft_trigger(*camera_ID, start_stop_signal*)

Send out a soft trigger on camera with given integer ID.

set_ID(*camera_ID, string_ID*)

Save string ID to firmware of camera with given integer ID

The saved ID is an array of 8 unsigned chars for some reason. To preserve some sanity this method takes an 8 byte UTF-8 string as input.

set_camera_mode(*camera_ID, mode_name*)

Set trigger mode for camera with given integer ID.

set_control_value(*camera_ID, control_type, value*)

Sets the value of the control *control_type* on camera with given integer ID

set_roi_format(*camera_ID, width, height, binning, image_type*)

Set the ROI size and image format settings for the camera with given integer ID

set_start_position(*camera_ID, start_x, start_y*)

Set position of the upper left corner of the ROI for camera with given integer ID

set_trigger_output_io_conf(*camera_ID, pin, pin_high, delay, duration*)

Set external trigger configuration of the camera with given integer ID.

start_exposure(*camera_ID*)

Start exposure on the camera with given integer ID

start_video_capture(*camera_ID*)

Start video capture mode on camera with given integer ID

stop_exposure(*camera_ID*)

Cancel current exposure on camera with given integer ID

stop_video_capture(*camera_ID*)

Stop video capture mode on camera with given integer ID

```
class panoptes.pocs.camera.libasi.BayerPattern(value, names=None, *, module=None,
                                              qualname=None, type=None, start=1,
                                              boundary=None)
```

Bases: `IntEnum`

Bayer filter type

BG = 1

GB = 3

GR = 2

RG = 0

class panoptes.pocs.camera.libasi.CameraInfo

Bases: Structure

Camera info structure

bayer_pattern

Structure/Union member

bit_depth

Structure/Union member

camera_ID

Structure/Union member

e_per_adu

Structure/Union member

has_ST4_port

Structure/Union member

has_cooler

Structure/Union member

has_mechanical_shutter

Structure/Union member

is_USB3_camera

Structure/Union member

is_USB3_host

Structure/Union member

is_color_camera

Structure/Union member

is_trigger_camera

Structure/Union member

max_height

Structure/Union member

max_width

Structure/Union member

name

Structure/Union member

pixel_size

Structure/Union member

supported_bins

Structure/Union member

supported_video_format

Structure/Union member

unused

Structure/Union member

```
class panoptes.pocs.camera.libasi.CameraMode(value, names=None, *, module=None, qualname=None,
                                              type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Camera status

END = -1

NORMAL = 0

TRIG_FALL_EDGE = 3

TRIG_HIGH_LEVEL = 5

TRIG_LOW_LEVEL = 6

TRIG_RISE_EDGE = 2

TRIG_SOFT_EDGE = 1

TRIG_SOFT_LEVEL = 4

```
class panoptes.pocs.camera.libasi.ControlCaps
```

Bases: `Structure`

Structure for caps (limits) on allowable parameter values for each camera control

control_type

Structure/Union member

default_value

Structure/Union member

description

Structure/Union member

is_auto_supported

Structure/Union member

is_writable

Structure/Union member

max_value

Structure/Union member

min_value

Structure/Union member

name

Structure/Union member

unused

Structure/Union member

```
class panoptes.pocs.camera.libasi.ControlType(value, names=None, *, module=None, qualname=None,
                                              type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Control types

```
ANTI_DEW_HEATER = 21
AUTO_MAX_BRIGHTNESS = 12
AUTO_MAX_EXP = 11
AUTO_MAX_GAIN = 10
AUTO_TARGET_BRIGHTNESS = 12
BANDWIDTHOVERLOAD = 6
BRIGHTNESS = 5
COOLER_ON = 17
COOLER_POWER_PERC = 15
EXPOSURE = 1
FAN_ON = 19
FLIP = 9
GAIN = 0
GAMMA = 2
HARDWARE_BIN = 13
HIGH_SPEED_MODE = 14
MONO_BIN = 18
OFFSET = 5
OVERCLOCK = 7
PATTERN_ADJUST = 20
TARGET_TEMP = 16
TEMPERATURE = 8
WB_B = 4
WB_R = 3
```

```
class panoptes.pocs.camera.libasi.ErrorCode(value, names=None, *, module=None, qualname=None,
                                             type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Error codes

```
BUFFER_TOO_SMALL = 13
CAMERA_CLOSED = 4
CAMERA_REMOVED = 5
END = 18
```

```
EXPOSURE_IN_PROGRESS = 15
GENERAL_ERROR = 16
INVALID_CONTROL_TYPE = 3
INVALID_FILEFORMAT = 7
INVALID_ID = 2
INVALID_IMGTYPE = 9
INVALID_INDEX = 1
INVALID_MODE = 17
INVALID_PATH = 6
INVALID_SEQUENCE = 12
INVALID_SIZE = 8
OUTOF_BOUNDARY = 10
SUCCESS = 0
TIMEOUT = 11
VIDEO_MODE_ACTIVE = 14
```

```
class panoptes.pocs.camera.libasi.ExposureStatus(value, names=None, *, module=None,
                                                  qualname=None, type=None, start=1,
                                                  boundary=None)
```

Bases: `IntEnum`

Exposure status codes

```
FAILED = 3
IDLE = 0
SUCCESS = 2
WORKING = 1
```

```
class panoptes.pocs.camera.libasi.FlipStatus(value, names=None, *, module=None, qualname=None,
                                              type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Flip status

```
BOTH = 3
HORIZ = 1
NONE = 0
VERT = 2
```

```
class panoptes.pocs.camera.libasi.GuideDirection(value, names=None, *, module=None,
                                                  qualname=None, type=None, start=1,
                                                  boundary=None)

    Bases: IntEnum
    Guider direction
    EAST = 2
    NORTH = 0
    SOUTH = 1
    WEST = 3

class panoptes.pocs.camera.libasi.ID
    Bases: Structure
    id
        Structure/Union member

class panoptes.pocs.camera.libasi.ImgType(value, names=None, *, module=None, qualname=None,
                                           type=None, start=1, boundary=None)

    Bases: IntEnum
    Supported video format
    END = -1
    RAW16 = 2
    RAW8 = 0
    RGB24 = 1
    Y8 = 3

class panoptes.pocs.camera.libasi.SupportedMode
    Bases: Structure
    Array of supported CameraModes, terminated with CameraMode.END
    modes
        Structure/Union member

class panoptes.pocs.camera.libasi.TrigOutput(value, names=None, *, module=None, qualname=None,
                                              type=None, start=1, boundary=None)

    Bases: IntEnum
    External trigger output.
    NONE = -1
    PINA = 0
    PINB = 1
```

panoptes.pocs.camera.libfli module

Low level interface to the FLI library

Reproduces in Python (using ctypes) the C interface provided by FLI's library.

class panoptes.pocs.camera.libfli.FLIDriver(library_path=None, **kwargs)

Bases: *AbstractSDKDriver*

FLIClose(handle)

Close a handle to an FLI device.

Parameters

handle (*ctypes.c_long*) – handle to close

FLIExposeFrame(handle)

Expose a frame for a given camera.

This function exposes a frame according the settings (image area, exposure time, binning, etc.) of the camera. The settings must have been previously set to valid values using the appropriate FLISet* methods. This function is non-blocking and returns once the exposure has stated.

Parameters

handle (*ctypes.c_long*) – handle of the camera to start the exposure on.

FLIGetArrayArea(handle)

Get the array area of the give camera.

This function finds the total area of the CCD array for a given camera. This area is specified in terms of an upper left point and a lower right point.

Parameters

handle (*ctypes.c_long*) – handle of the camera to get the array area of.

FLIGetCoolerPower(handle)

Get the cooler power level for a given camera.

Parameters

handle (*ctypes.c_long*) – handle of the camera to get the cooler power of.

Returns

cooler power, in percent.

Return type

float

FLIGetExposureStatus(handle)

Get the remaining exposure time of a given camera.

Parameters

handle (*ctypes.c_long*) – handle of the camera to get the remaining exposure time of.

Returns

remaining exposure time

Return type

astropy.units.Quantity

FLIGetFWRevision(handle)

Get firmware revision of a given device

Parameters

handle (*ctypes.c_long*) – handle of the camera device to get the firmware revision of.

Returns

firmware revision of the camera

Return type

int

FLIGetHWRevision(*handle*)

Get hardware revision of a given device

Parameters

handle (*ctypes.c_long*) – handle of the camera device to get the hardware revision of.

Returns

hardware revision of the cameras

Return type

int

FLIGetModel(*handle*)

Get the model of a given device.

Parameters

handle (*ctypes.c_long*) – handle of the device to get the model of.

Returns

model of the device

Return type

string

FLIGetPixelSize(*handle*)

Get the dimensions of a pixel in the array of a given device.

Parameters

handle (*ctypes.c_long*) – handle of the device to find the pixel size of.

Returns

(x, y) dimensions of a pixel.

Return type

astropy.units.Quantity

FLIGetSerialString(*handle*)

Get the serial string of a given camera.

Parameters

handle (*ctypes.c_long*) – handle of the camera device to get the serial strong of.

Returns

serial string of the camera

Return type

string

FLIGetTemperature(*handle*)

Get the temperature of a given camera.

Parameters

handle (*ctypes.c_long*) – handle of the camera device to get the temperature of.

Returns

temperature of the camera cold finger in degrees Celsius

Return type

`astropy.units.Quantity`

FLIGetVisibleArea(*handle*)

Get the visible array area of the give camera.

This function finds the visible area of the CCD array for a given camera. This area is specified in terms of an upper left point and a lower right point.

Parameters

handle (`ctypes.c_long`) – handle of the camera to get the array area of.

FLIGrabFrame(*handle*, *width*, *height*)

Grabs an image frame from a given camera.

This function grabs the entire image frame from the specified camera and returns it as a numpy array. The width and height of the image must be specified. The width and height should be consistent with the call to `FLISetImageArea()` that preceded the call to `FLIExposeFrame()`. This function should not be called until the exposure is complete, which can be confirmed with `FLIGetExposureStatus()`.

Parameters

- **handle** (`ctypes.c_long`) – handle of the camera to grab a frame from.
- **width** (`int`) – width of the image frame in pixels
- **height** (`int`) – height of the image frame in pixels

Returns

image from the camera

Return type

`numpy.ndarray`

FLIGrabRow(*handle*, *width*)

Grabs a row of image data from a given camera.

This function grabs the next available row of imae data from the specificed camera and returns it as a nupt array. The widht of the row must be specified. The width should be consistent with the call to `FLISetImageArea()` that preceded the call to `FLIExposeFrame()`. This function should not be called until the exposure is complete, which can be confirmed with `FLIGetExposureStatus`.

Parameters

- **handle** (`ctypes.c_long`) – handle of the camera to grab a row from.
- **width** (`int`) – width of the image row in pixelStart

Returns

row of image data

Return type

`numpy.ndarray`

FLIList(*interface_type*=2, *device_type*=256)

List available devices.

This function returns a list of available FLI devices, including the device port and model name.

Parameters

- **interface_type** (*int*, *optional*) – interface to search for connected devices. Valid values are libfli.FLIDOMAIN_USB (default), FLIDOMAIN_PARALLEL_PORT, FLIDOMAIN_SERIAL, FLIDOMAIN_SERIAL_1200, FLIDOMAIN_SERIAL_19200, FLIDOMAIN_INET.
- **device_types** (*int*, *optional*) – device type to search for. Valid values are libfli.FLIDEVICE_CAMERA (default), FLIDEVICE_FILTERWHEEL, FLIDEVICE_HS_FILTERWHEEL, FLIDEVICE_FOCUSER, FLIDEVICE_ENUMERATE_BY_CONNECTION, FLIDEVICE_RAW.

Returns

(port, model name) for each available device

Return type

list of tuples

FLIOpen(*port*, *interface_type*=2, *device_type*=256)

Get a handle to an FLI device.

This function requires the port, interface type and device type of the requested device. Valid ports can be obtained with the FLIList() method.

Parameters

- **port** (*str*) – port that the device is connected to, e.g. /dev/fliusb0
- **interface_type** (*int*, *optional*) – interface type of the requested device. Valid values are libfli.FLIDOMAIN_USB (default), FLIDOMAIN_PARALLEL_PORT, FLIDOMAIN_SERIAL, FLIDOMAIN_SERIAL_1200, FLIDOMAIN_SERIAL_19200, FLIDOMAIN_INET.
- **device_type** (*int*, *optional*) – device type of the requested device. Valid values are libfli.FLIDEVICE_CAMERA (default), FLIDEVICE_FILTERWHEEL, FLIDEVICE_HS_FILTERWHEEL, FLIDEVICE_FOCUSER, FLIDEVICE_ENUMERATE_BY_CONNECTION, FLIDEVICE_RAW.

Returns

an opaque handle used by library functions to refer to FLI hardware

Return type

ctypes.c_long

FLISetExposureTime(*handle*, *exposure_time*)

Set the exposure time for a camera.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera to set the exposure time of.
- **exposure_time** (*u.Quantity*) – required exposure time. A simple numeric type can be given instead of a Quantity, in which case the units are assumed to be seconds.

FLISetFrameType(*handle*, *frame_type*)

Set the frame type for a given camera.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera to set the frame type of.
- **frame_type** (*int*) – frame type. Valid values are libfli.FLI_FRAME_TYPE_NORMAL,
- **FLI_FRAME_TYPE_DARK** –
- **FLI_FRAME_TYPE_FLOOD** –

- **FLI_FRAME_TYPE_RBI_FLUSH.** –

FLISetHBin(*handle*, *bin_factor*)

Set the horizontal bin factor for a given camera.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera to set the horizontal bin factor for.
- **bin_factor** (*int*) – horizontal bin factor. The valid range is from 1 to 16 inclusive.

FLISetImageArea(*handle*, *upper_left*, *lower_right*)

Set the image area for a given camera.

This function sets the image area to an area specified in terms of an upperleft point and a lower right point. Note that the lower right point coordinate must take into account the horizontal and vertical bin factor settings, but the upper left coordinate is absolute.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera to set the image area of.
- **upper_left** (*int*, *int*) – (x, y) coordinate of upper left point
- **lower_right** (*int*, *int*) – (x, y) coordinate of lower right point

FLISetNFlashes(*handle*, *n_flushes*)

Set the number of flushes for a given camera.

This function sets the number of the times the CCD array of the camera is flushed before exposing a frame. Some FLI cameras support background flushing. Background flushing continuously flushes the CCD eliminating the need for pre-exposure flushings.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera to set the number of flushes for.
- **n_flushes** (*int*) – number of times to flush the CCD array before an exposure. The valid range is from 0 to 16 inclusive.

FLISetTemperature(*handle*, *temperature*)

Set the temperature of a given camera.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera device to set the temperature of.
- **temperature** (*astropy.units.Quantity*) – temperature to set the cold finger of the camera to. A simple numeric type can be given instead of a Quantity, in which case the units are assumed to be degrees Celsius.

FLISetVBin(*handle*, *bin_factor*)

Set the vertical bin factor for a given camera.

Parameters

- **handle** (*ctypes.c_long*) – handle of the camera to set the vertical bin factor for.
- **bin_factor** (*int*) – vertical bin factor. The valid range is from 1 to 16 inclusive.

get_SDK_version()

Get the version of the SDK

get_devices()

Gets currently connected camera info.

Returns

All currently connected camera serial numbers with corresponding device nodes.

Return type

dict

panoptes.pocs.camera.libfliconstants module**panoptes.pocs.camera.sbig module**

class panoptes.pocs.camera.sbig.Camera(name='SBIG Camera', *args, **kwargs)

Bases: *AbstractSDKCamera*

connect()

Connect to SBIG camera.

Gets a 'handle', serial number and specs/capabilities from the driver

property cooling_enabled

Current status of the camera's image sensor cooling system (enabled/disabled).

Can be set by assigning a bool.

property cooling_power

Current power level of the camera's image sensor cooling system (as a percentage of the maximum).

property gain

Image sensor gain in e-/ADU as reported by the camera.

property is_exposing

True if an exposure is currently under way, otherwise False

property target_temperature

Current value of the target temperature for the camera's image sensor cooling control.

Can be set by assigning an astropy.units.Quantity.

property temperature

Current temperature of the camera's image sensor.

panoptes.pocs.camera.sbigudrv module

Low level interface to the SBIG Universal Driver/Library.

Reproduces in Python (using ctypes) the C interface provided by SBIG's shared library, i.e. 1 function that does 72 different things selected by passing an integer as the first argument. This is basically a direct translation of the enums and structs defined in the library C-header to Python dicts and ctypes.Structures, plus a class (SBIGDriver) to load the library and call the single command function (SBIGDriver._send_command()).

class panoptes.pocs.camera.sbigudrv.CFWCommand(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `IntEnum`

Filter wheel command enum

`CLOSE_DEVICE = 5`

`GET_INFO = 3`

`GOTO = 1`

`INIT = 2`

`OPEN_DEVICE = 4`

`QUERY = 0`

```
class panoptes.pocs.camera.sbigudrv.CFWError(value, names=None, *, module=None, qualname=None,
                                              type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Filter wheel errors enum

`BAD_COMMAND = 2`

`BAD_MODEL = 5`

`BUSY = 1`

`CAL_ERROR = 3`

`DEVICE_NOT_CLOSED = 6`

`DEVICE_NOT_OPEN = 7`

`I2C_ERROR = 8`

`MOTOR_TIMEOUT = 4`

`NONE = 0`

```
class panoptes.pocs.camera.sbigudrv.CFWGetInfoSelect(value, names=None, *, module=None,
                                                       qualname=None, type=None, start=1,
                                                       boundary=None)
```

Bases: `IntEnum`

Filter wheel get info select enum

`CAL_DATA = 1`

`DATA_REGISTERS = 2`

`FIRMWARE_VERSION = 0`

```
class panoptes.pocs.camera.sbigudrv.CFWModelSelect(value, names=None, *, module=None,
                                                     qualname=None, type=None, start=1,
                                                     boundary=None)
```

Bases: `IntEnum`

Filter wheel model selection enum

```
AUTO = 6
CFW10 = 8
CFW10_SERIAL = 9
CFW1603 = 13
CFW2 = 1
CFW402 = 5
CFW5 = 2
CFW6A = 7
CFW8 = 3
CFW9 = 10
CFWL = 4
CFWL8 = 11
CFWL8G = 12
FW5_8300 = 15
FW5_STF_DETENT = 19
FW5_STX = 14
FW7_STX = 17
FW8_8300 = 16
FW8_STT = 18
UNKNOWN = 0
```

```
class panoptes.pocs.camera.sbigudrv.CFWParams
```

Bases: Structure

ctypes Structure used to hold the parameters for the CFW (colour filter wheel) command

cfwCommand

Structure/Union member

cfwModel

Structure/Union member

cfwParam1

Structure/Union member

cfwParam2

Structure/Union member

inLength

Structure/Union member

inPtr

Structure/Union member

outLength

Structure/Union member

outPtr

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**CFWResults**

Bases: Structure

ctypes Structure used to fold the results from the CFW (colour filter wheel) command

cfwError

Structure/Union member

cfwModel

Structure/Union member

cfwPosition

Structure/Union member

cfwResults1

Structure/Union member

cfwResults2

Structure/Union member

cfwStatus

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**CFWStatus**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: [IntEnum](#)

Filter wheel status enum

BUSY = 2

IDLE = 1

UNKNOWN = 0

class panoptes.pocs.camera.sbigudrv.**EndExposureParams**

Bases: Structure

ctypes Structure to hold the parameters for the End Exposure command.

ccd

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**EndReadoutParams**

Bases: Structure

ctypes Structure to hold the parameters for the End Readout Params.

ccd

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**EstablishLinkParams**

Bases: Structure

ctypes Structure to hold the parameters for the Establish Link command.

sbigUseOnly

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**EstablishLinkResults**

Bases: Structure

ctypes Structure to hold the results from the Establish Link command.

cameraType

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetCCDInfoParams**

Bases: Structure

ctypes Structure to hold the parameters for the Get CCD Info command, used obtain the details & capabilities of the connected camera.

request

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetCCDInfoResults0**

Bases: Structure

ctypes Structure to hold the results from the Get CCD Info command when used with requests 'CCD_INFO_IMAGING' or 'CCD_INFO_TRACKING'.

The firmwareVersion field is 4 digit binary coded decimal of the form XX.XX.

cameraType

Structure/Union member

firmwareVersion

Structure/Union member

name

Structure/Union member

readoutInfo

Structure/Union member

readoutModes

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetCCDInfoResults2**

Bases: Structure

ctypes Structure to hold the results from the Get CCD Info command when used with request 'CCD_INFO_EXTENDED'.

badColumns

Structure/Union member

columns

Structure/Union member

imagingABG

Structure/Union member

serialNumber

Structure/Union member

class panoptes.pocs.camera.sbigudrv.GetCCDInfoResults4

Bases: Structure

ctypes Structure to hold the results from the Get CCD Info command when used with requests 'CCD_INFO_EXTENDED2_IMAGING' or 'CCD_INFO_EXTENDED2_TRACKING'.

The capabilitiesBits is a bitmap, yay.

capabilities_b0

Structure/Union member

capabilities_b1

Structure/Union member

capabilities_b2

Structure/Union member

capabilities_b3

Structure/Union member

capabilities_b4

Structure/Union member

capabilities_b5

Structure/Union member

capabilities_unused

Structure/Union member

dumpExtra

Structure/Union member

class panoptes.pocs.camera.sbigudrv.GetCCDInfoResults6

Bases: Structure

ctypes Structure to hold the results from the Get CCD Info command when used with the request 'CCD_INFO_EXTENDED3'.

The sbigudrv.h C header says there should be three bitmask fields, each of type ulong, which would be 64 bits on this platform (OS X), BUT trial and error has determined they're actually 32 bits long.

camera_b0

Structure/Union member

camera_b1

Structure/Union member

camera_unused

Structure/Union member

ccd_b0

Structure/Union member

ccd_b1

Structure/Union member

ccd_unused

Structure/Union member

extraBits

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetDriverControlParams**

Bases: Structure

ctypes Structure to hold the parameters for the Get Driver Control command, used to query the value of a specific driver control parameter.

controlParameter

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetDriverControlResults**

Bases: Structure

ctypes Structure to hold the result from the Get Driver Control command, used to query the value of a specific driver control parameter

controlValue

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetDriverHandleResults**

Bases: Structure

ctypes Structure to hold the results from the Get Driver Handle command. The handle is the camera ID used when switching control between connected cameras with the Set Driver Handle command.

handle

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetDriverInfoParams**

Bases: Structure

ctypes Structure used to hold the parameters for the Get Driver Info command

request

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetDriverInfoResults0**

Bases: Structure

ctypes Structure used to hold the results from the Get Driver Info command

maxRequest

Structure/Union member

name

Structure/Union member

version

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**GetLinkStatusResults**

Bases: Structure

ctypes Structure to hold the results from the Get Link Status command.

baseAddress

Structure/Union member

cameraType

Structure/Union member

comFailed

Structure/Union member

comTotal

Structure/Union member

linkEstablished

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**OpenDeviceParams**

Bases: Structure

ctypes Structure to hold the parameters for the Open Device command.

deviceType

Structure/Union member

ipAddress

Structure/Union member

lptBaseAddress

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryCommandStatusParams**

Bases: Structure

ctypes Structure to hold the parameters for the Query Command Status command.

command

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryCommandStatusResults**

Bases: Structure

ctypes Structure to hold the results from the Query Command Status command.

status

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryTemperatureStatusParams**

Bases: Structure

ctypes Structure used to hold the parameters for the Query Temperature Status command.

request

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryTemperatureStatusResults**

Bases: Structure

ctypes Structure used to hold the results from the Query Temperature Status command (standard version).

ambientThermistor

Structure/Union member

ccdSetpoint

Structure/Union member

ccdThermistor

Structure/Union member

enabled

Structure/Union member

power

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryTemperatureStatusResults2**

Bases: Structure

ctypes Structure used to hold the results from the Query Temperature Status command (extended version).

ambientTemperature

Structure/Union member

ccdSetpoint

Structure/Union member

coolingEnabled

Structure/Union member

externalTrackingCCDPower

Structure/Union member

externalTrackingCCDTemperature

Structure/Union member

fanEnabled

Structure/Union member

fanPower

Structure/Union member

fanSpeed

Structure/Union member

heatsinkTemperature

Structure/Union member

imagingCCDPower

Structure/Union member

imagingCCDTemperature

Structure/Union member

trackingCCDPower

Structure/Union member

trackingCCDSetpoint

Structure/Union member

trackingCCDTemperature

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryUSBInfo**

Bases: Structure

ctypes (Sub-)Structure used to hold details of individual cameras returned by 'CC_QUERY_USB' command

cameraFound

Structure/Union member

cameraType

Structure/Union member

name

Structure/Union member

serialNumber

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryUSBResults**

Bases: Structure

ctypes Structure used to hold the results from 'CC_QUERY_USB' command (max 4 cameras).

camerasFound

Structure/Union member

usbInfo

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryUSBResults2**

Bases: Structure

ctypes Structure used to hold the results from 'CC_QUERY_USB2' command (max 8 cameras).

camerasFound

Structure/Union member

usbInfo

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**QueryUSBResults3**

Bases: Structure

ctypes Structure used to hold the results from 'CC_QUERY_USB3' command (max 24 cameras).

camerasFound

Structure/Union member

usbInfo

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**ReadoutInfo**

Bases: Structure

ctypes Structure to store details of an individual readout mode. An array of up to 20 of these will be returned as part of the GetCCDInfoResults0 struct when the Get CCD Info command is used with request 'CCD_INFO_IMAGING'.

The gain field is a 4 digit Binary Coded Decimal (yes, really) of the form XX.XX, in units of electrons/ADU.

The pixel_width and pixel_height fields are 6 digit Binary Coded Decimals for the form XXXXXX.XX in units of microns, helpfully supporting pixels up to 1 metre across.

gain

Structure/Union member

height

Structure/Union member

mode

Structure/Union member

pixelHeight

Structure/Union member

pixelWidth

Structure/Union member

width

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**ReadoutLineParams**

Bases: Structure

ctypes Structure to hold the parameters for the Readout Line command.

ccd

Structure/Union member

pixellength

Structure/Union member

pixelStart

Structure/Union member

readoutMode

Structure/Union member

class panoptes.pocs.camera.sbigudrv.**SBIGDriver**(*library_path=None, retries=1, **kwargs*)

Bases: [*AbstractSDKDriver*](#)

cfw_get_info(*handle, model='AUTO'*)

Get info from the colour filter wheel

This will return the usual status information plus the firmware version and the number of filter wheel positions.

Parameters

- **handle** (*int*) – handle of the camera that the filter wheel is connected to.

- **model** (*str*, *optional*) – Model of the filter wheel to control. Default is 'AUTO', which asks the driver to autodetect the model.

Returns

dictionary containing the 'model', 'firmware_version' and 'n_positions' for the filter wheel.

Return type

`dict`

Raises

RuntimeError – raised if the driver returns an error

cfw_goto(*handle*, *position*, *model*='AUTO', *cfw_event*=None, *timeout*=<Quantity 10. s>)

Move colour filter wheel to a given position

This function returns immediately after starting the move but spawns a thread to poll the filter wheel until the move completes (see `_cfw_poll` method for details). This thread will log the result of the move, and optionally set a `threading.Event` to signal that it has completed.

Parameters

- **handle** (*int*) – handle of the camera that the filter wheel is connected to.
- **position** (*int*) – position to move the filter wheel. Must be an integer ≥ 1 .
- **model** (*str*, *optional*) – Model of the filter wheel to control. Default is 'AUTO', which asks the driver to autodetect the model.
- **cfw_event** (*threading.Event*, *optional*) – Event to set once the move is complete
- **timeout** (*u.Quantity*, *optional*) – maximum time to wait for the move to complete. Should be a Quantity with time units. If a numeric type without units is given seconds will be assumed. Default is 10 seconds.

Returns

dictionary containing the 'model', 'position', 'status' and 'error' values returned by the driver.

Return type

`dict`

Raises

RuntimeError – raised if the driver returns an error

cfw_init(*handle*, *model*='AUTO', *timeout*=<Quantity 10. s>)

Initialise colour filter wheel

Sends the initialise command to the colour filter wheel attached to the camera specified with handle. This will generally not be required because all SBIG filter wheels initialise themselves on power up.

Parameters

- **handle** (*int*) – handle of the camera that the filter wheel is connected to.
- **model** (*str*, *optional*) – Model of the filter wheel to control. Default is 'AUTO', which asks the driver to autodetect the model.
- **timeout** (*u.Quantity*, *optional*) – maximum time to wait for the move to complete. Should be a Quantity with time units. If a numeric type without units is given seconds will be assumed. Default is 10 seconds.

Returns

dictionary containing the ‘model’, ‘position’, ‘status’ and ‘error’ values returned by the driver.

Return type

dict

Raises

RuntimeError – raised if the driver returns an error

cfw_query(*handle*, *model*='AUTO')

Query status of the colour filter wheel

This is mostly used to poll the filter wheel status after asking the filter wheel to move in order to find out when the move has completed.

Parameters

- **handle** (*int*) – handle of the camera that the filter wheel is connected to.
- **model** (*str*, *optional*) – Model of the filter wheel to control. Default is ‘AUTO’, which asks the driver to autodetect the model.

Returns

dictionary containing the ‘model’, ‘position’, ‘status’ and ‘error’ values returned by the driver.

Return type

dict

Raises

RuntimeError – raised if the driver returns an error

disable_vdd_optimized(*handle*)

Stops selective lowering of the CCD’s Vdd voltage to ensure consistent bias structures.

There are many driver control parameters, almost all of which we would not want to change from their default values. The one exception is DCP_VDD_OPTIMIZED. From the SBIG manual:

The DCP_VDD_OPTIMIZED parameter defaults to TRUE which lowers the CCD’s Vdd (which reduces amplifier glow) only for images 3 seconds and longer. This was done to increase the image throughput for short exposures as raising and lowering Vdd takes 100s of milliseconds. The lowering and subsequent raising of Vdd delays the image readout slightly which causes short exposures to have a different bias structure than long exposures. Setting this parameter to FALSE stops the short exposure optimization from occurring.

The default behaviour will improve image throughput for exposure times of 3 seconds or less but at the penalty of altering the bias structure between short and long exposures. This could cause systematic errors in bias frames, dark current measurements, etc. It’s probably not worth it.

establish_link()

get_sdk_version(*request_type*='DRIVER_STD')

Get the version of the SDK

get_ccd_info(*handle*)

Use Get CCD Info to gather all relevant info about CCD capabilities. Already have camera type, ‘name’ and serial number, this gets the rest.

get_devices()

Gets currently connected camera inf.

Returns

All currently connected camera serial numbers with corresponding handles.

Return type

`dict`

`get_driver_handle()`

`get_exposure_status(handle)`

Returns the current exposure status of the camera, e.g. 'CS_IDLE', 'CS_INTEGRATING'

`get_link_status()`

`open_device(device_type)`

`open_driver()`

`query_temp_status(handle)`

`readout(handle, readout_mode, top, left, height, width)`

property retries

`set_handle(handle)`

`set_temp_regulation(handle, target_temperature, enabled)`

`start_exposure(handle, seconds, dark, antiblooming, readout_mode, top, left, height, width)`

`class panoptes.pocs.camera.sbigudrv.SetDriverControlParams`

Bases: `Structure`

ctypes Structure to hold the parameters for the Set Driver Control command, used to set the value of a specific driver control parameter

`controlParameter`

Structure/Union member

`controlValue`

Structure/Union member

`class panoptes.pocs.camera.sbigudrv.SetDriverHandleParams`

Bases: `Structure`

ctypes Structure to hold the parameter for the Set Driver Handle command.

`handle`

Structure/Union member

`class panoptes.pocs.camera.sbigudrv.SetTemperatureRegulationParams`

Bases: `Structure`

ctypes Structure used to hold the parameters for the Set Temperature Regulation command.

`ccdSetpoint`

Structure/Union member

`regulation`

Structure/Union member

class panoptes.pocs.camera.sbigudrv.SetTemperatureRegulationParams2

Bases: Structure

ctypes Structure used to hold the parameters for the Set Temperature Regulation 2 command.

ccdSetpoint

Structure/Union member

regulation

Structure/Union member

class panoptes.pocs.camera.sbigudrv.StartExposureParams2

Bases: Structure

ctypes Structure to hold the parameters for the Start Exposure 2 command. (The Start Exposure command is deprecated.)

abgState

Structure/Union member

ccd

Structure/Union member

exposureTime

Structure/Union member

height

Structure/Union member

left

Structure/Union member

openShutter

Structure/Union member

readoutMode

Structure/Union member

top

Structure/Union member

width

Structure/Union member

class panoptes.pocs.camera.sbigudrv.StartReadoutParams

Bases: Structure

ctypes Structure to hold the parameters for the Start Readout command.

ccd

Structure/Union member

height

Structure/Union member

left

Structure/Union member

readoutMode

Structure/Union member

top

Structure/Union member

width

Structure/Union member

panoptes.pocs.camera.sdk module

```
class panoptes.pocs.camera.sdk.AbstractSDKCamera(name='Generic SDK camera', driver=<class  
    'panoptes.pocs.camera.sdk.AbstractSDKDriver'>,  
    library_path=None, filter_type=None,  
    target_temperature=None, *args, **kwargs)
```

Bases: [AbstractCamera](#)

property properties

A collection of camera properties as read from the camera

```
class panoptes.pocs.camera.sdk.AbstractSDKDriver(name, library_path=None, *args, **kwargs)
```

Bases: [PanBase](#)

abstract get_sdk_version()

Get the version of the SDK

abstract get_devices()

Get connected device UIDs and corresponding device nodes/handles/IDs.

property version

panoptes.pocs.camera.zwo module

```
class panoptes.pocs.camera.zwo.Camera(name='ZWO ASI Camera', gain=100, image_type=None, *args,  
    **kwargs)
```

Bases: [AbstractSDKCamera](#)

property bit_depth

ADC bit depth

connect(enable_cooling=False)

Connect to ZWO ASI camera.

Gets 'camera_ID' (needed for all driver commands), camera properties and details of available camera commands/parameters.

property cooling_enabled

Current status of the camera's image sensor cooling system (enabled/disabled)

property cooling_power

Current power level of the camera's image sensor cooling system (as a percentage).

property egain

Image sensor gain in e-/ADU for the current gain, as reported by the camera.

property gain

Current value of the camera's gain setting in internal units.

See *egain* for the corresponding electrons / ADU value.

property image_type

Current camera image type, one of 'RAW8', 'RAW16', 'Y8', 'RGB24'

property is_exposing

True if an exposure is currently under way, otherwise False

start_video(*seconds, filename_root, max_frames, image_type=None*)

stop_video()

property target_temperature

Current value of the target temperature for the camera's image sensor cooling control.

Can be set by assigning an `astropy.units.Quantity`

property temperature

Current temperature of the camera's image sensor

Module contents

`panoptes.pocs.camera.create_cameras_from_config`(*config=None, cameras=None, auto_primary=True, recreate_existing=False, *args, **kwargs*)

Create camera object(s) based on the config.

Creates a camera for each camera item listed in the config. Ensures the appropriate camera module is loaded.

Parameters

- **config** (*dict* or *None*) – A config object for a camera or None to lookup in config-server.
- **cameras** (*list* of *panoptes.pocs.camera.Camera* or *None*) – A list of camera objects or None.
- **auto_primary** (*bool*) – If True, when no camera is marked as the primary camera, the first camera in the list will be used as primary. Default True.
- **recreate_existing** (*bool*) – If True, a camera object will be recreated if an existing camera with the same *uid* is already assigned. Should currently only affect cameras that use the *sdk* (i.g. not DSLRs). Default False raises an exception if camera is already assigned.
- ***args** (*list*) – Passed to *get_config*.
- ****kwargs** (*dict*) – Can pass a *cameras* object that overrides the info in the configuration file. Can also pass *auto_detect* (*bool*) to try and automatically discover the ports. Any other items as passed to *get_config*.

Returns

An ordered dictionary of created camera objects, with the

camera name as key and camera instance as value. Returns an empty `OrderedDict` if there is no camera configuration items.

Return type

`OrderedDict`

Raises

- **error.CameraNotFound** – Raised if camera cannot be found at specified port or if auto_detect=True and no cameras are found.
- **error.PanError** – Description

`panoptes.pocs.camera.get_gphoto2_cmd()`

Finds the gphoto2 command on the system

`panoptes.pocs.camera.list_connected_cameras(endpoint: AnyHttpUrl | None = None)`

Detect connected cameras.

Uses gphoto2 to try and detect which cameras are connected. Cameras should be known and placed in config but this is a useful utility.

Returns

A list of the ports with detected cameras.

Return type

`list`

panoptes.pocs.dome package

Submodules

panoptes.pocs.dome.abstract_serial_dome module

`class panoptes.pocs.dome.abstract_serial_dome.AbstractSerialDome(*args, **kwargs)`

Bases: `AbstractDome`

Abstract base class for controlling a dome via a serial connection.

Takes care of a single thing: configuring the connection to the device.

connect()

Connects to the device via the serial port, if disconnected.

Returns

Returns True if connected, False otherwise.

Return type

`bool`

disconnect()

Disconnect from the dome controller.

Raises

An exception if unable to disconnect. –

property is_connected

True if connected to the hardware or driver.

verify_connected()

Throw an exception if not connected.

panoptes.pocs.dome.astrohaven module

class panoptes.pocs.dome.astrohaven.**AstrohavenDome**(*args, **kwargs)

Bases: [*AbstractSerialDome*](#)

Interface to an Astrohaven clamshell dome with a Vision 130 PLC and RS-232 interface.

Experience shows that it emits a status byte about once a second, with the codes as described in the Protocol class.

LISTEN_TIMEOUT = 3

MOVE_LISTEN_TIMEOUT = 0.1

MOVE_TIMEOUT = 10

NUM_CLOSE_FEEDBACKS = 2

close()

If not known to be closed, attempts to close the dome.

Must already be connected.

Returns: True if and when closed, False if unable to close.

property is_closed

True if dome is known to be closed.

property is_open

True if dome is known to be open.

open()

If not known to be open, attempts to open the dome.

Must already be connected.

Returns: True if and when open, False if unable to open.

property status

Return a dict with dome's current status.

panoptes.pocs.dome.astrohaven.**Dome**

alias of [*AstrohavenDome*](#)

class panoptes.pocs.dome.astrohaven.**Protocol**

Bases: [*object*](#)

A_CLOSE_LIMIT = 'X'

A_IS_CLOSED = '1'

A_OPEN_LIMIT = 'x'

BOTH_CLOSED = '0'

BOTH_OPEN = '3'

B_CLOSE_LIMIT = 'Y'

B_IS_CLOSED = '2'

```
B_OPEN_LIMIT = 'y'
CLOSE_A = 'A'
CLOSE_B = 'B'
CLOSE_BOTH = 'C'
OPEN_A = 'a'
OPEN_B = 'b'
OPEN_BOTH = 'O'
RESET = 'R'
STABLE_STATES = ('0', '3', '2', '1')
```

panoptes.pocs.dome.bisque module

class panoptes.pocs.dome.bisque.Dome(*args, **kwargs)

Bases: [AbstractDome](#)

docstring for Dome

close()

If not known to be closed, attempts to close the dome.

Must already be connected.

Returns: True if and when closed, False if unable to close.

connect()

Establish a connection to the dome controller.

The sub-class implementation can access configuration information from self._config; see PanBase for more common properties.

Returns: True if connected, False otherwise.

disconnect()

Disconnect from the dome controller.

Raises

An exception if unable to disconnect. –

find_home()

property is_closed

True if dome is known to be closed.

property is_connected

True if connected to the hardware or driver.

property is_open

True if dome is known to be open.

property is_parked

open()

If not known to be open, attempts to open the dome.

Must already be connected.

Returns: True if and when open, False if unable to open.

park()**property position****read(timeout=5)****read_slit_state()****property status**

A string representing the status of the dome for presentation.

This string is NOT for use in logic, only for presentation, as there is no requirement to produce the same string for different types of domes: a roll-off roof might have a very different status than a rotating dome that is coordinating its movements with the telescope mount.

Examples: 'Open', 'Closed', 'Opening', 'Closing', 'Left Moving', 'Right Stuck'

Returns: A string.

unpark()**write(value)****panoptes.pocs.dome.protocol_astrohaven_simulator module****panoptes.pocs.dome.simulator module**

class panoptes.pocs.dome.simulator.Dome(*args, **kwargs)

Bases: *AbstractDome*

Simulator for a Dome controller.

close()

If not known to be closed, attempts to close the dome.

Must already be connected.

Returns: True if and when closed, False if unable to close.

connect()

Establish a connection to the dome controller.

The sub-class implementation can access configuration information from self._config; see PanBase for more common properties.

Returns: True if connected, False otherwise.

disconnect()

Disconnect from the dome controller.

Raises

An exception if unable to disconnect. –

property is_closed

True if dome is known to be closed.

property is_open

True if dome is known to be open.

open()

If not known to be open, attempts to open the dome.

Must already be connected.

Returns: True if and when open, False if unable to open.

property status

A string representing the status of the dome for presentation.

This string is NOT for use in logic, only for presentation, as there is no requirement to produce the same string for different types of domes: a roll-off roof might have a very different status than a rotating dome that is coordinating its movements with the telescope mount.

Examples: 'Open', 'Closed', 'Opening', 'Closing', 'Left Moving', 'Right Stuck'

Returns: A string.

Module contents

```
class panoptes.pocs.dome.AbstractDome(*args, **kwargs)
```

Bases: [PanBase](#)

Abstract base class for controlling a non-rotating dome.

This assumes that the observatory 'dome' is not a classic rotating dome with a narrow slit, but instead something like a roll-off roof or clam-shell, which can be observed from when open, and that the other states (closed or moving) are not used for observing.

Adding support for a rotating dome would require coordination during observing to make sure that the opening tracks the field being observed.

abstract close()

If not known to be closed, attempts to close the dome.

Must already be connected.

Returns: True if and when closed, False if unable to close.

abstract connect()

Establish a connection to the dome controller.

The sub-class implementation can access configuration information from self._config; see PanBase for more common properties.

Returns: True if connected, False otherwise.

abstract disconnect()

Disconnect from the dome controller.

Raises

An exception if unable to disconnect. –

abstract is_closed()

True if dome is known to be closed.

property is_connected

True if connected to the hardware or driver.

abstract is_open()

True if dome is known to be open.

abstract open()

If not known to be open, attempts to open the dome.

Must already be connected.

Returns: True if and when open, False if unable to open.

abstract property status

A string representing the status of the dome for presentation.

This string is NOT for use in logic, only for presentation, as there is no requirement to produce the same string for different types of domes: a roll-off roof might have a very different status than a rotating dome that is coordinating its movements with the telescope mount.

Examples: 'Open', 'Closed', 'Opening', 'Closing', 'Left Moving', 'Right Stuck'

Returns: A string.

`panoptes.pocs.dome.create_dome_from_config(*args, **kwargs)`

If there is a dome specified in the config, create a driver for it.

A dome needs a config. We assume that there is at most one dome in the config, i.e. we don't support two different dome devices, such as might be the case if there are multiple independent actuators, for example slit, rotation and vents. Those would need to be handled by a single dome driver class.

`panoptes.pocs.dome.create_dome_simulator(*args, **kwargs)`

panoptes.pocs.filterwheel package

Submodules

panoptes.pocs.filterwheel.filterwheel module

```
class panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel(name='Generic Filter Wheel',
                                                                model='simulator',
                                                                camera=None,
                                                                filter_names=None,
                                                                timeout=None,
                                                                serial_number='XXXXXX',
                                                                dark_position=None,
                                                                focus_offsets=None, *args,
                                                                **kwargs)
```

Bases: [PanBase](#)

Base class for all filter wheels

Parameters

- **name** (*str*, optional) – name of the filter wheel
- **model** (*str*, optional) – model of the filter wheel

- **camera** (*pocs.camera.*.Camera, optional*) – camera that this filter wheel is associated with.
- **filter_names** (*list of str*) – names of the filters installed at each filter wheel position
- **timeout** (*u.Quantity, optional*) – maximum time to wait for a move to complete. Should be a Quantity with time units. If a numeric type without units is given seconds will be assumed. Default is None (no timeout).
- **serial_number** (*str, optional*) – serial number of the filter wheel, default 'XXXXXX'
- **dark_position** (*int or str, optional*) – used to specify either a filter wheel position or a filter name that should be used when taking dark exposures with a camera that is not able to take internal darks.
- **focus_offsets** (*abc.Mapping, optional*) – Dictionary of filter_name: focus offset pairs to apply when moving between filters. If None (default), no offsets are applied.

property camera

Reference to the Camera object that the FilterWheel is assigned to, if any. A filter wheel should only ever be assigned to one or zero Cameras!

abstract connect()

Connect to filter wheel

property current_filter

Name of the filter in the current position

filter_name(position)

Name of the filter in the given integer position.

property filter_names

List of the names of the filters installed in the filter wheel

property is_connected

Is the filterwheel available

abstract property is_moving

Is the filterwheel currently moving

property is_ready**property is_unidirectional****property model**

Model of the filter wheel

move_to(new_position, blocking=False)

Move the filter wheel to the given position.

The position can be expressed either as an integer, or as (part of) one of the names from the filter_names list. To allow filter names of the form '<filter band>_<serial number>' to be selected by band only position can be a substring from the start of one of the names in the filter_names list, provided that this produces only one match.

Parameters

- **new_position** (*int or str*) – position to move to.
- **blocking** (*bool, optional*) – If False (default) return immediately, if True block until the filter wheel move has been completed.

Returns

Event that will be set to signal when the move has completed

Return type

`threading.Event`

Raises

ValueError – if `new_position` is not a valid position specifier for this filterwheel.

Examples

Substring matching is useful when the filter names contain both the type of filter and a serial number, e.g. the following selects a g band filter without having to know its full name.

```
>>> from panoptes.pocs.filterwheel.simulator import FilterWheel
>>> fw = FilterWheel(filter_names=['u_12', 'g_04', 'r_09', 'i_20', 'z_07'])
>>> fw_event = fw.move_to('g')
>>> fw_event.wait()
True
>>> fw.current_filter
'g_04'
```

move_to_dark_position(*blocking=False*)

Move to filterwheel position for taking darks.

move_to_light_position(*blocking=False*)

Return to last filterwheel position from before taking darks.

property n_positions

Number of positions in the filter wheel

property name

Name of the filter wheel

abstract property position

Current integer position of the filter wheel

property uid

A serial number of the filter wheel

panoptes.pocs.filterwheel.libefw module

class `panoptes.pocs.filterwheel.libefw.EFWDriver`(*library_path=None, **kwargs*)

Bases: `AbstractSDKDriver`

calibrate(*filterwheel_ID*)

Calibrate filterwheel with given ID.

close(*filterwheel_ID*)

Close connection to filterwheel with given ID.

get_ID(*filterwheel_index*)

Get integer ID of filterwheel with a given index.

get_sdk_version()

Get the version for the SDK.

get_devices()

Get connected device ‘UIDs’ and corresponding device nodes/handles/IDs.

EFW SDK has no way to access any unique identifier for connected filterwheels. Instead we construct an ID from combination of filterwheel name, number of positions and integer ID. This will probably not be deterministic, in general, and is only guaranteed to be unique between multiple filterwheels on a single computer.

get_direction(*filterwheel_ID*)

Get current unidirectional/bidirectional setting of filterwheel with given ID.

get_num()

Get the count of connected EFW filterwheels.

get_position(*filterwheel_ID*)

Get current position of filterwheel with given ID.

get_product_ids()

Get product IDs of supported(?) EFW filterwheels.

The SDK documentation does not explain what the product IDs returned by this function are, but from experiment and analogy with a similar function in the ASI camera SDK it appears this is a list of the product IDs of the filterwheels that the SDK supports, not the product IDs of the connected filterwheels. There appears to be no way to obtain the product IDs of the connected filterwheel(s).

get_property(*filterwheel_ID*)

Get properties of filterwheel with given ID.

open(*filterwheel_ID*)

Open connection to filterwheel with given ID.

set_direction(*filterwheel_ID*, *unidirectional*)

Set unidirectional/bidirectional for filterwheel with given ID.

set_position(*filterwheel_ID*, *position*, *move_event=None*, *timeout=None*)

Set position of filterwheel with given ID.

This function returns immediately after starting the move but spawns a thread to poll the filter wheel until the move completes (see `_efw_poll` method for details). This thread will log the result of the move, and optionally set a `threading.Event` to signal that it has completed.

Parameters

- **filterwheel_ID** (*int*) – integer ID of the filterwheel that is moving.
- **position** (*int*) – position to move the filter wheel. Must an integer ≥ 0 .
- **move_event** (*threading.Event*, *optional*) – Event to set once the move is complete
- **timeout** (*u.Quantity*, *optional*) – maximum time to wait for the move to complete. Should be a `Quantity` with time units. If a numeric type without units is given seconds will be assumed.

Raises

panoptes.utils.error.PanError – raised if the driver returns an error starting the move.

```
class panoptes.pocs.filterwheel.libefw.EFWInfo
```

Bases: `Structure`

Filterwheel info structure.

id

Structure/Union member

name

Structure/Union member

slot_num

Structure/Union member

```
class panoptes.pocs.filterwheel.libefw.ErrorCode(value, names=None, *, module=None,
                                                  qualname=None, type=None, start=1,
                                                  boundary=None)
```

Bases: `IntEnum`

CLOSED = 9

END = -1

ERROR_STATE = 6

GENERAL_ERROR = 7

INVALID_ID = 2

INVALID_INDEX = 1

INVALID_VALUE = 3

MOVING = 5

NOT_SUPPORTED = 8

REMOVED = 4

SUCCESS = 0

panoptes.pocs.filterwheel.sbig module

```
class panoptes.pocs.filterwheel.sbig.FilterWheel(name='SBIG Filter Wheel', model='sbig',
                                                  camera=None, filter_names=None,
                                                  timeout=<Quantity 10. s>, serial_number=None,
                                                  *args, **kwargs)
```

Bases: `AbstractFilterWheel`

Class for SBIG filter wheels connected to the I2C port of an SBIG camera.

Parameters

- **name** (*str*, optional) – name of the filter wheel
- **model** (*str*, optional) – model of the filter wheel
- **camera** (`pocs.camera.sbig.Camera`) – camera that this filter wheel is associated with.
- **filter_names** (*list of str*) – names of the filters installed at each filter wheel position

- **timeout** (*u.Quantity, optional*) – maximum time to wait for a move to complete. Should be a Quantity with time units. If a numeric type without units is given seconds will be assumed. Default is 10 seconds.
- **serial_number** (*str*) – serial number of the filter wheel

connect()

Connect to filter wheel. Not called by `__init__` because we need the camera to be connected first. The SBIG camera `connect()` method will call this once it's OK to do so.

property firmware_version

Firmware version of the filter wheel

property is_moving

Is the filterwheel currently moving

property is_unidirectional**property position**

Current integer position of the filter wheel

recalibrate()

Reinitialises/recalibrates the filter wheel. It should not be necessary to call this as SBIG filter wheels initialise and calibrate themselves on power up.

panoptes.pocs.filterwheel.simulator module

```
class panoptes.pocs.filterwheel.simulator.FilterWheel(name='Simulated Filter Wheel',
                                                       model='panoptes.pocs.filterwheel.simulator.FilterWheel',
                                                       camera=None, filter_names=None,
                                                       timeout=<Quantity 10. s>,
                                                       serial_number=None, move_time=<Quantity
                                                       1. s>, unidirectional=True, *args, **kwargs)
```

Bases: [*AbstractFilterWheel*](#)

Class for simulated filter wheels.

Parameters

- **name** (*str, optional*) – name of the filter wheel
- **model** (*str, optional*) – model of the filter wheel
- **camera** (*pocs.camera.*.Camera, optional*) – camera that this filter wheel is associated with.
- **filter_names** (*list of str*) – names of the filters installed at each filter wheel position
- **timeout** (*u.Quantity, optional*) – maximum time to wait for a move to complete. Should be a Quantity with time units. If a numeric type without units is given seconds will be assumed. Default is 10 seconds.
- **serial_number** (*str*) – serial number of the filter wheel
- **move_time** (*astropy.units.Quantity, optional*) – time to move the filter wheel by one position, optional, default 1 second.
- **unidirectional** (*bool, optional*) – If True filterwheel will only rotate in one direction, if False filterwheel will move in either to get to the requested position via the shortest path. Default is True.

connect()

Connect to the filter wheel

property is_moving

Is the filterwheel currently moving

property is_unidirectional

property position

Current integer position of the filter wheel

panoptes.pocs.filterwheel.zwo module

```
class panoptes.pocs.filterwheel.zwo.FilterWheel(name='ZWO Filter Wheel', model='zwo',
                                                  camera=None, filter_names=None,
                                                  timeout=<Quantity 10. s>, serial_number=None,
                                                  library_path=None, unidirectional=True,
                                                  device_name=None, initial_filter=None, *args,
                                                  **kwargs)
```

Bases: [AbstractFilterWheel](#)

Class for ZWO filter wheels.

Parameters

- **name** (*str*, *optional*) – name of the filter wheel.
- **model** (*str*, *optional*) – model of the filter wheel.
- **camera** (*pocs.camera.camera.AbstractCamera*) – camera that this filterwheel is associated with.
- **filter_names** (*list of str*) – names of the filters installed at each filterwheel position.
- **timeout** (*u.Quantity*, *optional*) – maximum time to wait for a move to complete. Should be a Quantity with time units. If a numeric type without units is given seconds will be assumed. Default is 10 seconds.
- **serial_number** (*str*) – serial number of the filter wheel.
- **library_path** (*str*, *optional*) – path to the library e.g. `‘/usr/local/lib/libASICamera2.so’`.
- **unidirectional** (*bool*, *optional*) – If True filterwheel will only rotate in one direction, if False filterwheel will move in either to get to the requested position via the shortest path. Default is True in order to improve repeatability.
- **device_name** (*str*, *optional*) – If multiple filterwheels are connected to a single computer ‘device name’ (e.g. `‘EFW_7_0’`) can be used to select the desired one. See docstring of `pocs.filterwheel.libefw.EFWDriver.get_devices()` for details.
- **initial_filter** (*str or int*) – Name of filter (or integer position) to move to when initialising filter.

connect()

Connect to filter wheel.

property is_moving

Is the filterwheel currently moving

property is_unidirectional

property position

Current integer position of the filter wheel

recalibrate()

Reinitialises/recalibrates the filter wheel.

Module contents

panoptes.pocs.focuser package

Submodules

panoptes.pocs.focuser.astromechanics module

class panoptes.pocs.focuser.astromechanics.**Focuser**(*name='Astromechanics Focuser', model='Canon EF-232', vendor_id=1027, product_id=24577, zero_position=-25000, baudrate=38400, *args, **kwargs*)

Bases: [*AbstractSerialFocuser*](#)

Focuser class for control of a Canon DSLR lens via an Astromechanics Engineering Canon EF/EF-S adapter.

Min/max commands do not exist for the astromechanics controller, as well as other commands to get serial numbers and library/hardware versions. However, as they are marked with the decorator `@abstractmethod`, we have to override them.

Astromechanics focuser are currently very slow to respond to position queries. When they do respond, they give the exact position that was requested by the last `move_to` command (i.e. there is no reported position error). We can therefore avoid such queries by storing the current position in memory.

property max_position

Returns position of far limit of focus travel, in encoder units.

property min_position

Returns position of close limit of focus travel, in encoder units.

move_by(*increment*)

Move focuser by a given amount. Does not do any checking of the requested increment but will warn if the lens reports hitting a stop. :param increment: distance to move the focuser, in encoder units. :type increment: int

Returns

focuser position following the move, in encoder units.

Return type

`int`

move_to(*position*)

Moves focuser to a new position. Does not do any checking of the requested position but will warn if the lens reports hitting a stop. :param position: new focuser position, in encoder units. :type position: int

Returns

focuser position following the move, in encoder units.

Return type`int`**property position**

Current encoder position of the focuser

panoptes.pocs.focuser.birger module

```
class panoptes.pocs.focuser.birger.Focuser(name='Birger Focuser', model='Canon EF-232',
                                           max_command_retries=5, baudrate=115200, **kwargs)
```

Bases: *AbstractSerialFocuser*

Focuser class for control of a Canon DSLR lens via a Birger Engineering Canon EF-232 adapter.

```
connect(port, baudrate, **kwargs)
```

Connect to the Birger focuser. :param port: The serial port. :type port: int :param baudrate: The baudrate of the serial device. :type baudrate: int :param **kwargs: Parsed to super().connect

```
property firmware_version
```

Returns the version string of the Birger adaptor library (firmware).

```
property hardware_version
```

Returns the hardware version of the Birger adaptor.

```
property lens_info
```

Return basic lens info (e.g. '400mm,f28' for a 400 mm f/2.8 lens).

```
property max_position
```

Returns position of far limit of focus travel, in encoder units.

```
property min_position
```

Returns position of close limit of focus travel, in encoder units.

```
move_by(increment)
```

Move focuser by a given amount. Does not do any checking of the requested increment but will warn if the lens reports hitting a stop. :param increment: distance to move the focuser, in encoder units. :type increment: int

Returns

focuser position following the move, in encoder units.

Return type`int`

```
move_to(position)
```

Moves focuser to a new position. Does not do any checking of the requested position but will warn if the lens reports hitting a stop. :param position: new focuser position, in encoder units. :type position: int

Returns

focuser position following the move, in encoder units.

Return type`int`

```
property position
```

Returns current focus position in the lens focus encoder units.

panoptes.pocs.focuser.focuser module

```
class panoptes.pocs.focuser.focuser.AbstractFocuser(name='Generic Focuser', model='simulator',
                                                    port=None, camera=None, timeout=5,
                                                    initial_position=None, autofocus_range=None,
                                                    autofocus_step=None,
                                                    autofocus_seconds=None, autofocus_size=None,
                                                    autofocus_keep_files=None,
                                                    autofocus_take_dark=None,
                                                    autofocus_merit_function=None,
                                                    autofocus_merit_function_kwargs=None,
                                                    autofocus_mask_dilations=None,
                                                    autofocus_make_plots=False, *args, **kwargs)
```

Bases: [PanBase](#)

Base class for all focusers.

Parameters

- **name** (*str*, *optional*) – name of the focuser
- **model** (*str*, *optional*) – model of the focuser
- **port** (*str*, *optional*) – port the focuser is connected to, e.g. a device node
- **camera** (*pocs.camera.Camera*, *optional*) – camera that this focuser is associated with.
- **timeout** (*int*, *optional*) – time to wait for response from focuser.
- **initial_position** (*int*, *optional*) – if given the focuser will move to this position following initialisation.
- **autofocus_range** ((*int*, *int*) *optional*) – Coarse & fine focus sweep range, in encoder units
- **autofocus_step** ((*int*, *int*), *optional*) – Coarse & fine focus sweep steps, in encoder units
- **autofocus_seconds** (*scalar*, *optional*) – Exposure time for focus exposures
- **autofocus_size** (*int*, *optional*) – Size of square central region of image to use, default 500 x 500 pixels.
- **autofocus_keep_files** (*bool*, *optional*) – If True will keep all images taken during focusing. If False (default) will delete all except the first and last images from each focus run.
- **autofocus_take_dark** (*bool*, *optional*) – If True will attempt to take a dark frame before the focus run, and use it for dark subtraction and hot pixel masking, default True.
- **autofocus_merit_function** (*str/callable*, *optional*) – Merit function to use as a focus metric, default `vollath_F4`
- **autofocus_merit_function_kwargs** (*dict*, *optional*) – Dictionary of additional keyword arguments for the merit function.
- **autofocus_mask_dilations** (*int*, *optional*) – Number of iterations of dilation to perform on the saturated pixel mask (determine size of masked regions), default 10
- **(bool (autofocus_make_plots))** – Whether to write focus plots to images folder, default False.
- **optional** – Whether to write focus plots to images folder, default False.

autofocus(*seconds=None, focus_range=None, focus_step=None, cutout_size=None, keep_files=None, take_dark=None, merit_function=None, merit_function_kwargs=None, mask_dilations=None, coarse=False, make_plots=None, filter_name=None, blocking=False*)

Focuses the camera using the specified merit function. Optionally performs a coarse focus to find the approximate position of infinity focus, which should be followed by a fine focus before observing.

Parameters

- **seconds** (*scalar, optional*) – Exposure time for focus exposures, if not specified will use value from config.
- **focus_range** (*2-tuple, optional*) – Coarse & fine focus sweep range, in encoder units. Specify to override values from config.
- **focus_step** (*2-tuple, optional*) – Coarse & fine focus sweep steps, in encoder units. Specify to override values from config.
- **cutout_size** (*int, optional*) – Size of square central region of image to use, default 500 x 500 pixels.
- **keep_files** (*bool, optional*) – If True will keep all images taken during focusing. If False (default) will delete all except the first and last images from each focus run.
- **take_dark** (*bool, optional*) – If True will attempt to take a dark frame before the focus run, and use it for dark subtraction and hot pixel masking, default True.
- **merit_function** (*str/callable, optional*) – Merit function to use as a focus metric, default vollath_F4.
- **merit_function_kwargs** (*dict, optional*) – Dictionary of additional keyword arguments for the merit function.
- **mask_dilations** (*int, optional*) – Number of iterations of dilation to perform on the saturated pixel mask (determine size of masked regions), default 10
- **coarse** (*bool, optional*) – Whether to perform a coarse focus, otherwise will perform a fine focus. Default False.
- **make_plots** (*bool, optional*) – Whether to write focus plots to images folder. If not given will fall back on value of *autofocus_make_plots* set on initialisation, and if it wasn't set then will default to False.
- **filter_name** (*str, optional*) – The filter to use for focusing. If not provided, will use last light position.
- **blocking** (*bool, optional*) – Whether to block until autofocus complete, default False.

Returns

Event that will be set when autofocusing is complete

Return type

`threading.Event`

Raises

ValueError – If invalid values are passed for any of the focus parameters.

property autofocus_error

Error message from the most recent autofocus or None, if there was no error.

property camera

Reference to the Camera object that the Focuser is assigned to, if any. A Focuser should only ever be assigned to one or zero Cameras!

property is_connected

Is the focuser available

abstract is_moving()

True if the focuser is currently moving.

property is_ready**abstract max_position()**

Get position of far limit of focus travel, in encoder units

abstract min_position()

Get position of close limit of focus travel, in encoder units

move_by(*increment*)

Move focuser by a given amount.

Parameters

increment (*int*) – distance to move the focuser, in encoder units.

Returns

focuser position following the move, in encoder units.

Return type

int

abstract move_to(*position*)

Move focuser to new encoder position.

Parameters

position (*int*) – new focuser position, in encoder units.

Returns

focuser position following the move, in encoder units.

Return type

int

property position

Current encoder position of the focuser

property uid

A serial number for the focuser

panoptes.pocs.focuser.focuslynx module

```
class panoptes.pocs.focuser.focuslynx.Focuser(port, name='FocusLynx Focuser',  
                                              initial_position=None, focuser_number=1,  
                                              min_position=0, max_position=None, *args, **kwargs)
```

Bases: [*AbstractFocuser*](#)

Focuser class for control of telescope focusers using the Optec FocusLynx focus controller.

This includes the Starlight Instruments Focus Boss II controller, which is “powered by Optec”

Parameters

- **port** (*str*) – device node of the serial port the focuser controller is connected to, e.g. `‘/dev/ttyUSB0’`

- **name** (*str*, *optional*) – default ‘FocusLynx Focuser’
- **initial_position** (*int*, *optional*) – if given the focuser will drive to this encoder position following initialisation.
- **focuser_number** (*int*, *optional*) – for focus controllers that support more than one focuser set this number to specify which focuser should be controlled by this object. Default 1
- **min_position** (*int*, *optional*) – minimum allowed focuser position in encoder units, default 0
- **max_position** (*int*, *optional*) – maximum allowed focuser position in encoder units. If not given the value will be taken from the focuser’s internal config.

Additional positional and keyword arguments are passed to the base class, AbstractFocuser. See that class for a complete list.

connect()

property firmware_version

Firmware version of the focuser controller

halt()

Causes the focuser to immediately stop any movements

property hardware_version

Device type code of the focuser

property is_connected

Checks status of serial port to determine if connected.

property is_moving

True if the focuser is currently moving

property max_position

Position of far limit of focus travel, in encoder units

property min_position

Position of close limit of focus travel, in encoder units

move_by(*increment*, *blocking=True*)

Moves focuser by a given amount.

Parameters

- **increment** (*int*) – distance to move the focuser, in encoder units. New position must be between min_position and max_position.
- **blocking** (*bool*, *optional*) – If True (default) will block until the move is complete, otherwise will return immediately.

Returns

focuser position following the move. If blocking is True this will be the actual focuser position, if False it will be the target position.

Return type

int

move_to(*position*, *blocking=True*)

Moves focuser to a new position.

Parameters

- **position** (*int*) – new focuser position, in encoder units. Must be between `min_position` and `max_position`.
- **blocking** (*bool*, *optional*) – If `True` (default) will block until the move is complete, otherwise will return immediately.

Returns

focuser position following the move. If `blocking` is `True` this will be the actual focuser position, if `False` it will be the target position.

Return type

`int`

property position

Current focus position in encoder units

property temperature

Current temperature of the focuser, in degrees Celsius, as an `astropy.units.Quantity`

property uid

The user set ‘nickname’ of the focuser. Must be $\leq$ 16 characters

panoptes.pocs.focuser.serial module

class `panoptes.pocs.focuser.serial.AbstractSerialFocuser`(*baudrate=None*, *initial_position=None*,
args*, *kwargs*)

Bases: `AbstractFocuser`

connect(**args*, ***kwargs*)

Connect to the serial device. :param **args*: Parsed to `SerialData`. :param ***kwargs*: Parsed to `SerialData`.

property is_connected

True if the focuser serial device is currently connected.

property is_moving

True if the focuser is currently moving.

reconnect()

Close and open serial port and reconnect to focuser.

panoptes.pocs.focuser.simulator module

class `panoptes.pocs.focuser.simulator.Focuser`(*name='Simulated Focuser'*, *port='/dev/ttyFAKE'*, **args*,
***kwargs*)

Bases: `AbstractFocuser`

Simple focuser simulator

connect()

Simulator pretends to connect a focuser and obtain details, current state.

property is_moving

True if the focuser is currently moving.

property max_position

Returns position of far limit of focus travel, in encoder units

property min_position

Returns position of close limit of focus travel, in encoder units

move_by(*increment*)

Move focuser by a given amount

move_to(*position*)

Move focuser to a new encoder position

Module contents

panoptes.pocs.mount package

Subpackages

panoptes.pocs.mount.ioptron package

Submodules

panoptes.pocs.mount.ioptron.base module

class panoptes.pocs.mount.ioptron.base.**Mount**(*location*, *mount_version=None*, **args*, ***kwargs*)

Bases: [*AbstractSerialMount*](#)

Mount class for iOptron mounts.

initialize(*set_rates=True*, *unpark=False*, **arg*, ***kwargs*)

Initialize the connection with the mount and setup for location.

iOptron mounts are initialized by sending the following two commands to the mount:

- MountInfo

If the mount is successfully initialized, the `_setup_location_for_mount` method is also called.

Returns

Returns the value from `self.is_initialized`.

Return type

`bool`

property is_home

Mount home status.

Type

`bool`

park(*ra_direction=None, ra_seconds=None, dec_direction=None, dec_seconds=None, *args, **kwargs*)

Slews to the park position and parks the mount.

This still uses a custom park command because the mount will not allow the Declination axis to move below 0 degrees.

Note: When mount is parked no movement commands will be accepted.

Parameters

- **ra_direction** (*str* or *None*) – The direction to move the RA axis. If not provided (the default), then look at config setting, otherwise ‘west’.
- **ra_seconds** (*str* or *None*) – The number of seconds to move the RA axis at maximum move speed. If not provided (the default), then look at config setting, otherwise 15 seconds.
- **dec_direction** (*str* or *None*) – The direction to move the Declination axis. If not provided (the default), then look at config setting, otherwise ‘north’.
- **dec_seconds** (*str* or *None*) – The number of seconds to move the Declination axis at maximum move speed. If not provided (the default), then look at config setting, otherwise 15 seconds.

Returns

indicating success

Return type

bool

panoptes.pocs.mount.ioptron.cem40 module

class panoptes.pocs.mount.ioptron.cem40.**Mount**(*location, mount_version='0040', *args, **kwargs*)

Bases: *Mount*

search_for_home()

Search for the home position.

This method uses the internal homing pin on the CEM40 mount to return the mount to the home (or zero) position.

set_target_coordinates(*args, **kwargs)

After setting target coordinates, check number of positions.

The CEM40 can determine if there are 0, 1, or 2 possible positions for the given RA/Dec, with the latter being the case for the meridian flip.

panoptes.pocs.mount.ioptron.ieq30pro module

```
class panoptes.pocs.mount.ioptron.ieq30pro.Mount(location, mount_version='0030', *args, **kwargs)
    Bases: Mount
```

Module contents

```
class panoptes.pocs.mount.ioptron.MountGPS(value, names=None, *, module=None, qualname=None,
                                             type=None, start=1, boundary=None)
```

Bases: [IntEnum](#)

EXTRACTED = 2

OFF = 0

ON = 1

```
class panoptes.pocs.mount.ioptron.MountHemisphere(value, names=None, *, module=None,
                                                    qualname=None, type=None, start=1,
                                                    boundary=None)
```

Bases: [IntEnum](#)

NORTHERN = 1

SOUTHERN = 0

```
class panoptes.pocs.mount.ioptron.MountInfo(value, names=None, *, module=None, qualname=None,
                                              type=None, start=1, boundary=None)
```

Bases: [IntEnum](#)

The return type given by the MountInfo command to identify the mount.

CEM120 = 120

CEM120EC = 121

CEM120EC2 = 122

CEM25 = 25

CEM26 = 26

CEM26EC = 27

CEM40 = 40

CEM40EC = 41

CEM60 = 60

CEM60EC = 61

CEM70 = 70

CEM70EC = 71

GEM28 = 28

```
GEM28EC = 29
```

```
GEM45 = 43
```

```
GEM45EC = 44
```

```
iEQ30Pro = 30
```

```
iEQ45Pro = 45
```

```
iEQ45ProAA = 46
```

```
class panoptes.pocs.mount.ioptron.MountMovementSpeed(value, names=None, *, module=None,
                                                         qualname=None, type=None, start=1,
                                                         boundary=None)
```

```
Bases: IntEnum
```

```
SIDEREAL_1 = 1
```

```
SIDEREAL_128 = 6
```

```
SIDEREAL_16 = 4
```

```
SIDEREAL_2 = 2
```

```
SIDEREAL_256 = 7
```

```
SIDEREAL_512 = 8
```

```
SIDEREAL_64 = 5
```

```
SIDEREAL_8 = 3
```

```
SIDEREAL_MAX = 9
```

```
class panoptes.pocs.mount.ioptron.MountState(value, names=None, *, module=None, qualname=None,
                                                type=None, start=1, boundary=None)
```

```
Bases: IntEnum
```

```
AT_HOME = 7
```

```
GUIDING = 3
```

```
MERIDIAN_FLIPPING = 4
```

```
PARKED = 6
```

```
SLEWING = 2
```

```
STOPPED = 0
```

```
TRACKING = 1
```

```
TRACKING_PEC = 5
```

```
UNKNOWN = 8
```

```
class panoptes.pocs.mount.ioptron.MountTimeSource(value, names=None, *, module=None,
                                                      qualname=None, type=None, start=1,
                                                      boundary=None)
```

```
Bases: IntEnum
```

GPS = 3

HAND_CONTROLLER = 2

RS232 = 1

```
class panoptes.pocs.mount.ioptron.MountTrackingState(value, names=None, *, module=None,
                                                    qualname=None, type=None, start=1,
                                                    boundary=None)
```

Bases: `IntEnum`

CUSTOM = 4

KING = 3

LUNAR = 1

SIDEREAL = 0

SOLAR = 2

Submodules

panoptes.pocs.mount.bisque module

```
class panoptes.pocs.mount.bisque.Mount(*args, **kwargs)
```

Bases: `AbstractMount`

property at_mount_park

Mount slewing status.

Type

`bool`

connect()

Connects to the mount via the serial port (*self._port*)

Returns

Returns the `self.is_connected` property which checks the actual serial connection.

Return type

`bool`

disconnect()

```
initialize(unpark=False, *args, **kwargs)
```

Initialize the connection with the mount and setup for location.

If the mount is successfully initialized, the `_setup_location_for_mount` method is also called.

Returns

Returns the value from `self.is_initialized`.

Return type

`bool`

property is_home

Mount home status.

Type

bool

property is_parked

Mount parked status.

Type

bool

property is_slewing

Mount slewing status.

Type

bool

property is_tracking

Mount tracking status.

Type

bool

move_direction(*direction='north', seconds=1.0, arcmin=None, rate=None*)

Move mount in specified *direction* for given amount of *seconds*

park(*timeout=120*)

Slews to the park position and parks the mount.

Note: When mount is parked no movement commands will be accepted.

Returns

indicating success

Return type

bool

query(**args, **kwargs*)

Override the query method to use the command lock.

This is required because TheSkyX cannot handle simultaneous commands. This function will block until the lock is released.

read(*timeout=5*)**set_park_position**()**set_target_coordinates**(*coords*)

Sets the RA and Dec for the mount's current target.

Parameters

coords (*astropy.coordinates.SkyCoord*) – coordinates specifying target location

Returns

Boolean indicating success

Return type

bool

slew_to_home(*blocking=False, timeout=120*)

Slews the mount to the home position.

Note: Home position and Park position are not the same thing

Parameters

- **blocking** (*bool, optional*) – If command should block while slewing to home, default False.
- **timeout** (*int, optional*) – Timeout in seconds, default 120.

Returns

indicating success

Return type

`bool`

slew_to_target(*timeout=120, **kwargs*)

Slews to the current `_target_coordinates`

Returns

indicating success

Return type

`bool`

slew_to_zero(*blocking=False*)

Calls `slew_to_home` in base class. Can be overridden.

unpark()

Unparks the mount. Does not do any movement commands but makes them available again.

Returns

indicating success

Return type

`bool`

write(*value*)

panoptes.pocs.mount.mount module

class `panoptes.pocs.mount.mount.AbstractMount`(*location, commands=None, *args, **kwargs*)

Bases: `PanBase`

Abstract Base class for controlling a mount. This provides the basic functionality for the mounts. Sub-classes should override the `initialize` method for mount-specific issues as well as any helper methods specific mounts might need. See “NotImplemented Methods” section of this module.

Sets the following properties:

- `self.non_sidereal_available` = False
- `self.PEC_available` = False
- `self._is_initialized` = False

Parameters

- **config** (*dict*) – Custom configuration passed to base mount. This is usually read from the main system config.
- **commands** (*dict*) – Commands for the telescope. These are read from a yaml file that maps the mount-specific commands to common commands.
- **location** (*EarthLocation*) – An *astropy.coordinates.EarthLocation* that contains location information.

property at_mount_park

True if mount is at park position.

Type

bool

abstract connect()

Connect to the mount.

correct_tracking(*correction_info*, *axis_timeout=30.0*)

Make tracking adjustment corrections.

Parameters

- **correction_info** (*dict[tuple]*) – Correction info to be applied, see *get_tracking_correction*.
- **axis_timeout** (*float*, *optional*) – Timeout for adjustment in each axis, default 30 seconds.

Raises

error.Timeout – Timeout error.

disconnect()**distance_from_target()**

Get current distance from target

Returns

An angle represeting the current on-sky separation from the target

Return type

u.Angle

get_current_coordinates()

Reads out the current coordinates from the mount.

Note: See *_mount_coord_to_skycoord* and *_skycoord_to_mount_coord* for translation of mount specific coordinates to *astropy.coordinates.SkyCoord*

Returns

astropy.coordinates.SkyCoord

get_ms_offset(*offset*, *axis='ra'*)

Get offset in milliseconds at current speed

Parameters

- **offset** (*astropy.units.Angle*) – Offset in arcseconds
- **axis** (*str*) – The name of the axis to move, default 'ra'.

Returns

Offset in milliseconds at current speed

Return type

astropy.units.Quantity

get_target_coordinates()

Gets the RA and Dec for the mount's current target. This does NOT necessarily reflect the current position of the mount, see *get_current_coordinates*.

Return type

astropy.coordinates.SkyCoord

get_tracking_correction(*offset_info*, *pointing_ha*, *min_tracking_threshold=None*,
max_tracking_threshold=None)

Determine the needed tracking corrections from current position.

This method will determine the direction and number of milliseconds to correct the mount for each axis in order to correct for any tracking drift. The Declination axis correction ('north' or 'south') depends on the movement of the camera box with respect to the pier, which can be determined from the Hour Angle (HA) of the pointing image in the sequence.

Note: Correction values below 50ms will be skipped and values above 99999ms will be clipped.

Parameters

- **offset_info** (*OffsetError*) – A named tuple describing the offset error. See *pocs.images.OffsetError*.
- **pointing_ha** (*float*) – The Hour Angle (HA) of the mount at the beginning of the observation sequence in degrees. This affects the direction of the Dec adjustment.
- **min_tracking_threshold** (*int*, *optional*) – Minimum size of tracking correction allowed in milliseconds. Tracking corrections lower than this are ignored. Default 100ms from *self.min_tracking_threshold*.
- **max_tracking_threshold** (*int*, *optional*) – Maximum size of tracking correction allowed in milliseconds. Tracking corrections higher than this are set to this value. Default 99999ms from *self.max_tracking_threshold*.

Returns

Offset corrections for each axis as needed

```
dict: {  
    # axis: (arcsec, millisecond, direction)  
    'ra': (float, float, str),  
    'dec': (float, float, str),  
}
```

Return type

dict

property has_target

home_and_park(*args, **kwargs)

Convenience method to first slew to the home position and then park.

abstract initialize(*arg, **kwargs)

property is_connected

Checks the serial connection on the mount to determine if connection is open

Type

bool

property is_home

Mount home status.

Type

bool

property is_initialized

Has mount been initialised with connection

Type

bool

property is_parked

Mount parked status.

Type

bool

property is_slewing

Mount slewing status.

Type

bool

property is_tracking

Mount tracking status.

Type

bool

property location

The location details for the mount.

When a new location is set, `\_setup\_location\_for\_mount` is called, which will update the mount with the current location. It is anticipated the mount won't change locations while observing so this should only be done upon mount initialization.

Type

astropy.coordinates.SkyCoord

move_direction(*direction='north', seconds=1.0*)

Move mount in specified *direction* for given amount of *seconds*

property movement_speed

Movement speed when button pressed.

Type

bool

park(*args, **kwargs)

Slews to the park position and parks the mount.

The park position must be set manually first for this method to work.

Most mount subclasses will override this method to provide mount-specific park functionality.

Note: When mount is parked no movement commands will be accepted.

Returns

indicating success

Return type

bool

query(cmd, params=None, **kwargs)

Sends a query to the mount and returns response.

Performs a send and then returns response. Will do a translate on cmd first. This should be the major serial utility for commands. Accepts an additional args that is passed along with the command. Checks for and only accepts one args param.

Parameters

- **cmd** (str) – A command to send to the mount. This should be one of the commands listed in the mount commands yaml file.
- **params** (str, optional) – Params to pass to serial connection

Examples

```
>>> from panoptes.pocs.mount import create_mount_from_config
>>> mount = create_mount_from_config()
>>> mount.query('set_local_time', '101503')
'1'
>>> mount.query('get_local_time')
'101503'
```

Returns

The response from the mount.

Return type

str

Deleted Parameters:

*args: Parameters to be sent with command if required.

abstract read(*args, **kwargs)

search_for_home()

Search for the home position if supported.

set_target_coordinates(*coords*)

Sets the RA and Dec for the mount's current target.

Parameters

coords (*astropy.coordinates.SkyCoord*) – coordinates specifying target location

Returns

Boolean indicating success

Return type

bool

abstract set_tracking_rate(*direction='ra', delta=1.0*)

Sets the tracking rate for the mount

slew_to_coordinates(*coords, *args, **kwargs*)

Slews to given coordinates.

Parameters

coords (*astropy.SkyCoord*) – The coordinates to slew to.

Returns

indicating success

Return type

bool

slew_to_home(*blocking=True, timeout=180*)

Slews the mount to the home position.

Note: Home position and Park position are not the same thing

Parameters

- **blocking** (*bool, optional*) – If command should block while slewing to home, default True.
- **timeout** (*int, optional*) – Maximum time spent slewing to home, default 180 seconds.

Returns

indicating success

Return type

bool

slew_to_target(*blocking=False, timeout=180*)

Slews to the currently assigned target coordinates.

Slews the mount to the coordinates that have been assigned by `~set_target_coordinates`. If no coordinates have been set, do nothing and return *False*, otherwise return response from the mount.

If *blocking=True* then wait for up to *timeout* seconds for the mount to reach the *is_tracking* state. If a timeout occurs, raise a *pocs.error.Timeout* exception.

Parameters

- **blocking** (*bool, optional*) – If command should block while slewing to home, default False.
- **timeout** (*int, optional*) – Maximum time spent slewing to home, default 180 seconds.

Returns

indicating success

Return type

`bool`

slew_to_zero(*blocking=False*)

Calls *slew_to_home* in base class. Can be overridden.

property state

Mount state.

Type

`bool`

property status

property tracking_rate

Mount tracking rate

Type

`bool`

unpark()

Unparks the mount. Does not do any movement commands but makes them available again.

Returns

indicating success

Return type

`bool`

update_status()

Thin-wrapper to call the status property.

abstract write(*cmd*)

panoptes.pocs.mount.serial module

class panoptes.pocs.mount.serial.**AbstractSerialMount**(*location, *args, **kwargs*)

Bases: [*AbstractMount*](#), [*ABC*](#)

connect()

Connects to the mount via the serial port (*self._port*)

Returns

Returns the *self.is_connected* property (`bool`) which checks the actual serial connection.

disconnect()

read(**args*)

Reads from the serial connection.

Returns

Response from mount

Return type

`str`

set_tracking_rate(*direction*='ra', *delta*=0.0)

Set the tracking rate for the mount :param *direction*: Either *ra* or *dec* :type *direction*: str, optional :param *delta*: Offset multiple of sidereal rate, defaults to 0.0 :type *delta*: float, optional

write(*cmd*)

Sends a string command to the mount via the serial port.

First ‘translates’ the message into the form specific mount can understand using the mount configuration yaml file. This method is most often used from within *query* and may become a private method in the future.

Note: This command currently does not support the passing of parameters. See *query* instead.

Parameters

cmd (*str*) – A command to send to the mount. This should be one of the commands listed in the mount commands yaml file.

panoptes.pocs.mount.simulator module

class panoptes.pocs.mount.simulator.**Mount**(*location*, **args*, ***kwargs*)

Bases: *AbstractMount*

Mount class for a simulator. Use this when you don’t actually have a mount attached.

connect()

Connect to the mount.

disconnect()

get_current_coordinates()

Reads out the current coordinates from the mount.

Note: See *_mount_coord_to_skycoord* and *_skycoord_to_mount_coord* for translation of mount specific coordinates to *astropy.coordinates.SkyCoord*

Returns

astropy.coordinates.SkyCoord

get_ms_offset(*offset*, *axis*='ra')

Fake offset in milliseconds

Parameters

offset (*astropy.units.Angle*) – Offset in arcseconds

Returns

Offset in milliseconds at current speed

Return type

astropy.units.Quantity

initialize(*unpark*=False, **arg*, ***kwargs*)

Initialize the connection with the mount and setup for location.

iOptron mounts are initialized by sending the following two commands to the mount:e

- Version
- MountInfo

If the mount is successfully initialized, the `_setup_location_for_mount` method is also called.

Returns

Returns the value from `self._is_initialized`.

Return type

`bool`

move_direction(*direction*='north', *seconds*=1.0)

Move mount in specified *direction* for given amount of *seconds*

park()

Sets the mount to park for simulator

query(*cmd*, *params*=None, ***kwargs*)

Sends a query to the mount and returns response.

Performs a send and then returns response. Will do a translate on *cmd* first. This should be the major serial utility for commands. Accepts an additional args that is passed along with the command. Checks for and only accepts one args param.

Parameters

- **cmd** (*str*) – A command to send to the mount. This should be one of the commands listed in the mount commands yaml file.
- **params** (*str*, *optional*) – Params to pass to serial connection

Examples

```
>>> from panoptes.pocs.mount import create_mount_from_config
>>> mount = create_mount_from_config()
>>> mount.query('set_local_time', '101503')
'1'
>>> mount.query('get_local_time')
'101503'
```

Returns

The response from the mount.

Return type

`str`

Deleted Parameters:

**args*: Parameters to be sent with command if required.

read(**args*)

set_tracking_rate(*direction*='ra', *delta*=0.0)

Sets the tracking rate for the mount

slew_to_home(*blocking=False, timeout=1*)

Slews the mount to the home position.

Note: Home position and Park position are not the same thing

Returns

indicating success

Return type

`bool`

slew_to_target(*slew_delay=0.5, *args, **kwargs*)

Slews to the currently assigned target coordinates.

Slews the mount to the coordinates that have been assigned by `~set_target_coordinates`. If no coordinates have been set, do nothing and return *False*, otherwise return response from the mount.

If *blocking=True* then wait for up to *timeout* seconds for the mount to reach the *is_tracking* state. If a timeout occurs, raise a *pocs.error.Timeout* exception.

Parameters

- **blocking** (`bool`, *optional*) – If command should block while slewing to home, default *False*.
- **timeout** (`int`, *optional*) – Maximum time spent slewing to home, default 180 seconds.

Returns

indicating success

Return type

`bool`

stop_slew(*next_position='is_tracking'*)

unpark()

Unparks the mount. Does not do any movement commands but makes them available again.

Returns

indicating success

Return type

`bool`

write(*cmd*)

Module contents

`panoptes.pocs.mount.create_mount_from_config`(*mount_info=None, earth_location=None, *args, **kwargs*) → *AbstractMount*

Create a mount instance based on the provided config.

Creates an instance of the *AbstractMount* sub-class in the module specified in the config. Specifically, the class must be in a file called `pocs/mount/<DRIVER_NAME>.py`, and the class must be called *Mount*.

Parameters

- **mount_info** – Optional param which overrides the ‘mount’ entry in config if provided. Useful for testing.
- **earth_location** – *astropy.coordinates.EarthLocation* instance, representing the location of the mount on the Earth. If not specified, the config must include the observatory’s location (Latitude, Longitude and Altitude above mean sea level). Useful for testing.
- ***args** – Other positional args will be passed to the concrete class specified in the config.
- ****kwargs** – Other keyword args will be passed to the concrete class specified in the config.

Returns

An instance of the Mount class if the config (or mount_info) is complete. *None* if neither mount_info nor config[‘mount’] is provided.

Raises

error.MountNotFound – Exception raised when mount cannot be created because of incorrect configuration.

```
panoptes.pocs.mount.create_mount_simulator(mount_info=None, earth_location=None,  
                                             db_type='memory', *args, **kwargs)
```

panoptes.pocs.scheduler package

Subpackages

panoptes.pocs.scheduler.observation package

Submodules

panoptes.pocs.scheduler.observation.base module

```
class panoptes.pocs.scheduler.observation.base.Exposure(image_id: str, path: pathlib.Path,  
                                                         metadata: dict, is_primary: bool = False)
```

Bases: `object`

image_id: `str`

is_primary: `bool` = `False`

metadata: `dict`

path: `Path`

```
class panoptes.pocs.scheduler.observation.base.Observation(field, exptime=<Quantity 120. s>,  
                                                           min_nexp=60, exp_set_size=10,  
                                                           priority=100, filter_name=None,  
                                                           dark=False, *args, **kwargs)
```

Bases: `PanBase`

add_to_exposure_list(cam_name: str, exposure: `Exposure`)

Add the exposure to the list and mark as most recent

property `current_exp_num`: `int`

Return the current number of exposures.

Returns the maximum size of the exposure list from each camera.

Returns

The size of `self.exposure_list`.

Return type

`int`

property `directory`: `Path`

Return the directory for this Observation.

This return the base directory for the Observation. This does *not* include the subfolders for each of the cameras.

Returns

Full path to base directory.

Return type

`str`

property `exptime`

property `exptimes`

Exposure time as a list.

property `first_exposure`: `List[Dict[str, Exposure]] | None`

Return the first exposure information.

Returns

`image_id` and full path of the first exposure from the primary camera.

Return type

`tuple`

classmethod `from_dict`(`observation_config`: `Dict`, `field_class`=`'panoptes.pocs.scheduler.field.Field'`,
`observation_class`=`'panoptes.pocs.scheduler.observation.base.Observation'`)

Creates an *Observation* object from config dict.

Parameters

- **observation_config** (`dict`) – Configuration for *Field* and *Observation*.
- **field_class** (`str`, *optional*) – The full name of the python class to be used as default for the observation's *Field*. This can be overridden by specifying the “type” item under the `observation_config`'s “field” key. Default: `panoptes.pocs.scheduler.field.Field`.
- **observation_class** (`str`, *optional*) – The full name of the python class to be used as default for the observation object. This can be overridden by specifying the “type” item under the `observation_config`'s “observation” key. Default: `panoptes.pocs.scheduler.observation.base.Observation`.

get_exposure(`number`: `int` = 0) → `List[Dict[str, Exposure]] | None`

Returns the given exposure number.

property `last_exposure`: `List[Dict[str, Exposure]] | None`

Return the latest exposure information.

Returns

`image_id` and full path of most recent exposure from the primary camera

Return type`tuple`**property minimum_duration**

Minimum amount of time to complete the observation

property name

Name of the `~pocs.scheduler.field.Field` associated with the observation.

property pointing_image

Return the last pointing image.

Returns

image_id and full path of most recent pointing image from the primary camera.

Return type`tuple`**reset()**

Resets the exposure information for the observation

property seq_time

The time at which the observation was selected by the scheduler.

This is used for path name construction.

property set_duration

Amount of time per set of exposures.

property set_is_finished

Check if the current observing block has finished, which is True when the minimum number of exposures have been obtained and and integer number of sets have been completed. :returns: True if finished, False if not. :rtype: bool

property status: `Dict`

Observation status.

Returns

Dictionary containing current status of observation.

Return type`dict`**to_dict()**

Serialize the object to a dict.

panoptes.pocs.scheduler.observation.bias module

```
class panoptes.pocs.scheduler.observation.bias.BiasObservation(position, min_nexp=None,
                                                                exp_set_size=None)
```

Bases: `Observation`

panoptes.pocs.scheduler.observation.compound module

class panoptes.pocs.scheduler.observation.compound.**Observation**(*args, **kwargs)

Bases: *Observation*

An observation that consists of different combinations of exptimes.

property exptime

Return current exposure time as a u.Quantity.

property exptimes

Exposure time as a list.

classmethod from_dict(*args, **kwargs)

Creates an *Observation* object from config dict.

panoptes.pocs.scheduler.observation.dark module

class panoptes.pocs.scheduler.observation.dark.**DarkObservation**(position, exptimes=None)

Bases: *Observation*

property exptime

Return current exposure time as a u.Quantity.

Module contents

Submodules

panoptes.pocs.scheduler.constraint module

class panoptes.pocs.scheduler.constraint.**AlreadyVisited**(*args, **kwargs)

Bases: *BaseConstraint*

Simple Already Visited Constraint

A simple already visited constraint that determines if the given *observation* has already been visited before. If given *observation* has already been visited then it will not be considered for a call to become the *current observation*.

get_score(time, observer, observation, **kwargs)

class panoptes.pocs.scheduler.constraint.**Altitude**(horizon=None, obstructions=None, *args, **kwargs)

Bases: *BaseConstraint*

Implements altitude constraints for a horizon

get_score(time, observer, observation, **kwargs)

class panoptes.pocs.scheduler.constraint.**BaseConstraint**(weight=1.0, default_score=0.0, *args, **kwargs)

Bases: *PanBase*

get_score(time, observer, target, **kwargs)

```
class panoptes.pocs.scheduler.constraint.Duration(horizon=None, *args, **kwargs)
```

Bases: [BaseConstraint](#)

```
get_score(time, observer, observation, **kwargs)
```

```
class panoptes.pocs.scheduler.constraint.MoonAvoidance(separation=<Quantity 15. deg>, *args,
                                                         **kwargs)
```

Bases: [BaseConstraint](#)

```
get_score(time, observer, observation, **kwargs)
```

panoptes.pocs.scheduler.dispatch module

```
class panoptes.pocs.scheduler.dispatch.Scheduler(*args, **kwargs)
```

Bases: [BaseScheduler](#)

```
get_observation(time=None, show_all=False, constraints=None, read_file=False)
```

Get a valid observation.

Parameters

- **time** (*astropy.time.Time*, optional) – Time at which scheduler applies, defaults to time called
- **constraints** (*list of panoptes.pocs.scheduler.constraint.Constraint*, optional) – The constraints to check. If *None* (the default), use the *scheduler.constraints*.
- **show_all** (*bool*, optional) – Return all valid observations along with merit value, defaults to *False* to only get top value
- **constraints** – The constraints to check. If *None* (the default), use the *scheduler.constraints*
- **read_file** (*bool*, optional) – If the fields file should be reread before scheduling occurs, defaults to *False*.

Returns

A tuple (or list of tuples) with name and score of ranked observations

Return type

tuple or *list*

panoptes.pocs.scheduler.field module

```
class panoptes.pocs.scheduler.field.Field(name, position, equinox='J2000', *args, **kwargs)
```

Bases: *FixedTarget*, [PanBase](#)

property *field_name*

Flattened field name appropriate for paths

```
classmethod from_altaz(name, alt, az, location, time=None, *args, **kwargs)
```

Create a Field from AltAz coords, a location, and optional time.

panoptes.pocs.scheduler.scheduler module

class panoptes.pocs.scheduler.scheduler.**BaseScheduler**(*observer*, *fields_list=None*, *fields_file=None*, *constraints=None*, *args, **kwargs)

Bases: [PanBase](#)

add_observation(*observation_config*, **kwargs)

Adds an *Observation* to the scheduler.

Parameters

observation_config (*dict*) – Configuration dict for *Field* and *Observation*.

clear_available_observations()

Reset the list of available observations

property current_observation

The observation that is currently selected by the scheduler

Upon setting a new observation the *seq_time* is set to the current time and added to the *observed_list*. An old observation is reset (so that it can be used again - see *~pocs.scheduler.observation.reset*). If the new observation is the same as the old observation, nothing is done. The new observation can also be set to *None* to specify there is no current observation.

property fields_file

Field configuration file

A YAML list of config items, specifying a minimum of *name* and *position* for the *~pocs.scheduler.field.Field*. `Observation`s will be built from the list of fields.

A file will be read by *~pocs.scheduler.priority.read_field_list* upon being set.

Note: Setting a new *fields_file* will clear all existing fields

property fields_list

List of field configuration items

A YAML list of config items, specifying a minimum of *name* and *position* for the *~pocs.scheduler.field.Field*. `Observation`s will be built from the list of fields.

A file will be read by *~pocs.scheduler.priority.read_field_list* upon being set.

Note: Setting a new *fields_list* will clear all existing fields

abstract get_observation(*args, **kwargs)

Get a valid observation.

property has_valid_observations

observation_available(*observation*, *time*)

Check if observation is available at given time

Parameters

- **observation** (*pocs.scheduler.observation*) – An *Observation* object
- **time** (*astropy.time.Time*) – The time at which to check observation

property observations

Returns a dict of `~pocs.scheduler.observation.Observation` objects with `~pocs.scheduler.observation.Observation.field.field_name` as the key

Note: `read_field_list` is called if list is None

read_field_list()

Reads the field file and creates valid *Observations*.

remove_observation(field_name)

Removes an *Observation* from the scheduler

Parameters

field_name (*str*) – Field name corresponding to entry key in *observations*

reset_observed_list()

Reset the observed list

set_common_properties(time)

Sets some properties common to all observations, such as end of night, moon, etc.

property status**Module contents**

`panoptes.pocs.scheduler.create_constraints_from_config(config=None) → List[BaseConstraint]`

`panoptes.pocs.scheduler.create_scheduler_from_config(config=None, observer=None, iers_url=None, *args, **kwargs) → BaseScheduler | None`

Sets up the scheduler that will be used by the observatory

panoptes.pocs.sensor package**Submodules****panoptes.pocs.sensor.power module****panoptes.pocs.sensor.remote module**

`class panoptes.pocs.sensor.remote.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.pocs.sensor.weather module

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.pointing.on_enter(event_data)`

Pointing State

Take 30 second exposure and plate-solve to get the pointing error

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

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='panoptes')

Loads the state table :param state_table_name: Name of state table. Corresponds to filename in

`$POCS/conf_files/state_table/` directory or to absolute path if starts with “/”. Default ‘panoptes.yaml’.

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.

property next_state**run(exit_when_done=False, run_once=False, park_when_done=True, initial_next_state='ready')**

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)
- **park_when_done** (*bool*, *optional*) – If True (the default), park the mount when loop completes (i.e. when *keep_running* is False).
- **run_once** (*bool*, *optional*) – If the machine loop should only run one time, if False (the default) loop continuously.
- **initial_next_state** (*str*, *optional*) – The first state the machine should move to from the *sleeping* state, default *ready*.

stop_states()

Stops the machine loop on the next iteration by setting *do_states*=False

Module contents

panoptes.pocs.utils package

Subpackages

panoptes.pocs.utils.cli package

Submodules

panoptes.pocs.utils.cli.camera module

`panoptes.pocs.utils.cli.camera.take_pictures`(*num_images*: *int* = 1, *exptime*: *float* = 1.0, *output_dir*: *Path* = '/home/panoptes/images', *delay*: *float* = 0.0)

Takes pictures with cameras.

panoptes.pocs.utils.cli.config module

class `panoptes.pocs.utils.cli.config.HostInfo`(*, *host*: *str* = '127.0.0.1', *port*: *int* = 6563, *verbose*: *bool* = False)

Bases: `BaseModel`

Metadata for the Config Server

host: *str*

port: *int*

property *url*

verbose: *bool*

`panoptes.pocs.utils.cli.config.get_value`(*key*: *str* | *None* = <*typer.models.ArgumentInfo* object>, *parse*: *bool* = <*typer.models.OptionInfo* object>)

Get an item from the config

`panoptes.pocs.utils.cli.config.main`(*context*: *Context*)

`panoptes.pocs.utils.cli.config.restart`()

Restart the config server process via supervisorctl

`panoptes.pocs.utils.cli.config.server_running`()

Check if the config server is running

`panoptes.pocs.utils.cli.config.set_value`(*key*: *str* = <*typer.models.ArgumentInfo* object>, *value*: *str* = <*typer.models.ArgumentInfo* object>)

Get an item from the config

`panoptes.pocs.utils.cli.config.setup`()

Do initial setup of the config server

`panoptes.pocs.utils.cli.config.status`()

panoptes.pocs.utils.cli.main module

panoptes.pocs.utils.cli.mount module

`panoptes.pocs.utils.cli.mount.park_mount(confirm: bool = False)`

Parks the mount.

Warning: This will move the mount to the park position but will not do any safety checking. Please make sure the mount is safe to park before running this command.

`panoptes.pocs.utils.cli.mount.search_for_home(confirm: bool = False)`

Searches for the mount home position.

Warning: This will move the mount to the home position but will not do any safety checking. Please make sure the mount is safe to move before running this command.

`panoptes.pocs.utils.cli.mount.set_park_position(confirm: bool = False)`

Sets the park position.

Warning: This will move the mount to the park position but will not do any safety checking. Please make sure the mount is safe to move before running this command.

`panoptes.pocs.utils.cli.mount.setup_mount(confirm: bool = False)`

Sets up the mount port, type, and firmware.

panoptes.pocs.utils.cli.network module

panoptes.pocs.utils.cli.notebook module

`panoptes.pocs.utils.cli.notebook.check_for_jupyter()`

Check if Jupyter is installed

`panoptes.pocs.utils.cli.notebook.restart()`

Restart the jupyter server process via supervisorctl

`panoptes.pocs.utils.cli.notebook.set_password(environment: str = <typer.models.OptionInfo object>)`

Set a password for the notebook server

`panoptes.pocs.utils.cli.notebook.start(environment: str = <typer.models.OptionInfo object>, public: bool = <typer.models.OptionInfo object>, port: int = <typer.models.OptionInfo object>, notebook_dir: ~pathlib.Path = <typer.models.OptionInfo object>)`

Start a Jupyter notebook server

panoptes.pocs.utils.cli.power module

panoptes.pocs.utils.cli.run module

panoptes.pocs.utils.cli.sensor module

`panoptes.pocs.utils.cli.sensor.main(context: Context)`

```
panoptes.pocs.utils.cli.sensor.monitor(sensor_name: str, endpoint: str | None =  
                                         <typer.models.OptionInfo object>, store: bool =  
                                         <typer.models.OptionInfo object>, read_frequency: int =  
                                         <typer.models.OptionInfo object>, verbose: bool = False)
```

Continuously read remote sensor, optionally storing results.

panoptes.pocs.utils.cli.weather module

```
panoptes.pocs.utils.cli.weather.config(page='config', base_url='http://localhost:6566')
```

Get the configuration of the weather station.

```
panoptes.pocs.utils.cli.weather.get_page(page, base_url)
```

```
panoptes.pocs.utils.cli.weather.restart(service: str = 'pocs-weather-reader')
```

Restart the weather station service via supervisorctl

```
panoptes.pocs.utils.cli.weather.status(page='status', base_url='http://localhost:6566')
```

Get the status of the weather station.

Module contents

panoptes.pocs.utils.service package

Submodules

panoptes.pocs.utils.service.power module

panoptes.pocs.utils.service.weather module

Module contents

Submodules

panoptes.pocs.utils.cloud module

panoptes.pocs.utils.database module

```
class panoptes.pocs.utils.database.PocsDB(*args, **kwargs)
```

Bases: `PanFileDB`

PanFileDB wrapper that also writes to Firestore.

```
check_firestore()
```

Check if we should use firestore or not.

```
insert(collection, obj)
```

Insert document into local file and firestore db.

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

Inserts into the current collection locally and on firestore db.

panoptes.pocs.utils.error module

exception panoptes.pocs.utils.error.**AboveMaxExptime**(*msg*='Exposure time is too high for camera.',
**kwargs)

Bases: [PocsError](#)

An invalid exptime for a camera, too high.

exception panoptes.pocs.utils.error.**BelowMinExptime**(*msg*='Exposure time is too low for camera.',
**kwargs)

Bases: [PocsError](#)

An invalid exptime for a camera, too low.

exception panoptes.pocs.utils.error.**CameraBusy**(*msg*='Camera busy.', **kwargs)

Bases: [PocsError](#)

A camera is already busy.

exception panoptes.pocs.utils.error.**ImageSaturated**(*msg*='Image is saturated', **kwargs)

Bases: [PocsError](#)

An image is saturated.

exception panoptes.pocs.utils.error.**NotSafeError**(*msg*='Not safe', **kwargs)

Bases: [PanError](#)

Error for when safety fails.

exception panoptes.pocs.utils.error.**NotTwilightError**(*msg*='Not twilight', **kwargs)

Bases: [PanError](#)

Error for when taking twilight flats and not twilight.

exception panoptes.pocs.utils.error.**PocsError**(*msg*='Problem with POCS', **kwargs)

Bases: [PanError](#)

Error for a POCS level exception

panoptes.pocs.utils.location module

class panoptes.pocs.utils.location.**SiteDetails**(*observer*: *astroplan.observer.Observer*,
 earth_location:
 astropy.coordinates.earth.EarthLocation, *location*:
 dict)

Bases: [object](#)

earth_location: [EarthLocation](#)

location: [dict](#)

observer: [Observer](#)

panoptes.pocs.utils.location.**create_location_from_config**() → [SiteDetails](#)

Sets up the site and location details.

These items are read from the 'site' config directive and include:

- name

- latitude
- longitude
- timezone
- pressure
- elevation
- horizon

`panoptes.pocs.utils.location.download_iers_a_file(iers_url: str = None)`

Download the IERS A file.

This will download the IERS from the PANOPTES mirror and then set the auto download to False.

panoptes.pocs.utils.logger module

class `panoptes.pocs.utils.logger.PanLogger`

Bases: `object`

Custom formatter to have dynamic widths for logging.

Also provides a *handlers* dictionary to track attached handlers by id.

See <https://loguru.readthedocs.io/en/stable/resources/recipes.html#dynamically-formatting-messages-to-properly-align-values-with-padding>

format(*record*)

```
panoptes.pocs.utils.logger.get_logger(console_log_file='panoptes.log',
                                       full_log_file='panoptes_{time:YYYYMMDD!UTC}.log',
                                       serialize_full_log=False, log_dir=None,
                                       console_log_level='DEBUG', stderr_log_level='INFO',
                                       cloud_logging_level=None)
```

Creates a root logger for PANOPTES used by the PanBase object.

Two log files are created, one suitable for viewing on the console (via *tail*) and a full log file suitable for archive and later inspection. The full log file is serialized into JSON.

Note: This clobbers all existing loggers and forces the two files.

Parameters

- **console_log_file** (*str/None, optional*) – Filename for the file that is suitable for tailing in a shell (i.e., read by humans). This file is rotated daily however the files are not retained.
- **full_log_file** (*str/None, optional*) – Filename for log file that includes all levels and is serialized and rotated automatically. Useful for uploading to log service website. Defaults to *panoptes_{time:YYYYMMDD!UTC}.log.gz* with a daily rotation at 11:30am and a 7 day retention policy. If *None* then no file will be generated.
- **serialize_full_log** (*bool, optional*) – If the full log should be written as json for log analysis, default False.
- **log_dir** (*str/None, optional*) – The directory to place the log file, default local *logs*.
- **stderr_log_level** (*str, optional*) – The log level to show on stderr, default INFO.

- **console_log_level** (*str*, *optional*) – Log level for console file output, defaults to ‘SUCCESS’. Note that it should be a string that matches standard *logging* levels and also includes *TRACE* (below *DEBUG*) and *SUCCESS* (above *INFO*). Also note this is not the *stderr* output, but the output to the file to be tailed.
- **cloud_logging_level** (*bool*/*None*, *optional*) – If a valid log level is specified, send logs to cloud at that level. If *None* (the default) don’t send logs to the cloud.

Returns

A configured instance of the logger.

Return type

loguru.logger

panoptes.pocs.utils.plotting module

panoptes.pocs.utils.plotting.**make_autofocus_plot**(*output_path*, *initial_thumbnail*, *final_thumbnail*, *initial_focus*, *final_focus*, *focus_positions*, *metrics*, *merit_function*, *line_fit*=*None*, *plot_title*='Autofocus Plot', *plot_width*=9, *plot_height*=18)

Make autofocus plots.

This will make three plots, the top and bottom plots showing the initial and final thumbnail, respectively. The middle plot will contain the scatter plot for the *metrics* for the given *focus_positions*.

Parameters

- **output_path** (*str*) – Path for saving plot.
- **initial_thumbnail** (*np.array*) – The data for the initial thumbnail.
- **final_thumbnail** (*np.array*) – The data for the final thumbnail.
- **initial_focus** (*int*) – The initial focus position.
- **final_focus** (*int*) – The final focus position.
- **focus_positions** (*np.array*) – An array of *int* corresponding the focus positions.
- **metrics** (*np.array*) – An array of *float* corresponding to the measured metrics.
- **merit_function** (*str*) – The name of the merit function used to produce the metrics.
- **line_fit** (*tuple*(*np.array*, *np.array*)) – A tuple for the fitted line. The first entry should be an array of *int* used to calculate fit, the second entry should be an array of the fitted values.
- **plot_title** (*str*) – Title to use for plot
- **plot_width** (*int*) – The plot width in inches.
- **plot_height** (*int*) – The plot height in inches.

Returns

Full path the saved plot.

Return type

str

panoptes.pocs.utils.theskyx module

```
class panoptes.pocs.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

property is_connected

read(timeout=5)

write(value)

Module contents

Submodules

panoptes.pocs.base module

```
class panoptes.pocs.base.PanBase(config_host=None, config_port=None, *args, **kwargs)
```

Bases: `object`

Base class for other classes within the PANOPTES ecosystem

Defines common properties for each class (e.g. logger, config, db).

get_config(*args, **kwargs)

Thin-wrapper around client based get_config that sets default port.

See *panoptes.utils.config.client.get_config* for more information.

Parameters

- ***args** – Passed to get_config
- ****kwargs** – Passed to get_config

set_config(key, new_value, *args, **kwargs)

Thin-wrapper around client based set_config that sets default port.

See *panoptes.utils.config.client.set_config* for more information.

Parameters

- **key** (*str*) – The key name to use, can be namespaced with dots.
- **new_value** (*any*) – The value to store.
- ***args** – Passed to set_config
- ****kwargs** – Passed to set_config

panoptes.pocs.core module

panoptes.pocs.hardware module

Information about hardware supported by Panoptes.

```
class panoptes.pocs.hardware.HardwareName(value, names=None, *, module=None, qualname=None,
                                           type=None, start=1, boundary=None)
```

Bases: `Enum`

`camera = 'camera'`

`dome = 'dome'`

`mount = 'mount'`

`night = 'night'`

`power = 'power'`

`sensors = 'sensors'`

`theskyx = 'theskyx'`

`weather = 'weather'`

```
panoptes.pocs.hardware.get_all_names(all_names=None, without=None)
```

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.

```
>>> from panoptes.pocs.hardware import get_all_names
>>> get_all_names()
['camera', 'dome', 'mount', 'night', 'power', 'sensors', 'theskyx', 'weather']
>>> get_all_names(without='mount') # Single item
['camera', 'dome', 'night', 'power', 'sensors', 'theskyx', 'weather']
>>> get_all_names(without=['mount', 'power']) # List
['camera', 'dome', 'night', 'sensors', 'theskyx', 'weather']
```

```
>>> # You can alter available hardware if needed.
>>> get_all_names(['foo', 'bar', 'power'], without=['power'])
['bar', 'foo']
```

Parameters

- **all_names** (*list*) – The list of hardware.
- **without** (*iterable*) – Return all items except those in the list.

Returns

The sorted list of available hardware except those listed in *without*.

Return type

`list`

`panoptes.pocs.hardware.get_simulator_names(simulator=None, kwargs=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 function 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.

```
>>> from panoptes.pocs.hardware import get_simulator_names
>>> get_simulator_names()
[]
>>> get_simulator_names('all')
['camera', 'dome', 'mount', 'night', 'power', 'sensors', 'theskyx', 'weather']
```

Parameters

- **simulator** (*list*) – 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'.

Returns

List of names of the hardware to be simulated.

panoptes.pocs.images module

class `panoptes.pocs.images.Image(fits_file: Path, wcs_file=None, location=None, *args, **kwargs)`

Bases: [PanBase](#)

compute_offset(*ref_image*)

get_header_pointing()

Get the pointing information from the header

The header should contain the *RA-MNT* and *DEC-MNT* keywords, from which the header pointing coordinates are built.

get_wcs_pointing()

Get the pointing information from the WCS

Builds the pointing coordinates from the plate-solved WCS. These will be compared with the coordinates stored in the header.

property pointing_error

Pointing error namedtuple (delta_ra, delta_dec, magnitude)

Returns pointing error information. The first time this is accessed this will solve the field if not previously solved.

Returns

Pointing error information

Return type

namedtuple

solve_field(radius=15, **kwargs)

Solve field and populate WCS information.

Parameters

- **radius** (*scalar*) – The radius (in degrees) to search near RA-Dec. Defaults to 15°.
- ****kwargs** – Options to be passed to *get_solve_field*.

property wcs_file

WCS file name

When setting the WCS file name, the WCS information will be read, setting the *wcs* property.

class panoptes.pocs.images.**OffsetError**(delta_ra, delta_dec, magnitude)

Bases: `tuple`

delta_dec

Alias for field number 1

delta_ra

Alias for field number 0

magnitude

Alias for field number 2

panoptes.pocs.observatory module**Module contents**

4.2 Contributors

- Wilfred Tyler Gee wtylgergee@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](mailto:jermainegug32515601+jermainegug@users.noreply.github.com)
- [blackflip14 cdkrogers@yahoo.com](mailto:blackflip14cdkrogers@yahoo.com)
- [danjampro danjampro@sky.com](mailto:danjampro@sky.com)
- [Sushant Mehta mehtasushant05@gmail.com](mailto:SushantMehta@mehtasushant05@gmail.com)
- [kmeagle1515 46345142+kmeagle1515@users.noreply.github.com](mailto:kmeagle151546345142+kmeagle1515@users.noreply.github.com)
- [Dan Proole danjampro@sky.com](mailto:DanProole@danjampro@sky.com)
- [Jenny Tong mimming@google.com](mailto:JennyTong@mimming@google.com)
- [Kate Storey-Fisher ksf@google.com](mailto:KateStorey-Fisher@ksf@google.com)
- [Lee Spitler lee.spitler@mq.edu.au](mailto:LeeSpitler@lee.spitler@mq.edu.au)
- [Luca jeremylan@users.noreply.github.com](mailto:LucaJeremylan@users.noreply.github.com)
- [Sean Marquez capsulecorplab@gmail.com](mailto:SeanMarquez@capsulecorplab@gmail.com)
- [lucasholucasho lucasho2340@gmail.com](mailto:lucasholucasho@lucasho2340@gmail.com)
- [megwill4268 megan.will@ymail.com](mailto:megwill4268@megan.will@ymail.com)

4.3 License

The MIT License (MIT)

Copyright (c) 2014-2020 Project PANOPTES

Copyright 2016 Google Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

INDICES AND TABLES

- `modindex`
- `search`
- `contributing`
- [Changelog](#)
- `genindex`

PYTHON MODULE INDEX

p

panoptes, 9
panoptes.pocs, 107
panoptes.pocs.base, 104
panoptes.pocs.camera, 51
panoptes.pocs.camera.camera, 13
panoptes.pocs.camera.fli, 21
panoptes.pocs.camera.gphoto, 12
panoptes.pocs.camera.gphoto.base, 9
panoptes.pocs.camera.gphoto.canon, 11
panoptes.pocs.camera.gphoto.remote, 11
panoptes.pocs.camera.libasi, 22
panoptes.pocs.camera.libfli, 30
panoptes.pocs.camera.libfliconstants, 35
panoptes.pocs.camera.sbig, 35
panoptes.pocs.camera.sbigudrv, 35
panoptes.pocs.camera.sdk, 50
panoptes.pocs.camera.simulator, 13
panoptes.pocs.camera.simulator.ccd, 12
panoptes.pocs.camera.simulator.dslr, 13
panoptes.pocs.camera.zwo, 50
panoptes.pocs.dome, 56
panoptes.pocs.dome.abstract_serial_dome, 52
panoptes.pocs.dome.astrohaven, 53
panoptes.pocs.dome.bisque, 54
panoptes.pocs.dome.simulator, 55
panoptes.pocs.filterwheel, 64
panoptes.pocs.filterwheel.filterwheel, 57
panoptes.pocs.filterwheel.libefw, 59
panoptes.pocs.filterwheel.sbig, 61
panoptes.pocs.filterwheel.simulator, 62
panoptes.pocs.filterwheel.zwo, 63
panoptes.pocs.focuser, 71
panoptes.pocs.focuser.astromechanics, 64
panoptes.pocs.focuser.birger, 65
panoptes.pocs.focuser.focuser, 66
panoptes.pocs.focuser.focuslynx, 68
panoptes.pocs.focuser.serial, 70
panoptes.pocs.focuser.simulator, 70
panoptes.pocs.hardware, 105
panoptes.pocs.images, 106
panoptes.pocs.mount, 86
panoptes.pocs.mount.bisque, 75
panoptes.pocs.mount.ioptron, 73
panoptes.pocs.mount.ioptron.base, 71
panoptes.pocs.mount.ioptron.cem40, 72
panoptes.pocs.mount.ioptron.iej30pro, 73
panoptes.pocs.mount.mount, 77
panoptes.pocs.mount.serial, 83
panoptes.pocs.mount.simulator, 84
panoptes.pocs.scheduler, 93
panoptes.pocs.scheduler.constraint, 90
panoptes.pocs.scheduler.dispatch, 91
panoptes.pocs.scheduler.field, 91
panoptes.pocs.scheduler.observation, 90
panoptes.pocs.scheduler.observation.base, 87
panoptes.pocs.scheduler.observation.bias, 89
panoptes.pocs.scheduler.observation.compound, 90
panoptes.pocs.scheduler.observation.dark, 90
panoptes.pocs.scheduler.scheduler, 92
panoptes.pocs.sensor, 94
panoptes.pocs.sensor.remote, 93
panoptes.pocs.state, 98
panoptes.pocs.state.machine, 96
panoptes.pocs.state.states, 96
panoptes.pocs.state.states.default, 96
panoptes.pocs.state.states.default.analyzing, 94
panoptes.pocs.state.states.default.housekeeping, 94
panoptes.pocs.state.states.default.observing, 94
panoptes.pocs.state.states.default.parked, 95
panoptes.pocs.state.states.default.parking, 95
panoptes.pocs.state.states.default.pointing, 95
panoptes.pocs.state.states.default.ready, 95
panoptes.pocs.state.states.default.scheduling, 95
panoptes.pocs.state.states.default.sleeping, 95
panoptes.pocs.state.states.default.slewing,

95

panoptes.pocs.state.states.default.tracking,

96

panoptes.pocs.utils, 104

panoptes.pocs.utils.cli, 100

panoptes.pocs.utils.cli.camera, 98

panoptes.pocs.utils.cli.config, 98

panoptes.pocs.utils.cli.mount, 99

panoptes.pocs.utils.cli.notebook, 99

panoptes.pocs.utils.cli.sensor, 99

panoptes.pocs.utils.cli.weather, 100

panoptes.pocs.utils.database, 100

panoptes.pocs.utils.error, 101

panoptes.pocs.utils.location, 101

panoptes.pocs.utils.logger, 102

panoptes.pocs.utils.plotting, 103

panoptes.pocs.utils.service, 100

panoptes.pocs.utils.theskyx, 104

INDEX

A

- A_CLOSE_LIMIT (*panoptes.pocs.dome.astrohaven.Protocol* attribute), 53
- A_IS_CLOSED (*panoptes.pocs.dome.astrohaven.Protocol* attribute), 53
- A_OPEN_LIMIT (*panoptes.pocs.dome.astrohaven.Protocol* attribute), 53
- abgState (*panoptes.pocs.camera.sbigudrv.StartExposureParams* attribute), 49
- AboveMaxExptime, 101
- AbstractCamera (class in *panoptes.pocs.camera.camera*), 13
- AbstractDome (class in *panoptes.pocs.dome*), 56
- AbstractFilterWheel (class in *panoptes.pocs.filterwheel.filterwheel*), 57
- AbstractFocuser (class in *panoptes.pocs.focuser.focuser*), 66
- AbstractGPhotoCamera (class in *panoptes.pocs.camera.gphoto.base*), 9
- AbstractMount (class in *panoptes.pocs.mount.mount*), 77
- AbstractSDKCamera (class in *panoptes.pocs.camera.sdk*), 50
- AbstractSDKDriver (class in *panoptes.pocs.camera.sdk*), 50
- AbstractSerialDome (class in *panoptes.pocs.dome.abstract_serial_dome*), 52
- AbstractSerialFocuser (class in *panoptes.pocs.focuser.serial*), 70
- AbstractSerialMount (class in *panoptes.pocs.mount.serial*), 83
- add_observation() (*panoptes.pocs.scheduler.scheduler.BaseScheduler* method), 92
- add_to_exposure_list() (*panoptes.pocs.scheduler.observation.base.Observation* method), 87
- after_state() (*panoptes.pocs.state.machine.PanStateMachine* method), 96
- AlreadyVisited (class in *panoptes.pocs.scheduler.constraint*), 90
- Altitude (class in *panoptes.pocs.scheduler.constraint*), 90
- ambientTemperature (*panoptes.pocs.camera.sbigudrv.QueryTemperature* attribute), 43
- ambientThermistor (*panoptes.pocs.camera.sbigudrv.QueryTemperature* attribute), 43
- ANTI_DEW_HEATER (*panoptes.pocs.camera.libasi.ControlType* attribute), 26
- ASIDriver (class in *panoptes.pocs.camera.libasi*), 22
- AstrohavenDome (class in *panoptes.pocs.dome.astrohaven*), 53
- AT_HOME (*panoptes.pocs.mount.ioptron.MountState* attribute), 74
- at_mount_park (*panoptes.pocs.mount.bisque.Mount* property), 75
- at_mount_park (*panoptes.pocs.mount.mount.AbstractMount* property), 78
- AUTO (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 36
- AUTO_MAX_BRIGHTNESS (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
- AUTO_MAX_EXP (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
- AUTO_MAX_GAIN (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
- AUTO_TARGET_BRIGHTNESS (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
- autofocus() (*panoptes.pocs.camera.camera.AbstractCamera* method), 16
- autofocus() (*panoptes.pocs.focuser.focuser.AbstractFocuser* method), 66
- autofocus_error (*panoptes.pocs.focuser.focuser.AbstractFocuser* property), 67

B

- B_CLOSE_LIMIT (*panoptes.pocs.dome.astrohaven.Protocol* attribute), 53
- B_IS_CLOSED (*panoptes.pocs.dome.astrohaven.Protocol* attribute), 53
- B_OPEN_LIMIT (*panoptes.pocs.dome.astrohaven.Protocol* attribute), 53
- BAD_COMMAND (*panoptes.pocs.camera.sbigudrv.CFWError*

attribute), 36

BAD_MODEL (panoptes.pocs.camera.sbigudrv.CFWError attribute), 36

badColumns (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults attribute), 39

BANDWIDTHOVERLOAD (panoptes.pocs.camera.libasi.ControlType attribute), 27

baseAddress (panoptes.pocs.camera.sbigudrv.GetLinkStatusResults attribute), 42

BaseConstraint (class in panoptes.pocs.scheduler.constraint), 90

BaseScheduler (class in panoptes.pocs.scheduler.scheduler), 92

bayer_pattern (panoptes.pocs.camera.libasi.CameraInfo attribute), 25

BayerPattern (class in panoptes.pocs.camera.libasi), 24

before_state() (panoptes.pocs.state.machine.PanStateMachine method), 96

BelowMinExptime, 101

BG (panoptes.pocs.camera.libasi.BayerPattern attribute), 24

BiasObservation (class in panoptes.pocs.scheduler.observation.bias), 89

bit_depth (panoptes.pocs.camera.camera.AbstractCamera property), 17

bit_depth (panoptes.pocs.camera.gphoto.canon.Camera property), 11

bit_depth (panoptes.pocs.camera.libasi.CameraInfo attribute), 25

bit_depth (panoptes.pocs.camera.simulator.dslr.Camera property), 13

bit_depth (panoptes.pocs.camera.zwo.Camera property), 50

bitdepth (panoptes.pocs.camera.camera.AbstractCamera attribute), 15

BOTH (panoptes.pocs.camera.libasi.FlipStatus attribute), 28

BOTH_CLOSED (panoptes.pocs.dome.astrohaven.Protocol attribute), 53

BOTH_OPEN (panoptes.pocs.dome.astrohaven.Protocol attribute), 53

BRIGHTNESS (panoptes.pocs.camera.libasi.ControlType attribute), 27

BUFFER_TOO_SMALL (panoptes.pocs.camera.libasi.ErrorCode attribute), 27

BUSY (panoptes.pocs.camera.sbigudrv.CFWError attribute), 36

BUSY (panoptes.pocs.camera.sbigudrv.CFWStatus attribute), 38

attribute), 36

CAL_ERROR (panoptes.pocs.camera.sbigudrv.CFWError attribute), 36

calibrate() (panoptes.pocs.filterwheel.libefw.EFWDriver method), 59

Camera (class in panoptes.pocs.camera.fli), 21

Camera (class in panoptes.pocs.camera.gphoto.canon), 11

Camera (class in panoptes.pocs.camera.gphoto.remote), 11

Camera (class in panoptes.pocs.camera.sbig), 35

Camera (class in panoptes.pocs.camera.simulator.ccd), 12

Camera (class in panoptes.pocs.camera.simulator.dslr), 13

Camera (class in panoptes.pocs.camera.zwo), 50

camera (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property), 58

camera (panoptes.pocs.focuser.focuser.AbstractFocuser property), 67

camera (panoptes.pocs.hardware.HardwareName attribute), 105

camera_b0 (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults6 attribute), 40

camera_b1 (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults6 attribute), 40

CAMERA_CLOSED (panoptes.pocs.camera.libasi.ErrorCode attribute), 27

camera_ID (panoptes.pocs.camera.libasi.CameraInfo attribute), 25

CAMERA_REMOVED (panoptes.pocs.camera.libasi.ErrorCode attribute), 27

camera_unused (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults6 attribute), 40

CameraBusy, 101

cameraFound (panoptes.pocs.camera.sbigudrv.QueryUSBInfo attribute), 44

CameraInfo (class in panoptes.pocs.camera.libasi), 24

CameraMode (class in panoptes.pocs.camera.libasi), 25

camerasFound (panoptes.pocs.camera.sbigudrv.QueryUSBResults attribute), 44

camerasFound (panoptes.pocs.camera.sbigudrv.QueryUSBResults2 attribute), 44

camerasFound (panoptes.pocs.camera.sbigudrv.QueryUSBResults3 attribute), 44

cameraType (panoptes.pocs.camera.sbigudrv.EstablishLinkResults attribute), 39

cameraType (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults0 attribute), 39

cameraType (panoptes.pocs.camera.sbigudrv.GetLinkStatusResults attribute), 42

cameraType (panoptes.pocs.camera.sbigudrv.QueryUSBInfo attribute), 44

can_take_internal_darks

C

CAL_DATA (panoptes.pocs.camera.sbigudrv.CFWGetInfoSelect

attribute), 37

CFWL8 (*panoptes.pocs.camera.sbigudrv.CFWModelSelect attribute*), 37

CFWL8G (*panoptes.pocs.camera.sbigudrv.CFWModelSelect attribute*), 37

cfwModel (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 37

cfwModel (*panoptes.pocs.camera.sbigudrv.CFWResults attribute*), 38

CFWModelSelect (class in *panoptes.pocs.camera.sbigudrv*), 36

cfwParam1 (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 37

cfwParam2 (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 37

CFWParams (class in *panoptes.pocs.camera.sbigudrv*), 37

cfwPosition (*panoptes.pocs.camera.sbigudrv.CFWResults attribute*), 38

CFWResults (class in *panoptes.pocs.camera.sbigudrv*), 38

cfwResults1 (*panoptes.pocs.camera.sbigudrv.CFWResults attribute*), 38

cfwResults2 (*panoptes.pocs.camera.sbigudrv.CFWResults attribute*), 38

CFWStatus (class in *panoptes.pocs.camera.sbigudrv*), 38

cfwStatus (*panoptes.pocs.camera.sbigudrv.CFWResults attribute*), 38

check_firestore() (*panoptes.pocs.utils.database.PocsDatabase method*), 100

check_for_jupyter() (in module *panoptes.pocs.utils.cli.notebook*), 99

check_safety() (*panoptes.pocs.state.machine.PanStateMachine method*), 96

clear_available_observations() (*panoptes.pocs.scheduler.scheduler.BaseScheduler method*), 92

close() (*panoptes.pocs.dome.AbstractDome method*), 56

close() (*panoptes.pocs.dome.astrohaven.AstrohavenDome method*), 53

close() (*panoptes.pocs.dome.bisque.Dome method*), 54

close() (*panoptes.pocs.dome.simulator.Dome method*), 55

close() (*panoptes.pocs.filterwheel.libefw.EFWDriver method*), 59

CLOSE_A (*panoptes.pocs.dome.astrohaven.Protocol attribute*), 54

CLOSE_B (*panoptes.pocs.dome.astrohaven.Protocol attribute*), 54

CLOSE_BOTH (*panoptes.pocs.dome.astrohaven.Protocol attribute*), 54

close_camera() (*panoptes.pocs.camera.libasi.ASIDriver method*), 22

CLOSE_DEVICE (*panoptes.pocs.camera.sbigudrv.CFWCommand attribute*), 36

CLOSED (*panoptes.pocs.filterwheel.libefw.ErrorCode attribute*), 61

columns (*panoptes.pocs.camera.sbigudrv.GetCCDInfoResults2 attribute*), 39

comFailed (*panoptes.pocs.camera.sbigudrv.GetLinkStatusResults attribute*), 42

command (*panoptes.pocs.camera.sbigudrv.QueryCommandStatusParams attribute*), 42

command() (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera method*), 9

command() (*panoptes.pocs.camera.gphoto.remote.Camera method*), 11

compute_offset() (*panoptes.pocs.images.Image method*), 106

comTotal (*panoptes.pocs.camera.sbigudrv.GetLinkStatusResults attribute*), 42

config() (in module *panoptes.pocs.utils.cli.weather*), 100

connect() (*panoptes.pocs.camera.camera.AbstractCamera method*), 17

connect() (*panoptes.pocs.camera.fli.Camera method*), 21

connect() (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera method*), 9

connect() (*panoptes.pocs.camera.gphoto.canon.Camera method*), 11

connect() (*panoptes.pocs.camera.sbig.Camera method*), 35

connect() (*panoptes.pocs.camera.simulator.ccd.Camera method*), 12

connect() (*panoptes.pocs.camera.simulator.dslr.Camera method*), 13

connect() (*panoptes.pocs.camera.zwo.Camera method*), 50

connect() (*panoptes.pocs.dome.abstract_serial_dome.AbstractSerialDome method*), 52

connect() (*panoptes.pocs.dome.AbstractDome method*), 56

connect() (*panoptes.pocs.dome.bisque.Dome method*), 54

connect() (*panoptes.pocs.dome.simulator.Dome method*), 55

connect() (*panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel method*), 58

connect() (*panoptes.pocs.filterwheel.sbig.FilterWheel method*), 62

connect() (*panoptes.pocs.filterwheel.simulator.FilterWheel method*), 62

connect() (*panoptes.pocs.filterwheel.zwo.FilterWheel method*), 63

connect() (*panoptes.pocs.focuser.birger.Focuser method*), 65

connect() (*panoptes.pocs.focuser.focuslynx.Focuser method*), 65

method), 69

connect() (panoptes.pocs.focuser.serial.AbstractSerialFocuser method), 70

connect() (panoptes.pocs.focuser.simulator.Focuser method), 70

connect() (panoptes.pocs.mount.bisque.Mount method), 75

connect() (panoptes.pocs.mount.mount.AbstractMount method), 78

connect() (panoptes.pocs.mount.serial.AbstractSerialMount method), 83

connect() (panoptes.pocs.mount.simulator.Mount method), 84

connect() (panoptes.pocs.utils.theskysx.TheSkyX method), 104

control_type (panoptes.pocs.camera.libasi.ControlCaps attribute), 26

ControlCaps (class in panoptes.pocs.camera.libasi), 26

controlParameter (panoptes.pocs.camera.sbigudrv.GetDriverControlParameters attribute), 41

controlParameter (panoptes.pocs.camera.sbigudrv.SetDriverControlParameters attribute), 48

ControlType (class in panoptes.pocs.camera.libasi), 26

controlValue (panoptes.pocs.camera.sbigudrv.GetDriverControlParameters attribute), 41

controlValue (panoptes.pocs.camera.sbigudrv.SetDriverControlParameters attribute), 48

COOLER_ON (panoptes.pocs.camera.libasi.ControlType attribute), 27

COOLER_POWER_PERC (panoptes.pocs.camera.libasi.ControlType attribute), 27

cooling_enabled (panoptes.pocs.camera.camera.AbstractCamera attribute), 15

cooling_enabled (panoptes.pocs.camera.camera.AbstractCamera property), 18

cooling_enabled (panoptes.pocs.camera.fli.Camera property), 21

cooling_enabled (panoptes.pocs.camera.sbig.Camera property), 35

cooling_enabled (panoptes.pocs.camera.simulator.ccd.Camera property), 12

cooling_enabled (panoptes.pocs.camera.zwo.Camera property), 50

cooling_power (panoptes.pocs.camera.camera.AbstractCamera attribute), 15

cooling_power (panoptes.pocs.camera.camera.AbstractCamera property), 18

cooling_power (panoptes.pocs.camera.fli.Camera property), 21

cooling_power (panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera property), 9

cooling_power (panoptes.pocs.camera.sbig.Camera property), 35

cooling_power (panoptes.pocs.camera.simulator.ccd.Camera property), 12

cooling_power (panoptes.pocs.camera.zwo.Camera property), 50

coolingEnabled (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatus attribute), 43

correct_tracking() (panoptes.pocs.mount.mount.AbstractMount method), 78

create_cameras_from_config() (in module panoptes.pocs.camera), 51

create_constraints_from_config() (in module panoptes.pocs.scheduler), 93

create_dome_from_config() (in module panoptes.pocs.dome), 57

create_dome_simulator() (in module panoptes.pocs.dome), 57

create_location_from_config() (in module panoptes.pocs.utils.location), 101

create_mount_from_config() (in module panoptes.pocs.mount), 86

create_mount_simulator() (in module panoptes.pocs.mount), 87

create_scheduler_from_config() (in module panoptes.pocs.scheduler), 93

create_sky_num (panoptes.pocs.scheduler.observation.base.Observation property), 87

create_filter (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property), 58

current_observation (panoptes.pocs.scheduler.scheduler.BaseScheduler property), 92

CUSTOM (panoptes.pocs.mount.ioptron.MountTrackingState attribute), 75

D

DarkObservation (class in panoptes.pocs.scheduler.observation.dark), 90

DATA_REGISTERS (panoptes.pocs.camera.sbigudrv.CFWGetInfoSelect attribute), 36

default_value (panoptes.pocs.camera.libasi.ControlCaps attribute), 26

delta_dec (panoptes.pocs.images.OffsetError attribute), 107

delta_ra (panoptes.pocs.images.OffsetError attribute), 107

description (panoptes.pocs.camera.libasi.ControlCaps attribute), 26

DEVICE_NOT_CLOSED (panoptes.pocs.camera.sbigudrv.CFWError attribute), 36

DEVICE_NOT_OPEN (panoptes.pocs.camera.sbigudrv.CFWError attribute), 36

deviceType (panoptes.pocs.camera.sbigudrv.OpenDeviceParams attribute), 42

directory (*panoptes.pocs.scheduler.observation.base.Observation* property), 88
 disable_dark_subtract() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 disable_vdd_optimized() (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 47
 disconnect() (*panoptes.pocs.dome.abstract_serial_dome.AbstractSerialDome* method), 52
 disconnect() (*panoptes.pocs.dome.AbstractDome* method), 56
 disconnect() (*panoptes.pocs.dome.bisque.Dome* method), 54
 disconnect() (*panoptes.pocs.dome.simulator.Dome* method), 55
 disconnect() (*panoptes.pocs.mount.bisque.Mount* method), 75
 disconnect() (*panoptes.pocs.mount.mount.AbstractMount* method), 78
 disconnect() (*panoptes.pocs.mount.serial.AbstractSerialMount* method), 83
 disconnect() (*panoptes.pocs.mount.simulator.Mount* method), 84
 disconnect() (*panoptes.pocs.sensor.remote.RemoteMonitor* method), 94
 distance_from_target() (*panoptes.pocs.mount.mount.AbstractMount* method), 78
 Dome (class in *panoptes.pocs.dome.bisque*), 54
 Dome (class in *panoptes.pocs.dome.simulator*), 55
 Dome (in module *panoptes.pocs.dome.astrohaven*), 53
 dome (*panoptes.pocs.hardware.HardwareName* attribute), 105
 download_iers_a_file() (in module *panoptes.pocs.utils.location*), 102
 dumpExtra (*panoptes.pocs.camera.sbigudrv.GetCCDInfoResults* attribute), 40
 Duration (class in *panoptes.pocs.scheduler.constraint*), 90
E
 e_per_adu (*panoptes.pocs.camera.libasi.CameraInfo* attribute), 25
 earth_location (*panoptes.pocs.utils.location.SiteDetails* attribute), 101
 EAST (*panoptes.pocs.camera.libasi.GuideDirection* attribute), 29
 EFWDriver (class in *panoptes.pocs.filterwheel.libefw*), 59
 EFWInfo (class in *panoptes.pocs.filterwheel.libefw*), 60
 egain (*panoptes.pocs.camera.camera.AbstractCamera* attribute), 15
 egain (*panoptes.pocs.camera.camera.AbstractCamera* property), 18
 egain (*panoptes.pocs.camera.gphoto.canon.Camera* property), 11
 egain (*panoptes.pocs.camera.sbig.Camera* property), 35
 egain (*panoptes.pocs.camera.simulator.dslr.Camera* property), 13
 egain (*panoptes.pocs.camera.zwo.Camera* property), 50
 enable_dark_subtract() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 enabled (*panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults* attribute), 43
 END (*panoptes.pocs.camera.libasi.CameraMode* attribute), 26
 END (*panoptes.pocs.camera.libasi.ErrorCode* attribute), 27
 END (*panoptes.pocs.camera.libasi.ImgType* attribute), 29
 END (*panoptes.pocs.filterwheel.libefw.ErrorCode* attribute), 61
 EndExposureParams (class in *panoptes.pocs.camera.sbigudrv*), 38
 EndReadoutParams (class in *panoptes.pocs.camera.sbigudrv*), 38
 ERROR_STATE (*panoptes.pocs.filterwheel.libefw.ErrorCode* attribute), 61
 ErrorCode (class in *panoptes.pocs.camera.libasi*), 27
 ErrorCode (class in *panoptes.pocs.filterwheel.libefw*), 61
 establish_link() (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 47
 EstablishLinkParams (class in *panoptes.pocs.camera.sbigudrv*), 38
 EstablishLinkResults (class in *panoptes.pocs.camera.sbigudrv*), 39
 Exposure (class in *panoptes.pocs.scheduler.observation.base*), 87
 EXPOSURE (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
 exposure_error (*panoptes.pocs.camera.camera.AbstractCamera* property), 18
 EXPOSURE_IN_PROGRESS (*panoptes.pocs.camera.libasi.ErrorCode* attribute), 27
 ExposureStatus (class in *panoptes.pocs.camera.libasi*), 28
 exposureTime (*panoptes.pocs.camera.sbigudrv.StartExposureParams2* attribute), 49
 exptime (*panoptes.pocs.scheduler.observation.base.Observation* property), 88
 exptime (*panoptes.pocs.scheduler.observation.compound.Observation* property), 90
 exptime (*panoptes.pocs.scheduler.observation.dark.DarkObservation* property), 90
 exptimes (*panoptes.pocs.scheduler.observation.base.Observation* property), 88
 exptimes (*panoptes.pocs.scheduler.observation.compound.Observation*

property), 90

externalTrackingCCDPower (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 43

externalTrackingCCDTemperature (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 43

extraBits (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults6 attribute), 41

EXTRACTED (panoptes.pocs.mount.ioptron.MountGPS attribute), 73

F

FAILED (panoptes.pocs.camera.libasi.ExposureStatus attribute), 28

FAN_ON (panoptes.pocs.camera.libasi.ControlType attribute), 27

fanEnabled (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 43

fanPower (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 43

fanSpeed (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 43

Field (class in panoptes.pocs.scheduler.field), 91

field_name (panoptes.pocs.scheduler.field.Field property), 91

fields_file (panoptes.pocs.scheduler.scheduler.BaseScheduler property), 92

fields_list (panoptes.pocs.scheduler.scheduler.BaseScheduler property), 92

file_extension (panoptes.pocs.camera.camera.AbstractCamera attribute), 15

file_extension (panoptes.pocs.camera.camera.AbstractCamera property), 18

filter_name() (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel method), 58

filter_names (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property), 58

filter_type (panoptes.pocs.camera.camera.AbstractCamera attribute), 13

filter_type (panoptes.pocs.camera.camera.AbstractCamera property), 18

filter_wheel (panoptes.pocs.camera.camera.AbstractCamera attribute), 14

FilterWheel (class in panoptes.pocs.filterwheel.sbig), 61

FilterWheel (class in panoptes.pocs.filterwheel.simulator), 62

FilterWheel (class in panoptes.pocs.filterwheel.zwo), 63

find_home() (panoptes.pocs.dome.bisque.Dome method), 54

FIRMWARE_VERSION (panoptes.pocs.camera.sbigudrv.CFWGetInfoResults attribute), 36

firmware_version (panoptes.pocs.filterwheel.sbig.FilterWheel property), 62

firmwareVersion (panoptes.pocs.focuser.birger.Focuser property), 65

firmware_version (panoptes.pocs.focuser.focuslynx.Focuser property), 69

firmwareVersion (panoptes.pocs.camera.sbigudrv.GetCCDInfoResults0 attribute), 39

first_exposure (panoptes.pocs.scheduler.observation.base.Observation property), 88

FLIClose() (panoptes.pocs.camera.libfli.FLIDriver method), 30

FLIDriver (class in panoptes.pocs.camera.libfli), 30

FLIExposeFrame() (panoptes.pocs.camera.libfli.FLIDriver method), 30

FLIGetArrayArea() (panoptes.pocs.camera.libfli.FLIDriver method), 30

FLIGetCoolerPower() (panoptes.pocs.camera.libfli.FLIDriver method), 30

FLIGetExposureStatus() (panoptes.pocs.camera.libfli.FLIDriver method), 30

FLIGetFWRevision() (panoptes.pocs.camera.libfli.FLIDriver method), 30

FLIGetHWRRevision() (panoptes.pocs.camera.libfli.FLIDriver method), 31

FLIGetModel() (panoptes.pocs.camera.libfli.FLIDriver method), 31

FLIGetPixelSize() (panoptes.pocs.camera.libfli.FLIDriver method), 31

FLIGetSerialString() (panoptes.pocs.camera.libfli.FLIDriver method), 31

FLIGetTemperature() (panoptes.pocs.camera.libfli.FLIDriver method), 31

FLIGetVisibleArea() (panoptes.pocs.camera.libfli.FLIDriver method), 32

FLIGrabFrame() (panoptes.pocs.camera.libfli.FLIDriver method), 32

FLIGrabRow() (panoptes.pocs.camera.libfli.FLIDriver method), 32

FLIList() (panoptes.pocs.camera.libfli.FLIDriver method), 32

FLIOpen() (panoptes.pocs.camera.libfli.FLIDriver method), 33

FLIP (panoptes.pocs.camera.libasi.ControlType attribute), 27

FlipStatus (class in panoptes.pocs.camera.libasi), 28

FLISetExposureTime() (panoptes.pocs.camera.libfli.FLIDriver method), 33

FLISetFrameType() (*panoptes.pocs.camera.libfli.FLIDriver* attribute), 33
 FLISetHBin() (*panoptes.pocs.camera.libfli.FLIDriver* attribute), 34
 FLISetImageArea() (*panoptes.pocs.camera.libfli.FLIDriver* attribute), 34
 FLISetNFlushes() (*panoptes.pocs.camera.libfli.FLIDriver* attribute), 34
 FLISetTemperature() (*panoptes.pocs.camera.libfli.FLIDriver* attribute), 34
 FLISetVBin() (*panoptes.pocs.camera.libfli.FLIDriver* attribute), 34
 Focuser (class in *panoptes.pocs.focuser.astromechanics*), 64
 Focuser (class in *panoptes.pocs.focuser.birger*), 65
 Focuser (class in *panoptes.pocs.focuser.focuslynx*), 68
 Focuser (class in *panoptes.pocs.focuser.simulator*), 70
 focuser (*panoptes.pocs.camera.camera.AbstractCamera* attribute), 14
 format() (*panoptes.pocs.utils.logger.PanLogger* attribute), 102
 from_altaz() (*panoptes.pocs.scheduler.field.Field* class method), 91
 from_dict() (*panoptes.pocs.scheduler.observation.base.Observation* class method), 88
 from_dict() (*panoptes.pocs.scheduler.observation.compound.Observation* class method), 90
 FW5_8300 (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 37
 FW5_STF_DETENT (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 37
 FW5_STX (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 37
 FW7_STX (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 37
 FW8_8300 (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 37
 FW8_STT (*panoptes.pocs.camera.sbigudrv.CFWModelSelect* attribute), 37
G
 gain (*panoptes.pocs.camera.camera.AbstractCamera* attribute), 15
 GAIN (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
 gain (*panoptes.pocs.camera.sbigudrv.ReadoutInfo* attribute), 45
 gain (*panoptes.pocs.camera.zwo.Camera* property), 50
 GAMMA (*panoptes.pocs.camera.libasi.ControlType* attribute), 27
 GB (*panoptes.pocs.camera.libasi.BayerPattern* attribute), 24
 GEM28 (*panoptes.pocs.mount.ioptron.MountInfo* attribute), 73
 GEM28EC (*panoptes.pocs.mount.ioptron.MountInfo* attribute), 73
 GEM45 (*panoptes.pocs.mount.ioptron.MountInfo* attribute), 74
 GEM45EC (*panoptes.pocs.mount.ioptron.MountInfo* attribute), 74
 GENERAL_ERROR (*panoptes.pocs.camera.libasi.ErrorCode* attribute), 28
 GENERAL_ERROR (*panoptes.pocs.filterwheel.libefw.ErrorCode* attribute), 61
 get_all_names() (in module *panoptes.pocs.hardware*), 105
 get_camera_mode() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_camera_property() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_camera_property_by_id() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_camera_supported_mode() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_ccd_info() (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 47
 get_command_result() (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera* method), 9
 get_command_result() (*panoptes.pocs.camera.gphoto.remote.Camera* method), 11
 get_config() (*panoptes.pocs.base.PanBase* method), 104
 get_control_caps() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_control_value() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_current_coordinates() (*panoptes.pocs.mount.mount.AbstractMount* method), 78
 get_current_coordinates() (*panoptes.pocs.mount.simulator.Mount* method), 84
 get_cutout() (*panoptes.pocs.camera.camera.AbstractCamera* method), 18
 get_devices() (*panoptes.pocs.camera.libasi.ASIDriver* method), 22
 get_devices() (*panoptes.pocs.camera.libfli.FLIDriver* method), 34
 get_devices() (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 47

[get_devices\(\) \(panoptes.pocs.camera.sdk.AbstractSDKDriver method\), 91](#)
[get_devices\(\) \(panoptes.pocs.camera.simulator.ccd.SDKDriver method\), 92](#)
[get_devices\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 60](#)
[get_direction\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 60](#)
[get_driver_handle\(\) \(panoptes.pocs.camera.sbigudrv.SBIGDriver method\), 48](#)
[get_dropped_frames\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_exposure\(\) \(panoptes.pocs.scheduler.observation.base.Observation method\), 88](#)
[get_exposure_data\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_exposure_status\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_exposure_status\(\) \(panoptes.pocs.camera.sbigudrv.SBIGDriver method\), 48](#)
[get_gain_offset\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_gphoto2_cmd\(\) \(in module panoptes.pocs.camera\), 52](#)
[get_header_pointing\(\) \(panoptes.pocs.images.Image method\), 106](#)
[get_ID\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 22](#)
[get_ID\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 59](#)
[GET_INFO \(panoptes.pocs.camera.sbigudrv.CFWCommand attribute\), 36](#)
[get_link_status\(\) \(panoptes.pocs.camera.sbigudrv.SBIGDriver method\), 48](#)
[get_logger\(\) \(in module panoptes.pocs.utils.logger\), 102](#)
[get_ms_offset\(\) \(panoptes.pocs.mount.mount.AbstractMount method\), 78](#)
[get_ms_offset\(\) \(panoptes.pocs.mount.simulator.Mount method\), 84](#)
[get_num\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 60](#)
[get_num_of_connected_cameras\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_num_of_controls\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_observation\(\) \(panoptes.pocs.scheduler.dispatch.Scheduler method\), 50](#)
[get_observation\(\) \(panoptes.pocs.scheduler.scheduler.BaseScheduler method\), 92](#)
[get_page\(\) \(in module panoptes.pocs.utils.cli.weather\), 100](#)
[get_position\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 60](#)
[get_product_ids\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_product_ids\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 60](#)
[get_property\(\) \(panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera method\), 10](#)
[get_property\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 60](#)
[get_roi_format\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_score\(\) \(panoptes.pocs.scheduler.constraint.AlreadyVisited method\), 90](#)
[get_score\(\) \(panoptes.pocs.scheduler.constraint.Altitude method\), 90](#)
[get_score\(\) \(panoptes.pocs.scheduler.constraint.BaseConstraint method\), 90](#)
[get_score\(\) \(panoptes.pocs.scheduler.constraint.Duration method\), 91](#)
[get_score\(\) \(panoptes.pocs.scheduler.constraint.MoonAvoidance method\), 91](#)
[get_SDK_version\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 22](#)
[get_SDK_version\(\) \(panoptes.pocs.camera.libfli.FLIDriver method\), 34](#)
[get_SDK_version\(\) \(panoptes.pocs.camera.sbigudrv.SBIGDriver method\), 47](#)
[get_SDK_version\(\) \(panoptes.pocs.camera.sdk.AbstractSDKDriver method\), 50](#)
[get_SDK_version\(\) \(panoptes.pocs.camera.simulator.ccd.SDKDriver method\), 12](#)
[get_SDK_version\(\) \(panoptes.pocs.filterwheel.libefw.EFWDriver method\), 59](#)
[get_serial_number\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_shutterspeed_index\(\) \(panoptes.pocs.camera.gphoto.canon.Camera class method\), 11](#)
[get_simulator_names\(\) \(in module panoptes.pocs.hardware\), 105](#)
[get_start_position\(\) \(panoptes.pocs.camera.libasi.ASIDriver method\), 23](#)
[get_target_coordinates\(\) \(panoptes.pocs.mount.mount.AbstractMount method\), 79](#)
[get_tracking_correction\(\) \(panoptes.pocs.scheduler.dispatch.Scheduler method\), 50](#)

(*panoptes.pocs.mount.mount.AbstractMount* method), 79

`get_trigger_output_io_conf()` (*panoptes.pocs.camera.libasi.ASIDriver* method), 23

`get_value()` (in module *panoptes.pocs.utils.cli.config*), 98

`get_video_data()` (*panoptes.pocs.camera.libasi.ASIDriver* method), 23

`get_wcs_pointing()` (*panoptes.pocs.images.Image* method), 106

`GetCCDInfoParams` (class *panoptes.pocs.camera.sbigudrv*), 39

`GetCCDInfoResults0` (class *panoptes.pocs.camera.sbigudrv*), 39

`GetCCDInfoResults2` (class *panoptes.pocs.camera.sbigudrv*), 39

`GetCCDInfoResults4` (class *panoptes.pocs.camera.sbigudrv*), 40

`GetCCDInfoResults6` (class *panoptes.pocs.camera.sbigudrv*), 40

`GetDriverControlParams` (class *panoptes.pocs.camera.sbigudrv*), 41

`GetDriverControlResults` (class *panoptes.pocs.camera.sbigudrv*), 41

`GetDriverHandleResults` (class *panoptes.pocs.camera.sbigudrv*), 41

`GetDriverInfoParams` (class *panoptes.pocs.camera.sbigudrv*), 41

`GetDriverInfoResults0` (class *panoptes.pocs.camera.sbigudrv*), 41

`GetLinkStatusResults` (class *panoptes.pocs.camera.sbigudrv*), 41

`GOTO` (*panoptes.pocs.camera.sbigudrv.CFWCommand* attribute), 36

`goto_next_state()` (*panoptes.pocs.state.machine.PanStateMachine* attribute), 96

`gphoto_file_download()` (*panoptes.pocs.camera.gphoto.base.AbstractGPhoto* class method), 10

`GPS` (*panoptes.pocs.mount.ioptron.MountTimeSource* attribute), 74

`GR` (*panoptes.pocs.camera.libasi.BayerPattern* attribute), 24

`GuideDirection` (class in *panoptes.pocs.camera.libasi*), 28

`GUIDING` (*panoptes.pocs.mount.ioptron.MountState* attribute), 74

H

`halt()` (*panoptes.pocs.focuser.focuslynx.Focuser* method), 69

`HAND_CONTROLLER` (*panoptes.pocs.mount.ioptron.MountTimeSource* attribute), 75

`handle` (*panoptes.pocs.camera.sbigudrv.GetDriverHandleResults* attribute), 41

`handle` (*panoptes.pocs.camera.sbigudrv.SetDriverHandleParams* attribute), 48

`HARDWARE_BIN` (*panoptes.pocs.camera.libasi.ControlType* attribute), 27

`hardware_version` (*panoptes.pocs.focuser.birger.Focuser* property), 65

`hardware_version` (*panoptes.pocs.focuser.focuslynx.Focuser* property), 69

`HardwareName` (class in *panoptes.pocs.hardware*), 105

in `has_cooler` (*panoptes.pocs.camera.libasi.CameraInfo* attribute), 25

in `has_filterwheel` (*panoptes.pocs.camera.camera.AbstractCamera* property), 18

in `has_focuser` (*panoptes.pocs.camera.camera.AbstractCamera* property), 18

in `has_mechanical_shutter` (*panoptes.pocs.camera.libasi.CameraInfo* attribute), 25

in `has_ST4_port` (*panoptes.pocs.camera.libasi.CameraInfo* attribute), 25

in `has_target` (*panoptes.pocs.mount.mount.AbstractMount* property), 79

in `has_valid_observations` (*panoptes.pocs.scheduler.scheduler.BaseScheduler* property), 92

in `heatsinkTemperature` (*panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults* attribute), 43

in `height` (*panoptes.pocs.camera.sbigudrv.ReadoutInfo* attribute), 45

in `height` (*panoptes.pocs.camera.sbigudrv.StartExposureParams2* attribute), 49

in `height` (*panoptes.pocs.camera.sbigudrv.StartReadoutParams* attribute), 49

in `HIGH_SPEED_MODE` (*panoptes.pocs.camera.libasi.ControlType* attribute), 27

in `home_and_park()` (*panoptes.pocs.mount.mount.AbstractMount* method), 79

`HORIZ` (*panoptes.pocs.camera.libasi.FlipStatus* attribute), 28

`host` (*panoptes.pocs.utils.cli.config.HostInfo* attribute), 98

`HostInfo` (class in *panoptes.pocs.utils.cli.config*), 98

I

`I2C_ERROR` (*panoptes.pocs.camera.sbigudrv.CFWError* attribute), 36

`ID` (class in *panoptes.pocs.camera.libasi*), 29

`id` (*panoptes.pocs.camera.libasi.ID* attribute), 29

`id` (*panoptes.pocs.filterwheel.libefw.EFWInfo* attribute), 29

- IDLE (*panoptes.pocs.camera.libasi.ExposureStatus attribute*), 28
- IDLE (*panoptes.pocs.camera.sbigudrv.CFWStatus attribute*), 38
- iEQ30Pro (*panoptes.pocs.mount.ioptron.MountInfo attribute*), 74
- iEQ45Pro (*panoptes.pocs.mount.ioptron.MountInfo attribute*), 74
- iEQ45ProAA (*panoptes.pocs.mount.ioptron.MountInfo attribute*), 74
- Image (*class in panoptes.pocs.images*), 106
- image_id (*panoptes.pocs.scheduler.observation.base.Exposure attribute*), 87
- image_type (*panoptes.pocs.camera.camera.AbstractCamera attribute*), 15
- image_type (*panoptes.pocs.camera.zwo.Camera property*), 51
- ImageSaturated, 101
- imagingABG (*panoptes.pocs.camera.sbigudrv.GetCCDInfoResults2 attribute*), 39
- imagingCCDPower (*panoptes.pocs.camera.sbigudrv.QueryTemperatureResults2 attribute*), 43
- imagingCCDTemperature (*panoptes.pocs.camera.sbigudrv.QueryTemperatureResults2 attribute*), 43
- ImgType (*class in panoptes.pocs.camera.libasi*), 29
- INIT (*panoptes.pocs.camera.sbigudrv.CFWCommand attribute*), 36
- init_camera() (*panoptes.pocs.camera.libasi.ASIDriver method*), 23
- initialize() (*panoptes.pocs.mount.bisque.Mount method*), 75
- initialize() (*panoptes.pocs.mount.ioptron.base.Mount method*), 71
- initialize() (*panoptes.pocs.mount.mount.AbstractMount method*), 80
- initialize() (*panoptes.pocs.mount.simulator.Mount method*), 84
- inLength (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 37
- inPtr (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 37
- insert() (*panoptes.pocs.utils.database.PocsDB method*), 100
- insert_current() (*panoptes.pocs.utils.database.PocsDB method*), 100
- INVALID_CONTROL_TYPE (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_FILEFORMAT (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_ID (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_ID (*panoptes.pocs.filterwheel.libefw.ErrorCode attribute*), 61
- INVALID_IMGTYP (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_INDEX (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_INDEX (*panoptes.pocs.filterwheel.libefw.ErrorCode attribute*), 61
- INVALID_MODE (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_PATH (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_SEQUENCE (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_SIZE (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- INVALID_VALUE (*panoptes.pocs.filterwheel.libefw.ErrorCode attribute*), 61
- ipAddress (*panoptes.pocs.camera.sbigudrv.OpenDeviceParams attribute*), 42
- is_auto_supported (*panoptes.pocs.camera.libasi.ControlCaps attribute*), 36
- is_closed (*panoptes.pocs.dome.astrohaven.AstrohavenDome property*), 53
- is_closed (*panoptes.pocs.dome.bisque.Dome property*), 54
- is_closed (*panoptes.pocs.dome.simulator.Dome property*), 55
- is_closed() (*panoptes.pocs.dome.AbstractDome method*), 56
- is_color_camera (*panoptes.pocs.camera.libasi.CameraInfo attribute*), 25
- is_connected (*panoptes.pocs.camera.camera.AbstractCamera attribute*), 16
- is_connected (*panoptes.pocs.camera.camera.AbstractCamera property*), 18
- is_connected (*panoptes.pocs.dome.abstract_serial_dome.AbstractSerialDome property*), 52
- is_connected (*panoptes.pocs.dome.AbstractDome property*), 56
- is_connected (*panoptes.pocs.dome.bisque.Dome property*), 54
- is_connected (*panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property*), 58
- is_connected (*panoptes.pocs.focuser.focuser.AbstractFocuser property*), 67
- is_connected (*panoptes.pocs.focuser.focuslynx.Focuser property*), 69
- is_connected (*panoptes.pocs.focuser.serial.AbstractSerialFocuser property*), 70
- is_connected (*panoptes.pocs.mount.mount.AbstractMount property*), 80
- is_connected (*panoptes.pocs.utils.theskyx.TheSkyX property*), 104
- is_cooled_camera (*panoptes.pocs.camera.camera.AbstractCamera*

attribute), 16

is_cooled_camera (panoptes.pocs.camera.camera.AbstractCamera property), 18

is_exposing (panoptes.pocs.camera.camera.AbstractCamera attribute), 16

is_exposing (panoptes.pocs.camera.camera.AbstractCamera property), 18

is_exposing (panoptes.pocs.camera.fli.Camera property), 21

is_exposing (panoptes.pocs.camera.gphoto.base.AbstractCamera property), 10

is_exposing (panoptes.pocs.camera.gphoto.remote.Camera property), 11

is_exposing (panoptes.pocs.camera.sbig.Camera property), 35

is_exposing (panoptes.pocs.camera.zwo.Camera property), 51

is_home (panoptes.pocs.mount.bisque.Mount property), 75

is_home (panoptes.pocs.mount.ioptron.base.Mount property), 71

is_home (panoptes.pocs.mount.mount.AbstractMount property), 80

is_initialized (panoptes.pocs.mount.mount.AbstractMount property), 80

is_moving (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property), 58

is_moving (panoptes.pocs.filterwheel.sbig.FilterWheel property), 62

is_moving (panoptes.pocs.filterwheel.simulator.FilterWheel property), 63

is_moving (panoptes.pocs.filterwheel.zwo.FilterWheel property), 63

is_moving (panoptes.pocs.focuser.focuslynx.Focuser property), 69

is_moving (panoptes.pocs.focuser.serial.AbstractSerialFocuser property), 70

is_moving (panoptes.pocs.focuser.simulator.Focuser property), 70

is_moving() (panoptes.pocs.focuser.focuser.AbstractFocuser method), 68

is_observing (panoptes.pocs.camera.camera.AbstractCamera property), 19

is_open (panoptes.pocs.dome.astrohaven.AstrohavenDome property), 53

is_open (panoptes.pocs.dome.bisque.Dome property), 54

is_open (panoptes.pocs.dome.simulator.Dome property), 56

is_open() (panoptes.pocs.dome.AbstractDome method), 57

is_parked (panoptes.pocs.dome.bisque.Dome property), 54

is_parked (panoptes.pocs.mount.bisque.Mount property), 76

is_parked (panoptes.pocs.mount.mount.AbstractMount property), 80

is_primary (panoptes.pocs.camera.camera.AbstractCamera attribute), 14

is_primary (panoptes.pocs.scheduler.observation.base.Exposure attribute), 87

is_ready (panoptes.pocs.camera.camera.AbstractCamera attribute), 16

is_ready (panoptes.pocs.camera.camera.AbstractCamera property), 19

is_ready (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property), 58

is_ready (panoptes.pocs.focuser.focuser.AbstractFocuser property), 68

is_slewing (panoptes.pocs.mount.bisque.Mount property), 76

is_slewing (panoptes.pocs.mount.mount.AbstractMount property), 80

is_temperature_stable (panoptes.pocs.camera.camera.AbstractCamera attribute), 16

is_temperature_stable (panoptes.pocs.camera.camera.AbstractCamera property), 19

is_tracking (panoptes.pocs.mount.bisque.Mount property), 76

is_tracking (panoptes.pocs.mount.mount.AbstractMount property), 80

is_trigger_camera (panoptes.pocs.camera.libasi.CameraInfo attribute), 25

is_unidirectional (panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property), 58

is_unidirectional (panoptes.pocs.filterwheel.sbig.FilterWheel property), 62

is_unidirectional (panoptes.pocs.filterwheel.simulator.FilterWheel property), 63

is_unidirectional (panoptes.pocs.filterwheel.zwo.FilterWheel property), 63

is_USB3_camera (panoptes.pocs.camera.libasi.CameraInfo attribute), 25

is_USB3_host (panoptes.pocs.camera.libasi.CameraInfo attribute), 25

is_writable (panoptes.pocs.camera.libasi.ControlCaps attribute), 26

K

KING (panoptes.pocs.mount.ioptron.MountTrackingState attribute), 75

L

last_exposure (panoptes.pocs.scheduler.observation.base.Observation property), 88

[left \(panoptes.pocs.camera.sbigudrv.StartExposureParameters attribute\), 49](#)
[left \(panoptes.pocs.camera.sbigudrv.StartReadoutParameters attribute\), 49](#)
[lens_info \(panoptes.pocs.focuser.birger.Focuser property\), 65](#)
[library_path \(panoptes.pocs.camera.camera.AbstractCamera attribute\), 15](#)
[linkEstablished \(panoptes.pocs.camera.sbigudrv.GetLinkStatus attribute\), 42](#)
[list_connected_cameras\(\) \(in module panoptes.pocs.camera\), 52](#)
[LISTEN_TIMEOUT \(panoptes.pocs.dome.astrohaven.Astrohaven attribute\), 53](#)
[load_properties\(\) \(panoptes.pocs.camera.gphoto.base.AbstractGPhoto attribute\), 10](#)
[load_state_table\(\) \(panoptes.pocs.state.machine.PanStateMachine class method\), 97](#)
[location \(panoptes.pocs.mount.mount.AbstractMount property\), 80](#)
[location \(panoptes.pocs.utils.location.SiteDetails attribute\), 101](#)
[lptBaseAddress \(panoptes.pocs.camera.sbigudrv.OpenDemux attribute\), 42](#)
[LUNAR \(panoptes.pocs.mount.ioptron.MountTrackingState attribute\), 75](#)

M

[magnitude \(panoptes.pocs.images.OffsetError attribute\), 107](#)
[main\(\) \(in module panoptes.pocs.utils.cli.config\), 98](#)
[main\(\) \(in module panoptes.pocs.utils.cli.sensor\), 99](#)
[make_autofocus_plot\(\) \(in module panoptes.pocs.utils.plotting\), 103](#)
[max_height \(panoptes.pocs.camera.libasi.CameraInfo attribute\), 25](#)
[max_position \(panoptes.pocs.focuser.astromechanics.Focuser property\), 64](#)
[max_position \(panoptes.pocs.focuser.birger.Focuser property\), 65](#)
[max_position \(panoptes.pocs.focuser.focuslynx.Focuser property\), 69](#)
[max_position \(panoptes.pocs.focuser.simulator.Focuser property\), 71](#)
[max_position\(\) \(panoptes.pocs.focuser.focuser.AbstractFocuser method\), 68](#)
[max_value \(panoptes.pocs.camera.libasi.ControlCaps attribute\), 26](#)
[max_width \(panoptes.pocs.camera.libasi.CameraInfo attribute\), 25](#)
[maxRequest \(panoptes.pocs.camera.sbigudrv.GetDriverInfoResult attribute\), 41](#)
[MERIDIAN_FLIPPING \(panoptes.pocs.mount.ioptron.MountState attribute\), 74](#)

[metadata \(panoptes.pocs.scheduler.observation.base.Exposure attribute\), 87](#)
[min_position \(panoptes.pocs.focuser.astromechanics.Focuser property\), 64](#)
[min_position \(panoptes.pocs.focuser.birger.Focuser property\), 65](#)
[min_position \(panoptes.pocs.focuser.focuslynx.Focuser property\), 69](#)
[min_position \(panoptes.pocs.focuser.simulator.Focuser property\), 71](#)
[min_position\(\) \(panoptes.pocs.focuser.focuser.AbstractFocuser method\), 68](#)
[min_value \(panoptes.pocs.camera.libasi.ControlCaps attribute\), 26](#)
[min_value \(panoptes.pocs.scheduler.observation.base.Observation attribute\), 89](#)
[mode \(panoptes.pocs.camera.sbigudrv.ReadoutInfo attribute\), 45](#)
[model \(panoptes.pocs.camera.camera.AbstractCamera attribute\), 14](#)
[model \(panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property\), 58](#)
[modes \(panoptes.pocs.camera.libasi.SupportedMode attribute\), 29](#)
[module](#)
[panoptes, 9](#)
[panoptes.pocs, 107](#)
[panoptes.pocs.base, 104](#)
[panoptes.pocs.camera, 51](#)
[panoptes.pocs.camera.camera, 13](#)
[panoptes.pocs.camera.fli, 21](#)
[panoptes.pocs.camera.gphoto, 12](#)
[panoptes.pocs.camera.gphoto.base, 9](#)
[panoptes.pocs.camera.gphoto.canon, 11](#)
[panoptes.pocs.camera.gphoto.remote, 11](#)
[panoptes.pocs.camera.libasi, 22](#)
[panoptes.pocs.camera.libfli, 30](#)
[panoptes.pocs.camera.libfli.constants, 35](#)
[panoptes.pocs.camera.sbig, 35](#)
[panoptes.pocs.camera.sbigudrv, 35](#)
[panoptes.pocs.camera.sdk, 50](#)
[panoptes.pocs.camera.simulator, 13](#)
[panoptes.pocs.camera.simulator.ccd, 12](#)
[panoptes.pocs.camera.simulator.dslr, 13](#)
[panoptes.pocs.camera.zwo, 50](#)
[panoptes.pocs.dome, 56](#)
[panoptes.pocs.dome.abstract_serial_dome, 52](#)
[panoptes.pocs.dome.astrohaven, 53](#)
[panoptes.pocs.dome.bisque, 54](#)
[panoptes.pocs.dome.simulator, 55](#)
[panoptes.pocs.filterwheel, 64](#)
[panoptes.pocs.filterwheel.filterwheel, 57](#)
[panoptes.pocs.filterwheel.libefw, 59](#)

panoptes.pocs.filterwheel.sbig, 61
 panoptes.pocs.filterwheel.simulator, 62
 panoptes.pocs.filterwheel.zwo, 63
 panoptes.pocs.focuser, 71
 panoptes.pocs.focuser.astromechanics, 64
 panoptes.pocs.focuser.birger, 65
 panoptes.pocs.focuser.focuser, 66
 panoptes.pocs.focuser.focuslynx, 68
 panoptes.pocs.focuser.serial, 70
 panoptes.pocs.focuser.simulator, 70
 panoptes.pocs.hardware, 105
 panoptes.pocs.images, 106
 panoptes.pocs.mount, 86
 panoptes.pocs.mount.bisque, 75
 panoptes.pocs.mount.ioptron, 73
 panoptes.pocs.mount.ioptron.base, 71
 panoptes.pocs.mount.ioptron.cem40, 72
 panoptes.pocs.mount.ioptron.ieq30pro, 73
 panoptes.pocs.mount.mount, 77
 panoptes.pocs.mount.serial, 83
 panoptes.pocs.mount.simulator, 84
 panoptes.pocs.scheduler, 93
 panoptes.pocs.scheduler.constraint, 90
 panoptes.pocs.scheduler.dispatch, 91
 panoptes.pocs.scheduler.field, 91
 panoptes.pocs.scheduler.observation, 90
 panoptes.pocs.scheduler.observation.base, 87
 panoptes.pocs.scheduler.observation.bias, 89
 panoptes.pocs.scheduler.observation.compound, 90
 panoptes.pocs.scheduler.observation.dark, 90
 panoptes.pocs.scheduler.scheduler, 92
 panoptes.pocs.sensor, 94
 panoptes.pocs.sensor.remote, 93
 panoptes.pocs.state, 98
 panoptes.pocs.state.machine, 96
 panoptes.pocs.state.states, 96
 panoptes.pocs.state.states.default, 96
 panoptes.pocs.state.states.default.analyzing, 94
 panoptes.pocs.state.states.default.housekeeping, 94
 panoptes.pocs.state.states.default.observing, 94
 panoptes.pocs.state.states.default.parked, 95
 panoptes.pocs.state.states.default.parking, 95
 panoptes.pocs.state.states.default.pointing, 95
 panoptes.pocs.state.states.default.ready, 95
 panoptes.pocs.state.states.default.scheduling, 95
 panoptes.pocs.state.states.default.sleeping, 95
 panoptes.pocs.state.states.default.slewing, 95
 panoptes.pocs.state.states.default.tracking, 96
 panoptes.pocs.utils, 104
 panoptes.pocs.utils.cli, 100
 panoptes.pocs.utils.cli.camera, 98
 panoptes.pocs.utils.cli.config, 98
 panoptes.pocs.utils.cli.mount, 99
 panoptes.pocs.utils.cli.notebook, 99
 panoptes.pocs.utils.cli.sensor, 99
 panoptes.pocs.utils.cli.weather, 100
 panoptes.pocs.utils.database, 100
 panoptes.pocs.utils.error, 101
 panoptes.pocs.utils.location, 101
 panoptes.pocs.utils.logger, 102
 panoptes.pocs.utils.plotting, 103
 panoptes.pocs.utils.service, 100
 panoptes.pocs.utils.theskyx, 104
 monitor() (in module panoptes.pocs.utils.cli.sensor), 99
 MONO_BIN (panoptes.pocs.camera.libasi.ControlType attribute), 27
 MoonAvoidance (class in panoptes.pocs.scheduler.constraint), 91
 MOUNT_TIMEOUT (panoptes.pocs.camera.sbigudrv.CFWError attribute), 36
 Mount (class in panoptes.pocs.mount.bisque), 75
 Mount (class in panoptes.pocs.mount.ioptron.base), 71
 Mount (class in panoptes.pocs.mount.ioptron.cem40), 72
 Mount (class in panoptes.pocs.mount.ioptron.ieq30pro), 73
 Mount (class in panoptes.pocs.mount.simulator), 84
 mount (panoptes.pocs.hardware.HardwareName attribute), 105
 mount_is_initialized() (panoptes.pocs.state.machine.PanStateMachine method), 97
 mount_is_tracking() (panoptes.pocs.state.machine.PanStateMachine method), 97
 MountGPS (class in panoptes.pocs.mount.ioptron), 73
 MountHemisphere (class in panoptes.pocs.mount.ioptron), 73
 MountInfo (class in panoptes.pocs.mount.ioptron), 73
 MountMovementSpeed (class in panoptes.pocs.mount.ioptron), 74
 MountState (class in panoptes.pocs.mount.ioptron), 74
 MountTimeSource (class in

`panoptes.pocs.mount.ioptron)`, 74
MountTrackingState (class in `panoptes.pocs.mount.ioptron)`, 75
`move_by()` (`panoptes.pocs.focuser.astromechanics.Focuser` method), 64
`move_by()` (`panoptes.pocs.focuser.birger.Focuser` method), 65
`move_by()` (`panoptes.pocs.focuser.focuser.AbstractFocuser` method), 68
`move_by()` (`panoptes.pocs.focuser.focuslynx.Focuser` method), 69
`move_by()` (`panoptes.pocs.focuser.simulator.Focuser` method), 71
`move_direction()` (`panoptes.pocs.mount.bisque.Mount` method), 76
`move_direction()` (`panoptes.pocs.mount.mount.AbstractMount` method), 80
`move_direction()` (`panoptes.pocs.mount.simulator.Mount` method), 85
MOVE_LISTEN_TIMEOUT (`panoptes.pocs.dome.astrohaven.AstrohavenDome` attribute), 53
MOVE_TIMEOUT (`panoptes.pocs.dome.astrohaven.AstrohavenDome` attribute), 53
`move_to()` (`panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel` method), 58
`move_to()` (`panoptes.pocs.focuser.astromechanics.Focuser` method), 64
`move_to()` (`panoptes.pocs.focuser.birger.Focuser` method), 65
`move_to()` (`panoptes.pocs.focuser.focuser.AbstractFocuser` method), 68
`move_to()` (`panoptes.pocs.focuser.focuslynx.Focuser` method), 69
`move_to()` (`panoptes.pocs.focuser.simulator.Focuser` method), 71
`move_to_dark_position()` (`panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel` method), 59
`move_to_light_position()` (`panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel` method), 59
`movement_speed` (`panoptes.pocs.mount.mount.AbstractMount` property), 80
MOVING (`panoptes.pocs.filterwheel.libefw.ErrorCode` attribute), 61

N
`n_positions` (`panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel` property), 59
`name` (`panoptes.pocs.camera.camera.AbstractCamera` attribute), 14
`name` (`panoptes.pocs.camera.libasi.CameraInfo` attribute), 25

`name` (`panoptes.pocs.camera.libasi.ControlCaps` attribute), 26
`name` (`panoptes.pocs.camera.sbigudrv.GetCCDInfoResults0` attribute), 39
`name` (`panoptes.pocs.camera.sbigudrv.GetDriverInfoResults0` attribute), 41
`name` (`panoptes.pocs.camera.sbigudrv.QueryUSBInfo` attribute), 44
`name` (`panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel` property), 59
`name` (`panoptes.pocs.filterwheel.libefw.EFWInfo` attribute), 61
`name` (`panoptes.pocs.scheduler.observation.base.Observation` property), 89
`next_state` (`panoptes.pocs.state.machine.PanStateMachine` property), 97
`night` (`panoptes.pocs.hardware.HardwareName` attribute), 105
NONE (`panoptes.pocs.camera.libasi.FlipStatus` attribute), 28
NONE (`panoptes.pocs.camera.libasi.TrigOutput` attribute), 29
NONE (`panoptes.pocs.camera.sbigudrv.CFWError` attribute), 36
NORMAL (`panoptes.pocs.camera.libasi.CameraMode` attribute), 26
NORTH (`panoptes.pocs.camera.libasi.GuideDirection` attribute), 29
NORTHERN (`panoptes.pocs.mount.ioptron.MountHemisphere` attribute), 73
NOT_SUPPORTED (`panoptes.pocs.filterwheel.libefw.ErrorCode` attribute), 61
NotSafeError, 101
NotTwilightError, 101
NUM_CLOSE_FEEDBACKS (`panoptes.pocs.dome.astrohaven.AstrohavenDome` attribute), 53

O
Observation (class in `panoptes.pocs.scheduler.observation.base`), 87
Observation (class in `panoptes.pocs.scheduler.observation.compound`), 90
`observation_available()` (`panoptes.pocs.scheduler.scheduler.BaseScheduler` method), 92
`observers` (`panoptes.pocs.scheduler.scheduler.BaseScheduler` property), 92
`observer` (`panoptes.pocs.utils.location.SiteDetails` attribute), 101
OFF (`panoptes.pocs.mount.ioptron.MountGPS` attribute), 73

- OFFSET (*panoptes.pocs.camera.libasi.ControlType attribute*), 27
- OffsetError (*class in panoptes.pocs.images*), 107
- ON (*panoptes.pocs.mount.ioptron.MountGPS attribute*), 73
- on_enter() (*in module panoptes.pocs.state.states.default.analyzing*), 94
- on_enter() (*in module panoptes.pocs.state.states.default.housekeeping*), 94
- on_enter() (*in module panoptes.pocs.state.states.default.observing*), 94
- on_enter() (*in module panoptes.pocs.state.states.default.parked*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.parking*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.pointing*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.ready*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.scheduling*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.sleeping*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.slewing*), 95
- on_enter() (*in module panoptes.pocs.state.states.default.tracking*), 96
- open() (*panoptes.pocs.dome.AbstractDome method*), 57
- open() (*panoptes.pocs.dome.astrohaven.AstrohavenDome method*), 53
- open() (*panoptes.pocs.dome.bisque.Dome method*), 54
- open() (*panoptes.pocs.dome.simulator.Dome method*), 56
- open() (*panoptes.pocs.filterwheel.libefw.EFWDriver method*), 60
- OPEN_A (*panoptes.pocs.dome.astrohaven.Protocol attribute*), 54
- OPEN_B (*panoptes.pocs.dome.astrohaven.Protocol attribute*), 54
- OPEN_BOTH (*panoptes.pocs.dome.astrohaven.Protocol attribute*), 54
- open_camera() (*panoptes.pocs.camera.libasi.ASIDriver method*), 23
- OPEN_DEVICE (*panoptes.pocs.camera.sbigudrv.CFWCommand attribute*), 36
- open_device() (*panoptes.pocs.camera.sbigudrv.SBIGDriver method*), 48
- open_driver() (*panoptes.pocs.camera.sbigudrv.SBIGDriver method*), 48
- OpenDeviceParams (*class in panoptes.pocs.camera.sbigudrv*), 42
- openShutter (*panoptes.pocs.camera.sbigudrv.StartExposureParams2 attribute*), 49
- outLength (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 38
- OUTOF_BOUNDARY (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28
- outPtr (*panoptes.pocs.camera.sbigudrv.CFWParams attribute*), 38
- OVERCLOCK (*panoptes.pocs.camera.libasi.ControlType attribute*), 27
- ## P
- PanBase (*class in panoptes.pocs.base*), 104
- PanLogger (*class in panoptes.pocs.utils.logger*), 102
- panoptes module, 9
- panoptes.pocs module, 107
- panoptes.pocs.base module, 104
- panoptes.pocs.camera module, 51
- panoptes.pocs.camera.camera module, 13
- panoptes.pocs.camera.fli module, 21
- panoptes.pocs.camera.gphoto module, 12
- panoptes.pocs.camera.gphoto.base module, 9
- panoptes.pocs.camera.gphoto.canon module, 11
- panoptes.pocs.camera.gphoto.remote module, 11
- panoptes.pocs.camera.libasi module, 22
- panoptes.pocs.camera.libfli module, 30
- panoptes.pocs.camera.libfliconstants module, 35
- panoptes.pocs.camera.sbig module, 35
- panoptes.pocs.camera.sbigudrv module, 35
- panoptes.pocs.camera.sdk module, 50

panoptes.pocs.camera.simulator	panoptes.pocs.mount.ioptron.base
module, 13	module, 71
panoptes.pocs.camera.simulator.ccd	panoptes.pocs.mount.ioptron.cem40
module, 12	module, 72
panoptes.pocs.camera.simulator.dslr	panoptes.pocs.mount.ioptron.ieg30pro
module, 13	module, 73
panoptes.pocs.camera.zwo	panoptes.pocs.mount.mount
module, 50	module, 77
panoptes.pocs.dome	panoptes.pocs.mount.serial
module, 56	module, 83
panoptes.pocs.dome.abstract_serial_dome	panoptes.pocs.mount.simulator
module, 52	module, 84
panoptes.pocs.dome.astrohaven	panoptes.pocs.scheduler
module, 53	module, 93
panoptes.pocs.dome.bisque	panoptes.pocs.scheduler.constraint
module, 54	module, 90
panoptes.pocs.dome.simulator	panoptes.pocs.scheduler.dispatch
module, 55	module, 91
panoptes.pocs.filterwheel	panoptes.pocs.scheduler.field
module, 64	module, 91
panoptes.pocs.filterwheel.filterwheel	panoptes.pocs.scheduler.observation
module, 57	module, 90
panoptes.pocs.filterwheel.libefw	panoptes.pocs.scheduler.observation.base
module, 59	module, 87
panoptes.pocs.filterwheel.sbig	panoptes.pocs.scheduler.observation.bias
module, 61	module, 89
panoptes.pocs.filterwheel.simulator	panoptes.pocs.scheduler.observation.compound
module, 62	module, 90
panoptes.pocs.filterwheel.zwo	panoptes.pocs.scheduler.observation.dark
module, 63	module, 90
panoptes.pocs.focuser	panoptes.pocs.scheduler.scheduler
module, 71	module, 92
panoptes.pocs.focuser.astromechanics	panoptes.pocs.sensor
module, 64	module, 94
panoptes.pocs.focuser.birger	panoptes.pocs.sensor.remote
module, 65	module, 93
panoptes.pocs.focuser.focuser	panoptes.pocs.state
module, 66	module, 98
panoptes.pocs.focuser.focuslynx	panoptes.pocs.state.machine
module, 68	module, 96
panoptes.pocs.focuser.serial	panoptes.pocs.state.states
module, 70	module, 96
panoptes.pocs.focuser.simulator	panoptes.pocs.state.states.default
module, 70	module, 96
panoptes.pocs.hardware	panoptes.pocs.state.states.default.analyzing
module, 105	module, 94
panoptes.pocs.images	panoptes.pocs.state.states.default.housekeeping
module, 106	module, 94
panoptes.pocs.mount	panoptes.pocs.state.states.default.observing
module, 86	module, 94
panoptes.pocs.mount.bisque	panoptes.pocs.state.states.default.parked
module, 75	module, 95
panoptes.pocs.mount.ioptron	panoptes.pocs.state.states.default.parking
module, 73	module, 95

<code>panoptes.pocs.state.states.default.pointing</code>	<code>PARKED</code>	(<i>panoptes.pocs.mount.ioptron.MountState</i> attribute), 74
<code>module</code> , 95		
<code>panoptes.pocs.state.states.default.ready</code>	<code>path</code>	(<i>panoptes.pocs.scheduler.observation.base.Exposure</i> attribute), 87
<code>module</code> , 95		
<code>panoptes.pocs.state.states.default.scheduling</code>	<code>PATTERN_ADJUST</code>	(<i>panoptes.pocs.camera.libasi.ControlType</i> attribute), 27
<code>module</code> , 95		
<code>panoptes.pocs.state.states.default.sleeping</code>	<code>PINA</code>	(<i>panoptes.pocs.camera.libasi.TrigOutput</i> attribute), 29
<code>module</code> , 95		
<code>panoptes.pocs.state.states.default.slewing</code>	<code>PINB</code>	(<i>panoptes.pocs.camera.libasi.TrigOutput</i> attribute), 29
<code>module</code> , 95		
<code>panoptes.pocs.state.states.default.tracking</code>	<code>pixel_size</code>	(<i>panoptes.pocs.camera.libasi.CameraInfo</i> attribute), 25
<code>module</code> , 96		
<code>panoptes.pocs.utils</code>	<code>pixelHeight</code>	(<i>panoptes.pocs.camera.sbigudrv.ReadoutInfo</i> attribute), 45
<code>module</code> , 104		
<code>panoptes.pocs.utils.cli</code>	<code>pixelLength</code>	(<i>panoptes.pocs.camera.sbigudrv.ReadoutLineParams</i> attribute), 45
<code>module</code> , 100		
<code>panoptes.pocs.utils.cli.camera</code>	<code>pixelStart</code>	(<i>panoptes.pocs.camera.sbigudrv.ReadoutLineParams</i> attribute), 45
<code>module</code> , 98		
<code>panoptes.pocs.utils.cli.config</code>	<code>pixelWidth</code>	(<i>panoptes.pocs.camera.sbigudrv.ReadoutInfo</i> attribute), 45
<code>module</code> , 98		
<code>panoptes.pocs.utils.cli.mount</code>	<code>PocsDB</code>	(class in <i>panoptes.pocs.utils.database</i>), 100
<code>module</code> , 99		
<code>panoptes.pocs.utils.cli.notebook</code>	<code>PocsError</code>	, 101
<code>module</code> , 99		
<code>panoptes.pocs.utils.cli.sensor</code>	<code>pointing_error</code>	(<i>panoptes.pocs.images.Image</i> property), 106
<code>module</code> , 99		
<code>panoptes.pocs.utils.cli.weather</code>	<code>pointing_image</code>	(<i>panoptes.pocs.scheduler.observation.base.Observation</i> property), 89
<code>module</code> , 100		
<code>panoptes.pocs.utils.database</code>	<code>port</code>	(<i>panoptes.pocs.camera.camera.AbstractCamera</i> attribute), 14
<code>module</code> , 100		
<code>panoptes.pocs.utils.error</code>	<code>port</code>	(<i>panoptes.pocs.utils.cli.config.HostInfo</i> attribute), 98
<code>module</code> , 101		
<code>panoptes.pocs.utils.location</code>	<code>position</code>	(<i>panoptes.pocs.dome.bisque.Dome</i> property), 55
<code>module</code> , 101		
<code>panoptes.pocs.utils.logger</code>	<code>position</code>	(<i>panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel</i> property), 59
<code>module</code> , 102		
<code>panoptes.pocs.utils.plotting</code>	<code>position</code>	(<i>panoptes.pocs.filterwheel.sbig.FilterWheel</i> property), 62
<code>module</code> , 103		
<code>panoptes.pocs.utils.service</code>	<code>position</code>	(<i>panoptes.pocs.filterwheel.simulator.FilterWheel</i> property), 63
<code>module</code> , 100		
<code>panoptes.pocs.utils.theskyx</code>	<code>position</code>	(<i>panoptes.pocs.filterwheel.zwo.FilterWheel</i> property), 64
<code>module</code> , 104		
<code>PanStateMachine</code>	<code>position</code>	(<i>panoptes.pocs.focuser.astromechanics.Focuser</i> property), 65
(class in <i>panoptes.pocs.state.machine</i>), 96		
<code>park()</code>	<code>position</code>	(<i>panoptes.pocs.focuser.birger.Focuser</i> property), 65
(<i>panoptes.pocs.dome.bisque.Dome</i> method), 55		
<code>park()</code>	<code>position</code>	(<i>panoptes.pocs.focuser.focuser.AbstractFocuser</i> property), 68
(<i>panoptes.pocs.mount.bisque.Mount</i> method), 76		
<code>park()</code>	<code>position</code>	(<i>panoptes.pocs.focuser.focuslynx.Focuser</i> property), 70
(<i>panoptes.pocs.mount.ioptron.base.Mount</i> method), 71		
<code>park()</code>	<code>power</code>	(<i>panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults</i> attribute), 43
(<i>panoptes.pocs.mount.mount.AbstractMount</i> method), 80		
<code>park()</code>	<code>power</code>	(<i>panoptes.pocs.hardware.HardwareName</i> attribute), 105
(<i>panoptes.pocs.mount.simulator.Mount</i> method), 85		
<code>park_mount()</code>	<code>process_exposure()</code>	(<i>panoptes.pocs.camera.camera.AbstractCamera</i> method), 19
(in <i>panoptes.pocs.utils.cli.mount</i>), 99		

`process_exposure()` (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera* method), 10
`properties` (*panoptes.pocs.camera.camera.AbstractCamera* attribute), 16
`properties` (*panoptes.pocs.camera.sdk.AbstractSDKCamera* property), 50
`Protocol` (class in *panoptes.pocs.dome.astrohaven*), 53
`pulse_guide_off()` (*panoptes.pocs.camera.libasi.ASIDriver* method), 24
`pulse_guide_on()` (*panoptes.pocs.camera.libasi.ASIDriver* method), 24
Q
`QUERY` (*panoptes.pocs.camera.sbigudrv.CFWCommand* attribute), 36
`query()` (*panoptes.pocs.mount.bisque.Mount* method), 76
`query()` (*panoptes.pocs.mount.mount.AbstractMount* method), 81
`query()` (*panoptes.pocs.mount.simulator.Mount* method), 85
`query_temp_status()` (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 48
`QueryCommandStatusParams` (class in *panoptes.pocs.camera.sbigudrv*), 42
`QueryCommandStatusResults` (class in *panoptes.pocs.camera.sbigudrv*), 42
`QueryTemperatureStatusParams` (class in *panoptes.pocs.camera.sbigudrv*), 42
`QueryTemperatureStatusResults` (class in *panoptes.pocs.camera.sbigudrv*), 42
`QueryTemperatureStatusResults2` (class in *panoptes.pocs.camera.sbigudrv*), 43
`QueryUSBInfo` (class in *panoptes.pocs.camera.sbigudrv*), 44
`QueryUSBResults` (class in *panoptes.pocs.camera.sbigudrv*), 44
`QueryUSBResults2` (class in *panoptes.pocs.camera.sbigudrv*), 44
`QueryUSBResults3` (class in *panoptes.pocs.camera.sbigudrv*), 44
R
`RAW16` (*panoptes.pocs.camera.libasi.ImgType* attribute), 29
`RAW8` (*panoptes.pocs.camera.libasi.ImgType* attribute), 29
`read()` (*panoptes.pocs.dome.bisque.Dome* method), 55
`read()` (*panoptes.pocs.mount.bisque.Mount* method), 76
`read()` (*panoptes.pocs.mount.mount.AbstractMount* method), 81
`read()` (*panoptes.pocs.mount.serial.AbstractSerialMount* method), 83
`read()` (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera* method), 85
`read()` (*panoptes.pocs.utils.theskysx.TheSkyX* method), 104
`read_field_list()` (*panoptes.pocs.scheduler.scheduler.BaseScheduler* method), 93
`read_slit_state()` (*panoptes.pocs.dome.bisque.Dome* method), 55
`readiness` (*panoptes.pocs.camera.camera.AbstractCamera* property), 19
`readout()` (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 48
`readout_time` (*panoptes.pocs.camera.camera.AbstractCamera* attribute), 15
`readout_time` (*panoptes.pocs.camera.camera.AbstractCamera* property), 19
`ReadoutInfo` (class in *panoptes.pocs.camera.sbigudrv*), 44
`readoutInfo` (*panoptes.pocs.camera.sbigudrv.GetCCDInfoResults0* attribute), 39
`ReadoutLineParams` (class in *panoptes.pocs.camera.sbigudrv*), 45
`readoutMode` (*panoptes.pocs.camera.sbigudrv.ReadoutLineParams* attribute), 45
`readoutMode` (*panoptes.pocs.camera.sbigudrv.StartExposureParams2* attribute), 49
`readoutMode` (*panoptes.pocs.camera.sbigudrv.StartReadoutParams* attribute), 49
`readoutModes` (*panoptes.pocs.camera.sbigudrv.GetCCDInfoResults0* attribute), 39
`recalibrate()` (*panoptes.pocs.filterwheel.sbig.FilterWheel* method), 62
`recalibrate()` (*panoptes.pocs.filterwheel.zwo.FilterWheel* method), 64
`reconnect()` (*panoptes.pocs.focuser.serial.AbstractSerialFocuser* method), 70
`regulation` (*panoptes.pocs.camera.sbigudrv.SetTemperatureRegulationPa* attribute), 48
`regulation` (*panoptes.pocs.camera.sbigudrv.SetTemperatureRegulationPa* attribute), 49
`RemoteMonitor` (class in *panoptes.pocs.sensor.remote*), 93
`remove_observation()` (*panoptes.pocs.scheduler.scheduler.BaseScheduler* method), 93
`REMOVED` (*panoptes.pocs.filterwheel.libefw.ErrorCode* attribute), 61
`request` (*panoptes.pocs.camera.sbigudrv.GetCCDInfoParams* attribute), 39
`request` (*panoptes.pocs.camera.sbigudrv.GetDriverInfoParams* attribute), 41
`request` (*panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusParams* attribute), 42
`RESET` (*panoptes.pocs.dome.astrohaven.Protocol* at-

tribute), 54

reset() (*panoptes.pocs.scheduler.observation.base.Observation* method), 89

reset_observed_list() (*panoptes.pocs.scheduler.scheduler.BaseScheduler* method), 93

restart() (*in module panoptes.pocs.utils.cli.config*), 98

restart() (*in module panoptes.pocs.utils.cli.notebook*), 99

restart() (*in module panoptes.pocs.utils.cli.weather*), 100

retries (*panoptes.pocs.camera.sbigudrv.SBIGDriver* property), 48

RG (*panoptes.pocs.camera.libasi.BayerPattern* attribute), 24

RGB24 (*panoptes.pocs.camera.libasi.ImgType* attribute), 29

RS232 (*panoptes.pocs.mount.ioptron.MountTimeSource* attribute), 75

run() (*panoptes.pocs.state.machine.PanStateMachine* method), 97

S

SBIGDriver (*class in panoptes.pocs.camera.sbigudrv*), 45

sbigUseOnly (*panoptes.pocs.camera.sbigudrv.EstablishLinkParams* attribute), 39

Scheduler (*class in panoptes.pocs.scheduler.dispatch*), 91

SDKDriver (*class in panoptes.pocs.camera.simulator.ccd*), 12

search_for_home() (*in module panoptes.pocs.utils.cli.mount*), 99

search_for_home() (*panoptes.pocs.mount.ioptron.cem40.Mount* method), 72

search_for_home() (*panoptes.pocs.mount.mount.AbstractMount* method), 81

send_soft_trigger() (*panoptes.pocs.camera.libasi.ASIDriver* method), 24

sensors (*panoptes.pocs.hardware.HardwareName* attribute), 105

seq_time (*panoptes.pocs.scheduler.observation.base.Observation* property), 89

serialNumber (*panoptes.pocs.camera.sbigudrv.GetCCDInfoResults* attribute), 40

serialNumber (*panoptes.pocs.camera.sbigudrv.QueryUSBInfo* attribute), 44

server_running() (*in module panoptes.pocs.utils.cli.config*), 98

set_camera_mode() (*panoptes.pocs.camera.libasi.ASIDriver* method), 24

set_common_properties() (*panoptes.pocs.scheduler.scheduler.BaseScheduler* method), 93

set_config() (*panoptes.pocs.base.PanBase* method), 104

set_control_value() (*panoptes.pocs.camera.libasi.ASIDriver* method), 24

set_direction() (*panoptes.pocs.filterwheel.libefw.EFWDriver* method), 60

set_duration (*panoptes.pocs.scheduler.observation.base.Observation* property), 89

set_handle() (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 48

set_ID() (*panoptes.pocs.camera.libasi.ASIDriver* method), 24

set_is_finished (*panoptes.pocs.scheduler.observation.base.Observation* property), 89

set_park_position() (*in module panoptes.pocs.utils.cli.mount*), 99

set_park_position() (*panoptes.pocs.mount.bisque.Mount* method), 76

set_password() (*in module panoptes.pocs.utils.cli.notebook*), 99

set_position() (*panoptes.pocs.filterwheel.libefw.EFWDriver* method), 60

set_properties() (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoC* method), 10

set_property() (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCam* method), 10

set_roi_format() (*panoptes.pocs.camera.libasi.ASIDriver* method), 24

set_start_position() (*panoptes.pocs.camera.libasi.ASIDriver* method), 24

set_target_coordinates() (*panoptes.pocs.mount.bisque.Mount* method), 76

set_target_coordinates() (*panoptes.pocs.mount.ioptron.cem40.Mount* method), 72

set_target_coordinates() (*panoptes.pocs.mount.mount.AbstractMount* method), 81

set_temp_regulation() (*panoptes.pocs.camera.sbigudrv.SBIGDriver* method), 48

set_tracking_rate() (*panoptes.pocs.mount.mount.AbstractMount* method), 82

set_tracking_rate() (*panoptes.pocs.mount.serial.AbstractSerialMount* method), 83

set_tracking_rate() (*panoptes.pocs.mount.simulator.Mount* method), 83

method), 85

set_trigger_output_io_conf()
(panoptes.pocs.camera.libasi.ASIDriver
method), 24

set_value() (in module panoptes.pocs.utils.cli.config),
98

SetDriverControlParams (class in
panoptes.pocs.camera.sbigudrv), 48

SetDriverHandleParams (class in
panoptes.pocs.camera.sbigudrv), 48

SetTemperatureRegulationParams (class in
panoptes.pocs.camera.sbigudrv), 48

SetTemperatureRegulationParams2 (class in
panoptes.pocs.camera.sbigudrv), 48

setup() (in module panoptes.pocs.utils.cli.config), 98

setup_mount() (in module
panoptes.pocs.utils.cli.mount), 99

SIDEREAL (panoptes.pocs.mount.ioptron.MountTrackingState
attribute), 75

SIDEREAL_1 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_128 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_16 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_2 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_256 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_512 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_64 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_8 (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SIDEREAL_MAX (panoptes.pocs.mount.ioptron.MountMovementSpeed
attribute), 74

SiteDetails (class in panoptes.pocs.utils.location), 101

slew_to_coordinates()
(panoptes.pocs.mount.mount.AbstractMount
method), 82

slew_to_home() (panoptes.pocs.mount.bisque.Mount
method), 76

slew_to_home() (panoptes.pocs.mount.mount.AbstractMount
method), 82

slew_to_home() (panoptes.pocs.mount.simulator.Mount
method), 85

slew_to_target() (panoptes.pocs.mount.bisque.Mount
method), 77

slew_to_target() (panoptes.pocs.mount.mount.AbstractMount
method), 82

slew_to_target() (panoptes.pocs.mount.simulator.Mount
method), 86

slew_to_zero() (panoptes.pocs.mount.bisque.Mount
method), 77

slew_to_zero() (panoptes.pocs.mount.mount.AbstractMount
method), 83

SLEWING (panoptes.pocs.mount.ioptron.MountState
attribute), 74

slot_num (panoptes.pocs.filterwheel.libfw.EFWInfo
attribute), 61

SOLAR (panoptes.pocs.mount.ioptron.MountTrackingState
attribute), 75

solve_field() (panoptes.pocs.images.Image
method), 107

SOUTH (panoptes.pocs.camera.libasi.GuideDirection
attribute), 29

SOUTHERN (panoptes.pocs.mount.ioptron.MountHemisphere
attribute), 73

STABLE_STATES (panoptes.pocs.dome.astrohaven.Protocol
attribute), 54

start() (in module panoptes.pocs.utils.cli.notebook), 99

start_exposure() (panoptes.pocs.camera.libasi.ASIDriver
method), 24

start_exposure() (panoptes.pocs.camera.sbigudrv.SBIGDriver
method), 48

start_tether() (panoptes.pocs.camera.gphoto.base.AbstractGPhotoCam
method), 10

start_video() (panoptes.pocs.camera.zwo.Camera
method), 51

start_video_capture()
(panoptes.pocs.camera.libasi.ASIDriver
method), 24

StartExposureParams2 (class in
panoptes.pocs.camera.sbigudrv), 49

StartReadoutParams (class in
panoptes.pocs.camera.sbigudrv), 49

State (panoptes.pocs.mount.mount.AbstractMount prop-
erty), 83

status (panoptes.pocs.camera.sbigudrv.QueryCommandStatusResults
attribute), 42

status (panoptes.pocs.dome.AbstractDome
property), 57

status (panoptes.pocs.dome.astrohaven.AstrohavenDome
property), 53

status (panoptes.pocs.dome.bisque.Dome
property), 55

status (panoptes.pocs.dome.simulator.Dome
property), 56

status (panoptes.pocs.mount.mount.AbstractMount
property), 83

status (panoptes.pocs.scheduler.observation.base.Observation
property), 89

status (panoptes.pocs.scheduler.scheduler.BaseScheduler
property), 93

status() (in module panoptes.pocs.utils.cli.config), 98

status() (in module panoptes.pocs.utils.cli.weather),
100

stop_exposure() (panoptes.pocs.camera.libasi.ASIDriver
method), 24

method), 24
 stop_slew() (panoptes.pocs.mount.simulator.Mount method), 86
 stop_states() (panoptes.pocs.state.machine.PanStateMachine method), 97
 stop_video() (panoptes.pocs.camera.zwo.Camera method), 51
 stop_video_capture() (panoptes.pocs.camera.libasi.ASIDriver method), 24
 STOPPED (panoptes.pocs.mount.ioptron.MountState attribute), 74
 SUCCESS (panoptes.pocs.camera.libasi.ErrorCode attribute), 28
 SUCCESS (panoptes.pocs.camera.libasi.ExposureStatus attribute), 28
 SUCCESS (panoptes.pocs.filterwheel.libefw.ErrorCode attribute), 61
 supported_bins (panoptes.pocs.camera.libasi.CameraInfo attribute), 25
 supported_video_format (panoptes.pocs.camera.libasi.CameraInfo attribute), 25
 SupportedMode (class in panoptes.pocs.camera.libasi), 29

T

take_exposure() (panoptes.pocs.camera.camera.AbstractCamera method), 19
 take_observation() (panoptes.pocs.camera.camera.AbstractCamera method), 20
 take_observation() (panoptes.pocs.camera.simulator.ds9c.Camera method), 13
 take_pictures() (in module panoptes.pocs.utils.cli.camera), 98
 TARGET_TEMP (panoptes.pocs.camera.libasi.ControlType attribute), 27
 target_temperature (panoptes.pocs.camera.camera.AbstractCamera attribute), 14
 target_temperature (panoptes.pocs.camera.camera.AbstractCamera property), 20
 target_temperature (panoptes.pocs.camera.fli.Camera property), 21
 target_temperature (panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera property), 10
 target_temperature (panoptes.pocs.camera.sbig.Camera property), 35
 target_temperature (panoptes.pocs.camera.simulator.ccd.Camera property), 12
 target_temperature (panoptes.pocs.camera.zwo.Camera property), 51
 temperature (panoptes.pocs.camera.camera.AbstractCamera attribute), 14
 temperature (panoptes.pocs.camera.camera.AbstractCamera property), 20
 temperature (panoptes.pocs.camera.fli.Camera property), 21
 temperature (panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera property), 10
 TEMPERATURE (panoptes.pocs.camera.libasi.ControlType attribute), 27
 temperature (panoptes.pocs.camera.sbig.Camera property), 35
 temperature (panoptes.pocs.camera.simulator.ccd.Camera property), 12
 temperature (panoptes.pocs.camera.zwo.Camera property), 51
 temperature (panoptes.pocs.focuser.focuslynx.Focuser property), 70
 temperature_tolerance (panoptes.pocs.camera.camera.AbstractCamera attribute), 15
 temperature_tolerance (panoptes.pocs.camera.camera.AbstractCamera property), 21
 TheSkyX (class in panoptes.pocs.utils.theskyyx), 104
 theskyyx (panoptes.pocs.hardware.HardwareName attribute), 105
 timeout (panoptes.pocs.camera.camera.AbstractCamera attribute), 15
 TIMEOUT (panoptes.pocs.camera.libasi.ErrorCode attribute), 28
 to_dict() (panoptes.pocs.scheduler.observation.base.Observation method), 89
 top (panoptes.pocs.camera.sbigudrv.StartExposureParams2 attribute), 49
 top (panoptes.pocs.camera.sbigudrv.StartReadoutParams attribute), 49
 TRACKING (panoptes.pocs.mount.ioptron.MountState attribute), 74
 TRACKING_PEC (panoptes.pocs.mount.ioptron.MountState attribute), 74
 tracking_rate (panoptes.pocs.mount.mount.AbstractMount property), 83
 trackingCCDPower (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 43
 trackingCCDSecPoint (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 44
 trackingCCDTemperature (panoptes.pocs.camera.sbigudrv.QueryTemperatureStatusResults attribute), 44
 TRIG_FALL_EDGE (panoptes.pocs.camera.libasi.CameraMode attribute), 26
 TRIG_HIGH_LEVEL (panoptes.pocs.camera.libasi.CameraMode attribute), 26
 TRIG_LOW_LEVEL (panoptes.pocs.camera.libasi.CameraMode attribute), 26

attribute), 26

TRIG_RISE_EDGE (*panoptes.pocs.camera.libasi.CameraMode attribute*), 26

TRIG_SOFT_EDGE (*panoptes.pocs.camera.libasi.CameraMode attribute*), 26

TRIG_SOFT_LEVEL (*panoptes.pocs.camera.libasi.CameraMode attribute*), 26

TrigOutput (*class in panoptes.pocs.camera.libasi*), 29

U

uid (*panoptes.pocs.camera.camera.AbstractCamera attribute*), 14

uid (*panoptes.pocs.camera.camera.AbstractCamera property*), 21

uid (*panoptes.pocs.camera.gphoto.base.AbstractGPhotoCamera property*), 11

uid (*panoptes.pocs.filterwheel.filterwheel.AbstractFilterWheel property*), 59

uid (*panoptes.pocs.focuser.focuser.AbstractFocuser property*), 68

uid (*panoptes.pocs.focuser.focuslynx.Focuser property*), 70

UNKNOWN (*panoptes.pocs.camera.sbigudrv.CFWModelSelect attribute*), 37

UNKNOWN (*panoptes.pocs.camera.sbigudrv.CFWStatus attribute*), 38

UNKNOWN (*panoptes.pocs.mount.ioptron.MountState attribute*), 74

unpark() (*panoptes.pocs.dome.bisque.Dome method*), 55

unpark() (*panoptes.pocs.mount.bisque.Mount method*), 77

unpark() (*panoptes.pocs.mount.mount.AbstractMount method*), 83

unpark() (*panoptes.pocs.mount.simulator.Mount method*), 86

unused (*panoptes.pocs.camera.libasi.CameraInfo attribute*), 25

unused (*panoptes.pocs.camera.libasi.ControlCaps attribute*), 26

update_status() (*panoptes.pocs.mount.mount.AbstractMount method*), 83

url (*panoptes.pocs.utils.cli.config.HostInfo property*), 98

usbInfo (*panoptes.pocs.camera.sbigudrv.QueryUSBResults attribute*), 44

usbInfo (*panoptes.pocs.camera.sbigudrv.QueryUSBResults2 attribute*), 44

usbInfo (*panoptes.pocs.camera.sbigudrv.QueryUSBResults3 attribute*), 44

verify_connected() (*panoptes.pocs.dome.abstract_serial_dome.Abstract method*), 52

version (*panoptes.pocs.camera.sbigudrv.GetDriverInfoResults0 attribute*), 41

version (*panoptes.pocs.camera.sdk.AbstractSDKDriver property*), 50

VERT (*panoptes.pocs.camera.libasi.FlipStatus attribute*), 28

VIDEO_MODE_ACTIVE (*panoptes.pocs.camera.libasi.ErrorCode attribute*), 28

W

waiting_for_readout (*panoptes.pocs.camera.camera.AbstractCamera property*), 21

WB_B (*panoptes.pocs.camera.libasi.ControlType attribute*), 27

WB_R (*panoptes.pocs.camera.libasi.ControlType attribute*), 27

wcs_file (*panoptes.pocs.images.Image property*), 107

weather (*panoptes.pocs.hardware.HardwareName attribute*), 105

WEST (*panoptes.pocs.camera.libasi.GuideDirection attribute*), 29

width (*panoptes.pocs.camera.sbigudrv.ReadoutInfo attribute*), 45

width (*panoptes.pocs.camera.sbigudrv.StartExposureParams attribute*), 49

width (*panoptes.pocs.camera.sbigudrv.StartReadoutParams attribute*), 50

WORKING (*panoptes.pocs.camera.libasi.ExposureStatus attribute*), 28

write() (*panoptes.pocs.dome.bisque.Dome method*), 55

write() (*panoptes.pocs.mount.bisque.Mount method*), 77

write() (*panoptes.pocs.mount.mount.AbstractMount method*), 83

write() (*panoptes.pocs.mount.serial.AbstractSerialMount method*), 84

write() (*panoptes.pocs.mount.simulator.Mount method*), 86

write() (*panoptes.pocs.utils.theskyx.TheSkyX method*), 104

write_fits() (*panoptes.pocs.camera.camera.AbstractCamera method*), 21

Y

Y8 (*panoptes.pocs.camera.libasi.ImgType attribute*), 29

V

verbose (*panoptes.pocs.utils.cli.config.HostInfo attribute*), 98