pyastrobackend

Release 0.1

Michael Fulbright

Nov 01, 2020

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 What is pyastrobackend?

Pyastrobackend is a abstraction later which presents a singular API to Python applications allowing the use of ASCOM, Alpaca or INDI device driver frameworks underneath. The goal is allow one to have a single source tree for an astronomical application and be able to run it on a system using any of these hardware frameworks.

1.2 How does it work?

An application first determines which "backend" (for example, ASCOM or INDI) is required for the given system using the `get_backend_for_os()` function.

Then the application imports the appropriate backend and device control modules for the system.

Once the backend and devices are connected then all api calls are uniform between the ASCOM and INDI implementations. This allows a single code base to work on both.

1.3 Examples

Here is an exmaple of loading the appropriate backend and camera drivers:

```
from pyastrobackend.BackendConfig import get_backend, get_backend_for_os

backend_name = get_backend_for_os()
backend = get_backend(backend_name)
```

Later the backend and camera objects are created and the backend connected using:

```
rc = backend.connect()
if not rc:
    logging.error('Failed to connect to backend!')
    sys.exit(-1)

cam = backend.newCamera()
```

Finally the camera driver is connected using:

```
rc = cam.connect(camera_driver)
if not rc:
```

(continues on next page)

(continued from previous page)

```
    logging.error('Failed to connect to camera driver {camera_driver}!')
    sys.exit(-1)
```

Now the camera object is ready and can be used to takes images, etc.

CHAPTER

TWO

PYASTROBACKEND

2.1 pyastrobackend package

2.1.1 Subpackages

pyastrobackend.ASCOM package

Submodules

pyastrobackend.ASCOM.Camera module

pyastrobackend.ASCOM.FilterWheel module

Pure ASCOM solution

class pyastrobackend.ASCOM.FilterWheel.**FilterWheel**(*backend=None*)

Bases: *pyastrobackend.BaseBackend.BaseFilterWheel*

connect(*name*)

Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

disconnect()

Disconnect from device.

get_names()

Get names of all filter positions.

Returns List of filter names.

Return type list

get_num_positions()

Get number of filter positions.

Returns Number of filter positions

Return type int

get_position()

Get position of filter wheel. First position is 0!

Returns Position of filter wheel.

Return type int

get_position_name()
Get name of filter at current position.

Returns Name of current filter.

Return type str

has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

is_moving()
Check if filter wheel is moving.

Returns True if filter wheel is moving.

Return type bool

set_position(*pos*)
Sends request to driver to move filter wheel position

This DOES NOT wait for filter to move into position!

Use is_moving() method to check if its done.

set_position_name(*name*)
Sends request to driver to move filter wheel position

This DOES NOT wait for filter to move into position!

Use is_moving() method to check if its done.

show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

pyastrobackend.ASCOM.Focuser module

Pure ASCOM solution

class pyastrobackend.ASCOM.Focuser.**Focuser**(*backend=None*)
Bases: *pyastrobackend.BaseBackend.BaseFocuser*

connect(*name*)
Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

disconnect()
Disconnect from device.

get_absolute_position()
Get absolute position of focuser.

Returns Absolute position of focuser.

Return type int

get_current_temperature()
Get temperature from focuser.

Returns Temperature (C).

Return type float

get_max_absolute_position()
Get maximum possible absolute position of focuser.

Returns Absolute maximum possible position of focuser.

Return type int

has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

is_moving()
Check if focuser is moving.

Returns True if focuser is moving.

Return type bool

move_absolute_position(*abspos*)
Move focuser to absolute position.

Parameters **abspos** – Target position for focuser.

Returns True on success.

Return type bool

show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

stop()
Stop focuser motion..

Returns True on success.

Return type bool

pyastrobackend.ASCOM.Mount module

Pure ASCOM solution

class pyastrobackend.ASCOM.Mount.**Mount**(*backend=None*)
Bases: *pyastrobackend.BaseBackend.BaseMount*

abort_slew()
Abort slew.

Returns True on success.

Return type bool

can_park()
Test if a mount can park.

Returns True if mount can park.

Return type bool

connect(*name*)
Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

disconnect()
Disconnect from device.

get_pier_side()
Returns backend specific pier side information. **NOTE: NOT** recommended for use as ASCOM and INDI may give different results for different drivers - not tested extensively at all so use with caution.

Returns 'EAST', 'WEST' or None if unknown.

get_position_altaz()
Returns tuple of (alt, az) in degrees

get_position_radec()
Returns tuple of (ra, dec) with ra in decimal hours and dec in degrees

get_side_physical()
Get physical side of mount. **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

get_side_pointing()
Get side of meridian where mount is pointing. **NOTE** may not be same as result from get_side_physical() if counterweights are pointing up, etc! **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

get_tracking()
Get mount tracking state.

Returns True if tracking.

Return type bool

has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

is_parked()
Test if mount is parked.

Returns True if mount is parked.

Return type bool

is_slewing()
Test if mount is slewing.

Returns True if mount is slewing.

Return type bool

park()
Park mount.

Returns True on success.

Return type bool

set_tracking(*onoff*)
Enable/disable mount tracking.

Parameters **onoff** – Flag to turn tracking on/off.

Returns True on success.

Return type bool

show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

slew(*ra*, *dec*)
Slew to ra/dec with ra in decimal hours and dec in degrees

sync(*ra*, *dec*)
Sync to ra/dec with ra in decimal hours and dec in degrees

unpark()
Unark mount.

Returns True on success.

Return type bool

class pyastrobackend.ASCOM.Mount.**PierSide**(*value*)
Bases: enum.Enum

An enumeration.

EAST = 0

UNKNOWN = -1

WEST = 1

Module contents

pyastrobackend.Alpaca package

Submodules

pyastrobackend.Alpaca.AlpacaDevice module

class pyastrobackend.Alpaca.AlpacaDevice.**AlpacaDevice**
Bases: object

connect(*name*)

disconnect()

get_prop(*prop*, *params={}*, *returndict=False*)

has_chooser()

is_connected()

set_prop(*prop*, *params={}*)

show_chooser(*last_choice*)

pyastrobackend.Alpaca.Camera module

pyastrobackend.Alpaca.FilterWheel module

class pyastrobackend.Alpaca.FilterWheel.**FilterWheel**(*backend*)

Bases: *pyastrobackend.Alpaca.AlpacaDevice.AlpacaDevice*, *pyastrobackend.BaseBackend.BaseFilterWheel*

get_names()

Get names of all filter positions.

Returns List of filter names.

Return type list

get_num_positions()

Get number of filter positions.

Returns Number of filter positions

Return type int

get_position()

Get position of filter wheel. First position is 0!

Returns Position of filter wheel.

Return type int

get_position_name()

Get name of filter at current position.

Returns Name of current filter.

Return type str

is_moving()

Check if filter wheel is moving.

Returns True if filter wheel is moving.

Return type bool

set_position(*pos*)

Sends request to driver to move filter wheel position

This DOES NOT wait for filter to move into position!

Use is_moving() method to check if its done.

set_position_name(*name*)

Sends request to driver to move filter wheel position

This DOES NOT wait for filter to move into position!

Use is_moving() method to check if its done.

pyastrobackend.Alpaca.Focuser module

class pyastrobackend.Alpaca.Focuser.**Focuser**(*backend*)
Bases: *pyastrobackend.Alpaca.AlpacaDevice.AlpacaDevice*, *pyastrobackend.BaseBackend.BaseFocuser*

get_absolute_position()
Get absolute position of focuser.

Returns Absolute position of focuser.

Return type int

get_current_temperature()
Get temperature from focuser.

Returns Temperature (C).

Return type float

get_max_absolute_position()
Get maximum possible absolute position of focuser.

Returns Absolute maximum possible position of focuser.

Return type int

is_moving()
Check if focuser is moving.

Returns True if focuser is moving.

Return type bool

move_absolute_position(*abspos*)
Move focuser to absolute position.

Parameters **abspos** – Target position for focuser.

Returns True on success.

Return type bool

stop()
Stop focuser motion..

Returns True on success.

Return type bool

pyastrobackend.Alpaca.Mount module

class pyastrobackend.Alpaca.Mount.**Mount**(*backend*)
Bases: *pyastrobackend.Alpaca.AlpacaDevice.AlpacaDevice*, *pyastrobackend.BaseBackend.BaseMount*

abort_slew()
Abort slew.

Returns True on success.

Return type bool

can_park()
Test if a mount can park.

Returns True if mount can park.

Return type bool

get_pier_side()
Returns backend specific pier side information. **NOTE: NOT** recommended for use as ASCOM and INDI may give different results for different drivers - not tested extensively at all so use with caution.

Returns 'EAST', 'WEST' or None if unknown.

get_position_altaz()
Returns tuple of (alt, az) in degrees

get_position_radec()
Returns tuple of (ra, dec) with ra in decimal hours and dec in degrees

get_side_physical()
Get physical side of mount. **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

get_side_pointing()
Get side of meridian where mount is pointing. **NOTE** may not be same as result from get_side_physical() if counterweights are pointing up, etc! **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

get_tracking()
Get mount tracking state.

Returns True if tracking.

Return type bool

is_parked()
Test if mount is parked.

Returns True if mount is parked.

Return type bool

is_slewing()
Test if mount is slewing.

Returns True if mount is slewing.

Return type bool

park()
Park mount.

Returns True on success.

Return type bool

set_tracking(*onoff*)
Enable/disable mount tracking.

Parameters **onoff** – Flag to turn tracking on/off.

Returns True on success.

Return type bool

slew(*ra*, *dec*)
Slew to ra/dec with ra in decimal hours and dec in degrees

sync(*ra*, *dec*)
Sync to ra/dec with ra in decimal hours and dec in degrees

unpark()
Unark mount.

Returns True on success.

Return type bool

Module contents

pyastrobackend.INDI package

Submodules

pyastrobackend.INDI.IndiHelper module

pyastrobackend.INDI.IndiHelper.**connectDevice**(*indiclient*, *devicename*, *timeout=2*)

pyastrobackend.INDI.IndiHelper.**dump_Device**(*dev*)

pyastrobackend.INDI.IndiHelper.**dump_INumberVectorProperty**(*p*)

pyastrobackend.INDI.IndiHelper.**dump_ISwitchVectorProperty**(*p*)

pyastrobackend.INDI.IndiHelper.**dump_ITextVectorProperty**(*p*)

pyastrobackend.INDI.IndiHelper.**dump_Number**(*p*)

pyastrobackend.INDI.IndiHelper.**dump_Property**(*p*)

pyastrobackend.INDI.IndiHelper.**dump_PropertyVector**(*pv*)

pyastrobackend.INDI.IndiHelper.**findDeviceInterfaces**(*indidevice*)

pyastrobackend.INDI.IndiHelper.**findDeviceName**(*indidevice*)

pyastrobackend.INDI.IndiHelper.**findDevices**(*indiclient*, *timeout=2*)

pyastrobackend.INDI.IndiHelper.**findDevicesByClass**(*indiclient*, *device_class*)
class can be 'ccd', 'filter', 'focuser', 'guider', 'telescope'

pyastrobackend.INDI.IndiHelper.**findLight**(*ilvect*, *name*)

pyastrobackend.INDI.IndiHelper.**findNumber**(*invect*, *name*)

pyastrobackend.INDI.IndiHelper.**findSwitch**(*iswvect*, *name*)

pyastrobackend.INDI.IndiHelper.**findText**(*itvect*, *name*)

pyastrobackend.INDI.IndiHelper.**getLight**(*device*, *name*, *timeout=0.5*)

pyastrobackend.INDI.IndiHelper.**getNumber**(*device*, *name*, *timeout=0.5*)

pyastrobackend.INDI.IndiHelper.**getNumberState**(*device*, *propname*)

pyastrobackend.INDI.IndiHelper.**getSwitch**(*device*, *name*, *timeout=0.5*)

pyastrobackend.INDI.IndiHelper.**getText**(*device*, *name*, *timeout=0.5*)

pyastrobackend.INDI.IndiHelper.**getfindLight**(*device*, *propname*, *lightname*)

pyastrobackend.INDI.IndiHelper.**getfindLightState**(*device*, *propname*, *lightname*)

pyastrobackend.INDI.IndiHelper.**getfindNumber**(*device*, *propname*, *numname*)
Combines getNumber() and findNumber()

pyastrobackend.INDI.IndiHelper.**getfindNumberValue**(*device*, *propname*, *numname*)
Combines getNumber() and findNumber()

pyastrobackend.INDI.IndiHelper.**getfindSwitch**(*device*, *propname*, *swname*)

pyastrobackend.INDI.IndiHelper.**getfindSwitchState**(*device*, *propname*, *swname*)

pyastrobackend.INDI.IndiHelper.**getfindText**(*device*, *propname*, *txtname*)

pyastrobackend.INDI.IndiHelper.**getfindTextText**(*device*, *propname*, *txtname*)

pyastrobackend.INDI.IndiHelper.**setfindLightState**(*indiclient*, *device*, *propname*, *lightname*, *state*)

pyastrobackend.INDI.IndiHelper.**setfindNumberValue**(*indiclient*, *device*, *propname*, *numname*, *value*)

pyastrobackend.INDI.IndiHelper.**setfindSwitchState**(*indiclient*, *device*, *propname*, *swname*, *onoff*)

pyastrobackend.INDI.IndiHelper.**setfindTextText**(*indiclient*, *device*, *propname*, *txtname*, *value*)

pyastrobackend.INDI.IndiHelper.**strGetType**(*o*)

pyastrobackend.INDI.IndiHelper.**strIPState**(*s*)

pyastrobackend.INDI.IndiHelper.**strISState**(*s*)

Module contents

pyastrobackend.RPC package

Submodules

pyastrobackend.RPC.Camera module

RPC Camera solution

class pyastrobackend.RPC.Camera.**Camera**(*backend=None*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDevice*, *pyastrobackend.BaseBackend.BaseCamera*

check_exposure()
Check if exposure is complete.

Returns True if exposure complete.

Return type bool

check_exposure_success()
Check if exposure was successful - only valid if check_exposure() returns True.

Returns True if exposure complete.

Return type bool

event_callback(*event*, **args*)

get_binning()
Return pixel binning.

Returns A tuple containing the X and Y binning.

Return type (int, int)

get_camera_description()
Get the camera name - result depends on backend in use.

Returns Description for camera device.

Return type str

get_camera_gain()
Return gain for camera (not all cameras support).

Returns Camera gain

Return type float

get_camera_name()
Get the camera name - result depends on backend in use.

Returns Name of camera device.

Return type str

get_cooler_power()
Get cooler power use (percentage of maximum).

Returns Cooler power level.

Return type float

get_cooler_state()
Get cooler state.

Returns True if cooler is on.

Return type bool

get_current_temperature()
Return current camera temperature.

Returns Temperature (C)

Return type float

get_driver_info()
Get information about camera - result depends on backend in use.

Returns Driver information about camera device.

Return type str

get_driver_version()
Get version information about camera - result depends on backend in use.

Returns Driver version information.

Return type str

get_egain()
Return gain for camera as e- per ADU.

Returns Camera gain

Return type float

get_exposure_progress()
Get percentage completion of exposure. Use *supports_progress()* to test if driver supports this call.

Returns Percentage completion of exposure.

Return type int

get_frame()
Return region of interest (ROI) for image capture.

Returns A tuple containing the upper left (X, Y) for ROI and width/height.

Return type (int, int, int, int)

get_image_data()
Return image data from last image taken.

Returns Image data or None if not available.

get_max_binning()
Return maximum pixel binning supported.

Returns Maximum binning value.

Return type int

get_pixelsize()
Return pixel size for camera sensor.

Returns A tuple containing the X and Y pixel sizes.

Return type (float, float)

get_settings()
Returns most settings for camera as dict. Useful for RPC drivers to reduce round trips.

Following keys (not all will get values on all drivers):

- binning: (tuple) X, Y binning
- framesize: (tuple) Width, height of sensor
- roi: (tuple) Upper left corner and width, height of roi
- pixelsize: (tuple) X, Y pixel size
- egain: (float) Gain of camera in e-/ADU
- camera_gain: (float) Internal gain of camera
- camera_offset: (float) Internal offset of camera
- camera_usbbandwidth: (int) Internal USB traffic settings of camera
- camera_current_temperature: (float) Current temperature of camera
- camera_target_temperature: (float) Target temperature of camera
- cooler_state: (bool) Cooler on/off status
- cooler_power: (float) Power (0-100%) level of cooler

Returns Dictionary of settings

Return type dict

get_size()
Return size of sensor in pixels.

Returns A tuple containing the X and Y pixel sizes.

Return type (int, int)

get_state()
Get camera state.

Returns

-1 Camera State unknown

0 Camera idle

2 Camera is busy (exposing)

5 Camera error

Rtype int

get_target_temperature()
Return current target camera cooler temperature.

Returns Target cooler temperature (C)

Return type float

save_image_data(*path*, *overwrite=False*)
Save image data from last image taken to file. NOTE: Not availale for all backends - check with supports_saveimage().

Parameters **filename** – Filename for output file

Ptype str

Returns Image data or None if not available.

set_binning(*binx*, *biny*)
Set pixel binning.

Parameters

- **binx** – X binning
- **biny** – Y binning

Returns True on success.

Return type bool

set_camera_gain(*gain*)
Set gain for camera (not all cameras support).

Returns True on success.

Return type bool

set_cooler_state(*onoff*)
Set cooler on or off

Parameters **onff** – True to turn on camera.

Returns True on success.

Return type bool

set_frame(*minx*, *miny*, *width*, *height*)
Set region of interest (ROI) for image capture.

Parameters

- **minx** – Leftmost extent of ROI.
- **miny** – Uppermost extent of ROI.
- **width** – Width of ROI.
- **height** – Height of ROI.

Returns True on success.

Return type bool

set_target_temperature(*temp_c*)
Set target camera cooler temperature.

Parameters **temp_c** – Target cooler temperature (C)

Returns True on success.

Return type bool

start_exposure(*expos*)
Start an exposure.

Parameters **expos** – Exposure length (seconds)

Ptype float

Returns True on success.

Return type bool

stop_exposure()
Stop exposure.

Returns True on success.

Return type bool

supports_progress()
Check if exposure progress is supported.

Returns True if progress info is available.

Return type bool

supports_saveimage()
Test if drive has 'saveimage' method.

Returns True if available, False otherwise.

Return type bool

class pyastrobackend.RPC.Camera.**RPCCameraThread**(*port*, *user_data*, **args*, ***kwargs*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDeviceThread*

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

pyastrobackend.RPC.FilterWheel module

RPC FilterWheel solution

class pyastrobackend.RPC.FilterWheel.**FilterWheel**(*backend=None*)

Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDevice*, *pyastrobackend.BaseBackend.BaseFilterWheel*

event_callback(*event*, **args*)

get_names()

Get names of all filter positions.

Returns List of filter names.

Return type list

get_num_positions()

Get number of filter positions.

Returns Number of filter positions

Return type int

get_position()

Get position of filter wheel. First position is 0!

Returns Position of filter wheel.

Return type int

get_position_name()

Get name of filter at current position.

Returns Name of current filter.

Return type str

is_moving()

Check if filter wheel is moving.

Returns True if filter wheel is moving.

Return type bool

set_position(*pos*)

Sends request to driver to move filter wheel position

This DOES NOT wait for filter to move into position!

Use is_moving() method to check if its done.

set_position_name(*name*)
Sends request to driver to move filter wheel position

This DOES NOT wait for filter to move into position!

Use is_moving() method to check if its done.

class pyastrobackend.RPC.FilterWheel.**RPCFilterWheelThread**(*port*, *user_data*, **args*, ***kwargs*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDeviceThread*

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

pyastrobackend.RPC.Focuser module

RPC Focuser solution

class pyastrobackend.RPC.Focuser.**Focuser**(*backend=None*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDevice*, *pyastrobackend.BaseBackend.BaseFocuser*

event_callback(*event*, **args*)

get_absolute_position()
Get absolute position of focuser.

Returns Absolute position of focuser.

Return type int

get_current_temperature()
Get temperature from focuser.

Returns Temperature (C).

Return type float

get_max_absolute_position()
Get maximum possible absolute position of focuser.

Returns Absolute maximum possible position of focuser.

Return type int

is_moving()
Check if focuser is moving.

Returns True if focuser is moving.

Return type bool

move_absolute_position(*abspos*)
Move focuser to absolute position.

Parameters **abspos** – Target position for focuser.

Returns True on success.

Return type bool

stop()
Stop focuser motion..

Returns True on success.

Return type bool

class pyastrobackend.RPC.Focuser.**RPCFocuserThread**(*port*, *user_data*, **args*, ***kwargs*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDeviceThread*

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

pyastrobackend.RPC.Mount module

RPC Mount solution

class pyastrobackend.RPC.Mount.**Mount**(*backend=None*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDevice*, *pyastrobackend.BaseBackend.BaseMount*

abort_slew()
Abort slew.

Returns True on success.

Return type bool

can_park()
Test if a mount can park.

Returns True if mount can park.

Return type bool

event_callback(*event*, **args*)

get_pier_side()
Returns backend specific pier side information. **NOTE: NOT** recommended for use as ASCOM and INDI may give different results for different drivers - not tested extensively at all so use with caution.

Returns 'EAST', 'WEST' or None if unknown.

get_position_altaz()
Returns tuple of (alt, az) in degrees

get_position_radec()
Returns tuple of (ra, dec) with ra in decimal hours and dec in degrees

get_side_physical()
Get physical side of mount. **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

get_side_pointing()
Get side of meridian where mount is pointing. **NOTE** may not be same as result from get_side_physical() if counterweights are pointing up, etc! **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

get_tracking()
Get mount tracking state.

Returns True if tracking.

Return type bool

is_parked()
Test if mount is parked.

Returns True if mount is parked.

Return type bool

is_slewing()
Test if mount is slewing.

Returns True if mount is slewing.

Return type bool

park()
Park mount.

Returns True on success.

Return type bool

send_radec_command(*cmd*, *ra*, *dec*)

set_tracking(*onoff*)
Enable/disable mount tracking.

Parameters **onoff** – Flag to turn tracking on/off.

Returns True on success.

Return type bool

slew(*ra*, *dec*)
Slew to ra/dec with ra in decimal hours and dec in degrees

sync(*ra*, *dec*)
Sync to ra/dec with ra in decimal hours and dec in degrees

unpark()
Unark mount.

Returns True on success.

Return type bool

class pyastrobackend.RPC.Mount.**RPCMountThread**(*port*, *user_data*, **args*, ***kwargs*)
Bases: *pyastrobackend.RPC.RPCDeviceBase.RPCDeviceThread*

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

pyastrobackend.RPC.RPCDeviceBase module

RPC Device solution

class pyastrobackend.RPC.RPCDeviceBase.**RPCDevice**(*backend=None*)
Bases: object

connect(*name*)

disconnect()

event_callback(*event*, **args*)

get_list_value(*value_method*, *value_key*)

get_scalar_value(*value_method*, *value_key*, *value_types*)

has_chooser()

is_connected()

send_command(*command*, *params={}*)

send_server_request(*req*, *paramsdict=None*)

set_scalar_value(*value_method*, *value_key*, *value*)

show_chooser(*last_choice*)

wait_for_response(*reqid*, *timeout=90*)

class pyastrobackend.RPC.RPCDeviceBase.**RPCDeviceThread**(*port*, *user_data*, **args*, ***kwargs*)
Bases: threading.Thread

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

check_rpc_command_status(*req_id*)
See if response available for request id req_id and returns it. Removes from list of requests.

close()

emit(*event*, **args*)

initialize()

populate_buffer()
Read in any new data into buffer.

queue_rpc_command(*cmd*, *argsdict*)
Accept rpc command and dictionary of arguments and creates the json request dictionary and submits to command queue for rpc client thread.

read_next_json_block()
read terminated JSON blocks

run()
Method representing the thread's activity.

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

send_polling_response()

server_disconnected()

Module contents

2.1.2 Submodules

2.1.3 pyastrobackend.ASCOMBackend module

2.1.4 pyastrobackend.AlpacaBackend module

2.1.5 pyastrobackend.BackendConfig module

pyastrobackend.BackendConfig.**get_backend**(*backend_name*)
Returns a backend object for the requested backend.

Parameters **backend_name** (*str*) – Name of desired backend.

Raises **Exception** – If unavailable backend requested raises exception.

Returns Backend instance

Return type Backend object

pyastrobackend.BackendConfig.**get_backend_choices**()
Returns all valid values for the backend name.

Returns Names of all possible backends.

Return type List[str]

pyastrobackend.BackendConfig.**get_backend_for_os**()
Return the backend matching the current system.

If the environmental variable "PYASTROBACKEND" is defined it will override the default value.

Returns Name of the default backend for this platform.

Return type str

2.1.6 pyastrobackend.BaseBackend module

class pyastrobackend.BaseBackend.**BaseCamera**
Bases: object

Definition of the camera class to be subclassed by actual classes implementing a particular camera driver.

abstract check_exposure()
Check if exposure is complete.

Returns True if exposure complete.

Return type bool

abstract check_exposure_success()
Check if exposure was successful - only valid if check_exposure() returns True.

Returns True if exposure complete.

Return type bool

abstract connect(*name*)
Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

abstract disconnect()
Disconnect from device.

abstract get_binning()
Return pixel binning.

Returns A tuple containing the X and Y binning.

Return type (int, int)

abstract get_camera_description()
Get the camera name - result depends on backend in use.

Returns Description for camera device.

Return type str

abstract get_camera_gain()
Return gain for camera (not all cameras support).

Returns Camera gain

Return type float

abstract get_camera_name()
Get the camera name - result depends on backend in use.

Returns Name of camera device.

Return type str

abstract get_cooler_power()
Get cooler power use (percentage of maximum).

Returns Cooler power level.

Return type float

abstract get_cooler_state()
Get cooler state.

Returns True if cooler is on.

Return type bool

abstract get_current_temperature()
Return current camera temperature.

Returns Temperature (C)

Return type float

abstract get_driver_info()
Get information about camera - result depends on backend in use.

Returns Driver information about camera device.

Return type str

abstract get_driver_version()
Get version information about camera - result depends on backend in use.

Returns Driver version information.

Return type str

abstract get_egain()
Return gain for camera as e- per ADU.

Returns Camera gain

Return type float

abstract get_exposure_progress()
Get percentage completion of exposure. Use *supports_progress()* to test if driver supports this call.

Returns Percentage completion of exposure.

Return type int

abstract get_frame()
Return region of interest (ROI) for image capture.

Returns A tuple containing the upper left (X, Y) for ROI and width/height.

Return type (int, int, int, int)

abstract get_image_data()
Return image data from last image taken.

Returns Image data or None if not available.

abstract get_max_binning()
Return maximum pixel binning supported.

Returns Maximum binning value.

Return type int

abstract get_pixelsize()
Return pixel size for camera sensor.

Returns A tuple containing the X and Y pixel sizes.

Return type (float, float)

abstract get_settings()
Returns most settings for camera as dict. Useful for RPC drivers to reduce round trips.

Following keys (not all will get values on all drivers):

- binning: (tuple) X, Y binning
- framesize: (tuple) Width, height of sensor
- roi: (tuple) Upper left corner and width, height of roi
- pixelsize: (tuple) X, Y pixel size
- egain: (float) Gain of camera in e-/ADU
- camera_gain: (float) Internal gain of camera
- camera_offset: (float) Internal offset of camera
- camera_usbbandwidth: (int) Internal USB traffic settings of camera
- camera_current_temperature: (float) Current temperature of camera
- camera_target_temperature: (float) Target temperature of camera
- cooler_state: (bool) Cooler on/off status
- cooler_power: (float) Power (0-100%) level of cooler

Returns Dictionary of settings

Return type dict

abstract get_size()
Return size of sensor in pixels.

Returns A tuple containing the X and Y pixel sizes.

Return type (int, int)

abstract get_state()
Get camera state.

Returns

-1	Camera State unknown

0 Camera idle

2 Camera is busy (exposing)

5 Camera error

Rtype int

abstract get_target_temperature()
Return current target camera cooler temperature.

Returns Target cooler temperature (C)

Return type float

abstract has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

abstract is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

abstract save_image_data()
Save image data from last image taken to file. NOTE: Not availale for all backends - check with supports_saveimage().

Parameters **filename** – Filename for output file

Ptype str

Returns Image data or None if not available.

abstract set_binning(*binx*, *biny*)
Set pixel binning.

Parameters

- **binx** – X binning
- **biny** – Y binning

Returns True on success.

Return type bool

abstract set_camera_gain(*gain*)
Set gain for camera (not all cameras support).

Returns True on success.

Return type bool

abstract set_cooler_state(*onoff*)
Set cooler on or off

Parameters **onff** – True to turn on camera.

Returns True on success.

Return type bool

abstract set_frame(*minx*, *miny*, *width*, *height*)
Set region of interest (ROI) for image capture.

Parameters

- **minx** – Leftmost extent of ROI.
- **miny** – Uppermost extent of ROI.
- **width** – Width of ROI.
- **height** – Height of ROI.

Returns True on success.

Return type bool

abstract set_target_temperature(*temp_c*)
Set target camera cooler temperature.

Parameters **temp_c** – Target cooler temperature (C)

Returns True on success.

Return type bool

abstract show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

abstract start_exposure(*expos*)
Start an exposure.

Parameters **expos** – Exposure length (seconds)

Ptype float

Returns True on success.

Return type bool

abstract stop_exposure()
Stop exposure.

Returns True on success.

Return type bool

abstract supports_progress()
Check if exposure progress is supported.

Returns True if progress info is available.

Return type bool

abstract supports_saveimage()
Test if drive has 'saveimage' method.

Returns True if available, False otherwise.

Return type bool

class pyastrobackend.BaseBackend.**BaseDeviceBackend**
Bases: object

Definition of the backend class to be subclassed by actual classes implementing a particular backend.

A backend represents the communication mechanism for the different device actions to interact with the actual device drivers underneath. For INDI this would be the indi-server. For ASCOM it is a placeholder as there is no actual conduit since all calls are within the process.

abstract connect()
Connect to the backennd.

abstract disconnect()
Disconnect from backend.

abstract isConnected()
Test to see if backend is connected.

Returns True if connected, False otherwise.

Return type bool

abstract newCamera()
Create a new *BaseCamera* reference.

Returns *BaseCamera* object.

Return type *BaseCamera*

abstract newFilterWheel()
Create a new *BaseFilterWheel* reference.

Returns *BaseFilterWheel* object.

Return type *BaseFilterWheel*

abstract newFocuser()
Create a new *BaseFocuser* reference.

Returns *BaseFocuser* object.

Return type *BaseFocuser*

abstract newMount()
Create a new *BaseMount* reference.

Returns *BaseMount* object.

Return type *BaseMount*

class pyastrobackend.BaseBackend.**BaseFilterWheel**
Bases: object

abstract connect(*name*)
Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

abstract disconnect()
Disconnect from device.

abstract get_names()
Get names of all filter positions.

Returns List of filter names.

Return type list

abstract get_num_positions()
Get number of filter positions.

Returns Number of filter positions

Return type int

abstract get_position()
Get position of filter wheel. First position is 0!

Returns Position of filter wheel.

Return type int

abstract get_position_name()
Get name of filter at current position.

Returns Name of current filter.

Return type str

abstract has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

abstract is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

abstract is_moving()
Check if filter wheel is moving.

Returns True if filter wheel is moving.

Return type bool

abstract set_position(*abspos*)
Set position of filter wheel. First position is 0!

Parameters **abspos** – New position of filter wheel.

Returns True on success.

Return type int

abstract set_position_name(*name*)
Set position of filter wheel by filter name..

Parameters **name** – Name of new position of filter wheel.

Returns True on success.

Return type int

abstract show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

class pyastrobackend.BaseBackend.**BaseFocuser**
Bases: object

abstract connect(*name*)
Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

abstract disconnect()
Disconnect from device.

abstract get_absolute_position()
Get absolute position of focuser.

Returns Absolute position of focuser.

Return type int

abstract get_current_temperature()
Get temperature from focuser.

Returns Temperature (C).

Return type float

get_max_absolute_position()
Get maximum possible absolute position of focuser.

Returns Absolute maximum possible position of focuser.

Return type int

abstract has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

abstract is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

abstract is_moving()
Check if focuser is moving.

Returns True if focuser is moving.

Return type bool

abstract move_absolute_position(*abspos*)
Move focuser to absolute position.

Parameters **abspos** – Target position for focuser.

Returns True on success.

Return type bool

abstract show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

abstract stop()
Stop focuser motion..

Returns True on success.

Return type bool

class pyastrobackend.BaseBackend.**BaseMount**
Bases: object

abstract abort_slew()
Abort slew.

Returns True on success.

Return type bool

abstract can_park()
Test if a mount can park.

Returns True if mount can park.

Return type bool

abstract connect(*name*)
Connect to device.

Parameters **name** – Name of driver.

Returns True on success.

Return type bool

abstract disconnect()
Disconnect from device.

abstract get_pier_side()
Returns backend specific pier side information. **NOTE: NOT** recommended for use as ASCOM and INDI may give different results for different drivers - not tested extensively at all so use with caution.

Returns 'EAST', 'WEST' or None if unknown.

abstract get_position_altaz()
Get alt/az position of mount.

Returns Tuple of (alt, az) in degrees.

Return type (float, float)

abstract get_position_radec()
Get RA/DEC position of mount.

Returns Tuple of (ra, dec) with ra in decimal hours and dec in degrees.

Return type (float, float)

abstract get_side_physical()
Get physical side of mount. **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

abstract get_side_pointing()
Get side of meridian where mount is pointing. **NOTE** may not be same as result from get_side_physical() if counterweights are pointing up, etc! **NOTE: NOT** tested extensively with all INDI drivers so it is recommended to test results for 'normal' and 'through the pole' positions on both side of the pier with a given mount driver!

Returns 'EAST', 'WEST' or None if unknown.

abstract get_tracking()
Get mount tracking state.

Returns True if tracking.

Return type bool

abstract has_chooser()
Test if a device chooser UI (ie., ASCOM) is available or not.

Returns True if chooser available, False otherwise.

Return type bool

abstract is_connected()
Test if a device is connected.

Returns True if connected, False otherwise.

Return type bool

abstract is_parked()
Test if mount is parked.

Returns True if mount is parked.

Return type bool

abstract is_slewing()
Test if mount is slewing.

Returns True if mount is slewing.

Return type bool

abstract park()
Park mount.

Returns True on success.

Return type bool

abstract set_tracking(*onoff*)
Enable/disable mount tracking.

Parameters **onoff** – Flag to turn tracking on/off.

Returns True on success.

Return type bool

abstract show_chooser(*last_choice*)
Launch chooser for driver.

Use *has_chooser()* to test if one is available for a given backend/camera.

Returns True on success.

Return type bool

abstract slew(*ra*, *dec*)
Slew mount to RA/DEC position.

Parameters

- **ra** – RA in decimal hours.
- **dec** – DEC in degrees.

Returns True on success.

Return type bool

abstract sync(*ra*, *dec*)
Sync mount to RA/DEC position.

Parameters

- **ra** – RA in decimal hours.
- **dec** – DEC in degrees.

Returns True on success.

Return type bool

abstract unpark()
Unark mount.

Returns True on success.

Return type bool

2.1.7 pyastrobackend.INDIBackend module

2.1.8 pyastrobackend.RPCBackend module

RPC solution

class pyastrobackend.RPCBackend.**DeviceBackend**(*mainThread=True*)
Bases: *pyastrobackend.BaseBackend.BaseDeviceBackend*

connect()
Connect to the backennd.

disconnect()
Disconnect from backend.

isConnected()
Test to see if backend is connected.

Returns True if connected, False otherwise.

Return type bool

name()

newCamera()
Create a new `BaseCamera` reference.

Returns `BaseCamera` object.

Return type `BaseCamera`

newFilterWheel()
Create a new `BaseFilterWheel` reference.

Returns `BaseFilterWheel` object.

Return type `BaseFilterWheel`

newFocuser()
Create a new `BaseFocuser` reference.

Returns `BaseFocuser` object.

Return type `BaseFocuser`

newMount()
Create a new `BaseMount` reference.

Returns `BaseMount` object.

Return type `BaseMount`

2.1.9 pyastrobackend.SimpleDeviceInterface module

2.1.10 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

E

F

G

H

I

M

N

P

Q

R

S

U

W