Alpyca Library

Release 3.1.2

Robert B. Denny <rdenny@dc3.com>

Jan 27, 2026

Chapter 1

Welcome to Alpyca 3.1.2

This document describes the Alpyca package, a Python API client library for ASCOM Alpaca, produced by the ASCOM Initiative, and derived from Ethan Chappel's Alpyca 1.0.0. Ethan kindly released the name **Alpyca** to the ASCOM Initiative, hence this expanded package starts life as Version 2.0.

The package provides all of the ASCOM Standard universal interfaces to astronomical devices using the Alpaca network protocol. As an application developer, your usage of the various devices is simplified and universal, independent of the particular make/model of device.

For example, the same code can be used to control any ASCOM-compatible telescope. This includes not only telescopes that are controlled with classic ASCOM/COM on a Windows machine, but also any telescopes which are *not* connected to a Windows machine, but instead speak Alpaca natively. The Windows ASCOM Remote middleware gives an Alpaca interface to any Windows-resident device, allowing you to use the device via this library from any platform on the net or local host.

Tip

Start Here: Introduction and Quick Start

Note

This is version 3.1.2, a maintenance release. For release notes see the CHANGES document on the Alpyca GitHub repository.

For background see About Alpaca and ASCOM on the ASCOM Initiative web site. As an astronomy developer wanting to use Alpaca, we suggest you look over Alpaca Developers Info and join the ASCOM Driver and Application Development Support Forum.

Attention!

Alpaca is not dependent on Windows! See About Alpaca and ASCOM.

Introduction and Quick Start

This package provides access to ASCOM compatible astronomy devices via the Alpaca network protocol. For more information see the ASCOM Initiative web site, specifically the Alpaca Developers Info section, the Alpaca API Reference, and the ASCOM Master Interfaces (Alpaca and COM).

2.1 Status of This Document

The descriptions of the ASCOM Standard interfaces implemented in Alpyca are our best efforts as of January 2026, including the results of over a year of discussion and decisions, ultimately resulting in the new interface revisions in the ASCOM Platform 7. None of these changes are breaking. They are additions needed to support asynchronous operations for Alpaca, and clarifications of existing documentation to specificallly describe already asynchronous interface members. For details see the Release notes in the Master Interfaces document liked above.

Note

Changes to the interfaces are *not* breaking. Your code using this library is safe from being broken by such changes in the future, as we never make breaking changes to interface members.

2.2 Installation

Requires Python 3.9 or later. The package installs from PyPi as:

```
pip install alpyca
```

or if you have the source code in a tar file, extract it and run:

```
python3 setup.py install
```

2.3 General Usage Pattern

To connect and control a device, the basic steps are:

1. Import the device class and Alpaca exceptions you plan to catch

2. Create an instance of the device class, giving the IP:port and device index on the Alpaca server for the device(s)
3. Connect to the Alpaca server/device
4. Call methods and read/write properties as desired, catching exceptions(!)
5. Assure that you disconnect from the device.

You will be controlling *physical devices* with your function calls here. Devices are more susceptible to problems than software. There are some very important things to be aware of:

- Some of the methods (initiator functions) are non-blocking (asynchronous) and will return right away if the operation was *started* successfully. These are clearly marked in the docs. You must validate that the operation completes *successfully* (later) by reading a *completion property* which is documented with each non-blocking function.
- You will receive an exception wherever anything fails to complete *successfully*. Not only might an initiator raise an exception, but the completion property will raise one as well if the operation failed *while in progress*. Use a `finally` clause to assure that you disconnect from the device no matter what. Please see .
- Asking the device to do something by calling an (asynchronous) method requires you to wait until the device indicates it has completed your request. Please see .

2.4 Simple Example

Run the self-contained cross-platform Alpaca Omni Simulator on your local system

Then execute this little program:

```
import time
from alpaca.telescope import *              # Multiple Classes including
 Enumerations
from alpaca.exceptions import *             # Or just the exceptions you want to
 catch

T = Telescope('localhost:32323', 0)         # Local Omni Simulator
try:
    T.Connect()                             # Asynchronous in Platform 7
    while T.Connecting:
        time.sleep(0.5)
    print(f'Connected to {T.Name}')
    print(T.Description)
    T.Tracking = True                       # Needed for slewing (see below)
    print('Starting slew...')
    T.SlewToCoordinatesAsync(T.SiderealTime + 2, 50)    # 2 hrs east of meridian
    while(T.Slewing):
        time.sleep(5)                       # What do a few seconds matter?
    print('... slew completed successfully.')
    print(f'RA={T.RightAscension} DE={T.Declination}')
    print('Turning off tracking then attempting to slew...')
    T.Tracking = False
    T.SlewToCoordinatesAsync(T.SiderealTime + 2, 55)  # 5 deg slew N
    # This will fail for tracking being off
    print("... you won't get here!")
except Exception as e:                      # Should catch specific
```

```
 InvalidOperationException
    print(f'Caught {type(e).__name__}')
    print(f'  Slew failed: {e.message}')     # Using exception named properties
finally:                                     # Assure that you disconnect
    print("Disconnecting...")
    T.Connected = False
```

Results:

```
Connected to Alpaca Telescope Sim
Software Telescope Simulator for ASCOM
Starting slew...
... slew completed successfully.
RA=10.939969572854931 DE=50
Turning off tracking then attempting to slew...
Caught InvalidOperationException
  Slew failed: SlewToCoordinatesAsync is not allowed when tracking is False
Disconnecting...
done
```

2.5 Enhancement: Emulation of Platform 7 Async Connection API

If you connect to a device whose `InterfaceVersion` indicates that it is older and does not support the new asynchronous API of Platform 7 as described in the Release Notes of the Master Interfaces Document linked above, this library will provide emulation of those calls internally. If `Connect()` or `Disconnect()` fail after the call returns, the exception will be delivered on the next read of `Connecting`.

Warning

If you don't read `Connecting` at least once after calling `Connect()` or `Disconnect()` you may miss an exception indicating that the operation failed.

Note

This emulation feature is provided to help beginners who aren't yet aware of the additional asynchronous features added to devices for Alpaca in Platform 7. Older drivers will still not support the other features as described in the Release Notes of the Master Interfaces Document linked above. It is up to you to check the `InterfaceVersion` to make sure your device supports the new Platform 7 features.

2.6 Member Capitalization

This help file provides detailed descriptions of the ASCOM Interfaces for all supported device types. Note that, rather than follow **PEP 8**, the method and property names, as well as enumerations and exceptions, all follow the capitalization that has historically been assigned to ASCOM interface members. The Class and member descriptions, notes, and exceptions raised all follow the universal ASCOM standards established long ago.

2.7 Numeric Datatypes

The Alpyca library takes care of numeric conversions so you always work in native Python numbers. When comparing numeric datatypes here in Python 3, keep the following in mind:

- Python 3's `float` is equivalent to a double-precision floating point in other languages (e.g. `double` in C#, 64-bit)
- Python 3's `int` is not restricted by the number of bits, and can expand to the limit of available memory.

Example:

```
# A Python 3 program to demonstrate that we can store
# large numbers in Python 3
x = 1000000000000000000000000000000000000000000000000
x = x + 1
print (x)
```

Output:

```
10000000000000000000000000000000000000000000001
```

2.8 Common Misconceptions and Confusions

Throughout the evolution of ASCOM, and particularly recently with Alpaca, our goal has been to provide a strong framework for reliability and integrity. We see newcomers to programming looking for help on the ASCOM Driver and Application Development Support Forum. There are a few subject areas within which misconceptions and confusion are common. Before starting an application development project with Alpyca, you may benefit from reviewing the following design principles that are *foundational*:

- The General Principles
- Asynchronous APIs
- Exceptions in ASCOM

ASCOM Alpaca Device Classes

Each of these Classes implements the properties, methods, exceptions, and enumerated constants of the corresponding ASCOM device interface.

3.1 Camera Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master ICameraV4 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

Note

See the Example: Acquiring an Image, Creating FITS Image below.

class `alpaca.camera.`**Camera** (*address: str*, *device_number: int*, *protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard iCamera V4 Interface.

Initialize the Camera object

Parameters

- **address** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **device_number** (`int`) – The index of the device (usually 0)
- **protocol** (`str, optional`) – Only if device needs https. Defaults to "http".

AbortExposure ( ) → None

Abort the current exposure, if any, and returns the camera to Idle state.

Raises

- **NotConnectedException** – If the device is not connected.
- **InvalidOperationException** – If not currently possible (e.g. during image download)
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Unlike `StopExposure()` this method simply discards any partially-acquired image data and returns the camera to idle.
- Will not raise an exception if the camera is already idle.

Master Interfaces Reference

Camera.AbortExposure()

`Action` (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **`ActionName`** – A name from `SupportedActions` that represents the action to be carried out.
- **`*Parameters`** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`ActionNotImplementedException`** – If the driver does not support the
- **`requested`** – ActionName. The supported action names are listed in `SupportedActions`.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`CommandBlind` (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters

- **`Command`** – The literal command string to be transmitted.

- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandBool (*Command: str, Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandString (*Command: str, Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

Connect ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note
- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

Disconnect ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`PulseGuide` (*Direction: GuideDirections*, *Duration: int*) → None

Pulse guide in the specified direction for the specified time (ms).

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters

- **`Direction`** – `GuideDirections`
- **`Interval`** – duration of the guide move, milliseconds

Raises

- **`NotImplementedException`** – If the camera does not support pulse guiding (`CanPulseGuide` property is False)
- **`NotConnectedException`** – If the device is not connected.
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous**: The method returns as soon pulse-guiding operation has been *successfully* started with `IsPulseGuiding` property True. However, you may find that `IsPulseGuiding` is False when you get around to checking it if the 'pulse' is short. This is still a success if you get False back and not an exception. See How can I tell if my asynchronous request failed after being started?
- Some cameras have implemented this as a Synchronous (blocking) operation.
- `GuideDirections` for North and South have varying interpretations by German Equatorial mounts. Some GEM mounts interpret North to be the same rotation direction of the declination axis regardless of their pointing state ("side of the pier"). Others truly implement North and South by reversing the dec-axis rotation depending on their pointing state. **Apps must be prepared for either behavior**.

Master Interfaces Reference

Camera.PulseGuide()

`StartExposure` (*Duration: float*, *Light: bool*) → None

Start an exposure.

Non-blocking: Returns with `ImageReady` = False if exposure has *successfully* been started. See How can I tell if my asynchronous request failed after being started?

Parameters

- **`Duration`** – Duration of exposure in seconds.
- **`Light`** – True for light frame, False for dark frame.

Raises

- **`InvalidValueException`** – If Duration is invalid, or if `BinX`, `BinY`, `NumX`, `NumY`, `StartX`, and `StartY` form an illegal combination.
- **`InvalidOperationException`** – If `CanAsymmetricBin` is False, yet `BinX` is not equal to `BinY`.
- **`NotConnectedException`** – If the device is not connected.
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. You may get this when reading IMageReady. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use `ImageReady` to determine if the exposure has been *successfully* completed. See How can I tell if my asynchronous request failed after being started?
- Refer to `ImageReady` for additional info.
- See the Example: Acquiring an Image, Creating FITS Image below.

Master Interfaces Reference

Camera.StartExposure()

`StopExposure` ( ) → None

Stop the current exposure, if any, and download the image data already acquired.

Raises

- **`NotImplementedException`** – If the camera cannot stop an in-progress exposure and save the already-acquired image data (`CanStopExposure` is False)
- **`NotConnectedException`** – If the device is not connected.
- **`InvalidOperationException`** – If not currently possible (e.g. during image download)
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Unlike `AbortExposure()` this method cuts an exposure short while preserving the image data acquired so far, making it available to the app.
- If an exposure is in progress, the readout process is initiated. Ignored if readout is already in process.
- Will not raise an exception if the camera is already idle.

Master Interfaces Reference

Camera.StopExposure()

property `BayerOffsetX`: int

The X offset of the Bayer matrix, as defined in property `SensorType`

Raises

- **`NotImplementedException`** – Monochrome cameras throw this exception, colour cameras do not.
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.
- **`InvalidValueException`** – If not valid.

Note

- The value returned will be in the range 0 to M-1 where M is the width of the Bayer matrix. The offset is relative to the 0,0 pixel in the sensor array, and does not change to reflect subframe settings.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.BayerOffsetX

property `BayerOffsetY`: int

The Y offset of the Bayer matrix, as defined in property `SensorType`

Raises

- **`NotImplementedException`** – Monochrome cameras throw this exception, colour cameras do not.
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.
- **`InvalidValueException`** – If not valid.

Note

- The value returned will be in the range 0 to M-1 where M is the width of the Bayer matrix. The offset is relative to the 0,0 pixel in the sensor array, and does not change to reflect subframe settings.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.BayerOffsetY

property `BinX`: int

(Read/Write) Set or return the binning factor for the X axis.

Raises

- **`InvalidValueException`** – If the given binning value is invalid
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Will default to 1 when the camera connection is established.
- If `CanAssymetricBin` is False, then the binning values must be the same. Setting this property will result in BinY being the same value.
- Camera does not check for compatible subframe values when this property is set; rather they are checked upon `StartExposure()`.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.BinX

property `BinY`: int

(Read/Write) Set or return the binning factor for the Y axis.

Raises

- **`InvalidValueException`** – If the given binning value is invalid
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Will default to 1 when the camera connection is established.
- If `CanAssymetricBin` is False, then the binning values must be the same. Setting this property will result in BinY being the same value.
- Camera does not check for compatible subframe values when this property is set; rather they are checked upon `StartExposure()`.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.BinY

property `CCDTemperature`: float

The current CCD temperature in degrees Celsius.

Raises

- **`InvalidValueException`** – If data unavailable.
- **`NotImplementedException`** – If not supported (no cooler)
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Camera.CCDTemperature

property `CameraState`: CameraStates

The camera's operational state (`CameraStates`)

Raises

- **`NotConnectedException`** – If the camera status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Camera.CameraState

property `CameraXSize`: int

The width of the camera sensor in unbinned pixels

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CameraXSize

property `CameraYSize`: int

The height of the camera sensor in unbinned pixels

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CameraYSize

property `CanAbortExposure`: bool

The camera can abort exposures

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Some cameras support `AbortExposure()`, which allows the exposure to be terminated before the exposure timer completes, *with the image being discarded*. Returns True if `AbortExposure()` is available, False if not. See also `StopExposure()`
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanAbortExposure

property `CanAsymmetricBin`: bool

The camera supports asymmetric binning

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If true, the camera can have different binning on the X and Y axes, as determined by BinX and BinY. If false, the binning must be equal on the X and Y axes.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanAsymmetricBin

property CanFastReadout: bool

The camera supports a fast readout mode

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanFastReadout

property CanGetCoolerPower: bool

The camera's cooler power level is available via `CoolerPower`

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanGetCoolerPower

property CanPulseGuide: bool

The camera supports pulse guiding via `PulseGuide()`

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanPulseGuide

property `CanSetCCDTemperature`: bool

The camera cooler temperature can be controlled

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If True, the camera's cooler setpoint can be adjusted. If False, the camera either uses open-loop cooling or does not have the ability to adjust temperature from software, and setting the `SetCCDTemperature` property has no effect.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanSetCCDTemperature

property `CanStopExposure`: bool

The camera can stop exposures

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Some cameras support `StopExposure()`, which allows the exposure to be terminated before the exposure timer completes, *but will still read out the image*. Returns True if `StopExposure()` is available, False if not. See also `AbortExposure()`.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.CanStopExposure

property Connected: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property Connecting: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `CoolerOn`: bool

(Read/Write) Turn the camera cooler on and off or return the current cooler on/off state.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`NotImplementedException`** – If not supported (no cooler)
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Warning

Turning the cooler off when the cooler is operating at high delta-T (typically >20C below ambient) may result in thermal shock. Repeated thermal shock may lead to damage to the sensor or cooler stack. Please consult the documentation supplied with the camera for further information.

Master Interfaces Reference

Camera.CoolerOn

property `CoolerPower`: float

The current cooler power level in percent.

Raises

- **`NotImplementedException`** – If not supported (no cooler)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Camera.CoolerPower

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `ElectronsPerADU`: float

The gain of the camera in photoelectrons per A/D unit.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Some cameras have multiple gain modes, resulting in this value changing.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.ElectronsPerADU

property `ExposureMax`: float

The maximum exposure time (sec) supported by `StartExposure()`.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.ExposureMax

property `ExposureMin`: float

The minimum exposure time (sec) supported by `StartExposure()`.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.ExposureMin

property `ExposureResolution`: float

The smallest increment in exposure time (sec) supported by `StartExposure()`.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This can be used, for example, to specify the resolution of a user interface "spin control" used to dial in the exposure time.
- The duration provided to `StartExposure()` does not have to be an exact multiple of this number; the driver will choose the closest available value. Also in some cases the resolution may not be constant over the full range of exposure times; in this case the smallest increment will be chosen by the driver. A value of 0.0 indicates that there is no minimum resolution except that imposed by the resolution of the float data type.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.ExposureResolution

property `FastReadout`: bool

(Read/Write) Gets or sets Fast Readout Mode.

Raises

- **`NotImplementedException`** – If FastReadout is not supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This function may in some cases interact with `ReadoutModes`; for example, there may be modes where the Fast/Normal switch is meaningless. In this case, it may be preferable to use the `ReadoutModes` feature to control fast /normal switching.

Master Interfaces Reference

Camera.FastReadout

property `FullWellCapacity`: float

The full well capacity of the camera (see Notes).

Raises

- **`NotConnectedException`** – If the device is not connected.
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Reports the full well capacity of the camera in electrons, at the current camera settings (binning, SetupDialog settings, etc.).
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.FullWellCapacity

property `Gain`: int

(Read/Write) Gets or sets the current gain value or index (**see Notes**)

Raises

- **`InvalidValueException`** – If the supplied valus is not valid
- **`NotImplementedException`** – If neither **gains index** mode nor **gains value** mode are supported.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

The Gain property is used to adjust the gain setting of the camera and has two modes of operation:

- **Gains-Index:** The Gain property is the selected gain's index within the `Gains` array of textual gain descriptions.
  - In this mode the Gains method returns a *0-based* array of strings, which describe available gain settings e.g. "ISO 200", "ISO 1600"
  - `GainMin` and `GainMax` will throw a `NotImplementedException`.
- **Gains-Value:** The Gain property is a direct numeric representation of the camera's gain.
  - In this mode the `GainMin` and `GainMax` properties must return integers specifying the valid range for Gain.
  - The `Gains` array property will throw a `NotImplementedException`.

A driver can support none, one or both gain modes depending on the camera's capabilities. However, only one mode can be active at any one moment because both modes share the Gain property to return the gain value. Your application can determine which mode is operational by reading the `GainMin`, `GainMax` property and this Gain property. If a property can be read then its associated mode is active, if it throws a `NotImplementedException` then the mode is not active.

Important

The `ReadoutMode` may in some cases affect the gain of the camera; if so, the driver must ensure that the two properties do not conflict if both are used.

Master Interfaces Reference

Camera.Gain

property `GainMax`: int

Maximum gain value that this camera supports (see notes and `Gain`)

Raises

- **`NotImplementedException`** – If the `Gain` property is not implemented or is operating in **gains-index** mode.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

When `Gain` is operating in **gain-value** mode:

- GainMax must return the camera's highest valid `Gain` setting
- The `Gains` property will throw **NotImplementedException**

GainMax and `GainMin` act together and that either both will return values, or both will throw **NotImplementedException**.

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.GainMax

property `GainMin`: int

Minimum gain value that this camera supports (see notes and `Gain`)

Raises

- **`NotImplementedException`** – If the `Gain` property is not implemented or is operating in **gains-index** mode.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

When `Gain` is operating in **gain-value** mode:

- GainMin must return the camera's highest valid `Gain` setting
- The `Gains` property will throw **NotImplementedException**

GainMin and `GainMax` act together and that either both will return values, or both will throw **NotImplementedException**.

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.GainMin

property `Gains`: List[str]

List of Gain *names* supported by the camera (see notes and `Gain`)

Raises

- **`NotImplementedException`** – If the `Gain` property is not imple-

mented or is operating in **gains-value** mode.

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

When Gain is operating in the **gains-index** mode:

- The Gains property returns a list of available gain setting *names*.
- The GainMax and GainMin properties will throw **NotImplementedException**.

The returned gain names could, for example, be a list of ISO settings for a DSLR camera or a list of gain names for a CMOS camera. Typically the application software will display the returned gain names in a drop list, from which the astronomer can select the required value. The application can then configure the required gain by setting the camera's Gain property to the *array index* of the selected description.

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.Gains

property HasShutter: bool

Indicate whether the camera has a mechanical shutter.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

If HasShutter is False, the StartExposure() method will ignore the Light parameter.

Master Interfaces Reference

Camera.HasShutter

property HeatSinkTemperature: float

The current heat sink (aka "ambient") temperature (deg C).

Raises

- **NotConnectedException** – If the device is not connected

- **NotImplementedException** – If CanSetCCDTemperature is False
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Camera.HeatSinkTemperature

property ImageArray: List[int]

Return a multidimensional list containing the exposure pixel values.

Raises

- **InvalidOperationException** – If no image data is available
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The returned array is in row-major format, and typically must be transposed for use with *numpy* and *astropy* for creating FITS files. See the example below.
- Automatically adapts to devices returning either JSON image data or the much faster ImageBytes format. In either case the returned nested list array contains standard Python int or float pixel values. See the . See ImageArrayInfo for metadata covering the returned image data.

Master Interfaces Reference

Camera.ImageArray

property ImageArrayInfo: ImageMetadata

Get image metadata sucn as dimensions, data type, rank.
See Class ImageMetadata for the properties available.

Note

If no image has been retrieved via ImageArray, this returns None.

Master Interfaces Reference

Camera.ImageArrayInfo

property ImageArrayRaw: array

Return an array containing the exposure pixel values.

Raises

- **InvalidOperationException** – If no image data is available
- **NotConnectedException** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The returned array is in row-major format, and typically must be transposed for use with *numpy* and *astropy* for creating FITS files. See the example below.
- Automatically adapts to devices returning either JSON image data or the much faster ImageBytes format. In either case the returned array contains standard Python int or float pixel values. See the . See `ImageArrayInfo` for metadata covering the returned image data.

Master Interfaces Reference

Camera.ImageArray

property ImageReady: bool

Indicates that an image is ready to be downloaded.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions.The device cannot *successfully* complete the previous `Expose()` request (see Attention below).

Note

- If ImageReady returns a valid False or True value, then the *non-blocking* process of acquiring an image is *proceeding normally* or has been *successful*.
- ImageReady will be False immediately upon return from `StartExposure()`. It will remain False until the exposure has been *successfully* completed and an image is ready for download.

Attention!

- If the camera encounters a problem which prevents or prevented it from *successfully* completing the exposure, the driver will raise an exception when you attempt to read ImageReady.

Master Interfaces Reference

Camera.ImageReady

property InterfaceVersion: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `IsPulseGuiding`: bool

Indicates that the camera is currently in a `PulseGuide()` operation.

Raises

- `NotConnectedException` – If the device is not connected
- `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. See Attention below. The device did not *successfully* complete the request.

Note

- If IsPulseGuiding returns a valid True or False value, then the process of pulse-guiding is *proceeding normally* or has completed *successfully*, respectively.
- IsPulseGuiding will be True immediately upon return from `PulseGuide()`. It will remain True until the requested pulse-guide interval has elapsed, and the pulse-guiding operation has been *successfully* completed. If `PulseGuide()` returns with IsPulseGuiding = False, then you can assume that the operation *succeeded* with a very short pulse-guide interval.

Attention!

- If the camera encounters a problem which prevents it from *successfully* completing the the pulse-guiding operation, the driver will raise an exception when you attempt to read IsPulseGuiding.

Master Interfaces Reference

Camera.IsPulseGuiding

property `LastExposureDuration`: float

Report the actual exposure duration in seconds (i.e. shutter open time).

Raises

- **`NotImplementedException`** – If the camera doesn't support this feature
- **`InvalidOperationException`** – If no image has yet been *successfully* acquired.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. See Attention below. The device did not *successfully* complete the request.

Note

- This may differ from the exposure time requested due to shutter latency, camera timing precision, etc.

Master Interfaces Reference

Camera.LastExposureDuration

property `LastExposureStartTime`: str

Start time of the last exposure in FITS standard format, UTC.

Raises

- **`NotImplementedException`** – If the camera doesn't support this feature
- **`InvalidOperationException`** – If no image has yet been *successfully* acquired.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. See Attention below. The device did not *successfully* complete the request.

Note

Reports the actual exposure UTC start date/time in the FITS-standard / ISO-8601 CCYY-MM-DDThh:mm:ss[.sss...] format.

Master Interfaces Reference

Camera.LastExposureStartTime

property `MaxADU`: int

The maximum ADU value of the camera.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.MaxADU

property MaxBinX: int

The maximum supported X binning value of the camera.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.MaxBinX

property MaxBinY: int

The maximum supported Y binning value of the camera.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.MaxBinY

property Name: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **DriverException** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property NumX: int

(Read/Write) Set or return the current subframe width.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If binning is active, value is in binned pixels.
- Defaults to `CameraXSize` with `StartX` = 0 (full frame) on initial camera startup.

Attention!

- No error check is performed for incompatibilty with `BinX`, and `StartX`, If these values are incompatible, you will receive an **InvalidValueException** from a subsequent call to `StartExposure()`.

Master Interfaces Reference

Camera.NumX

property NumY: int

(Read/Write) Set or return the current subframe height.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If binning is active, value is in binned pixels.
- Defaults to `CameraYSize` with `StartY` = 0 (full frame) on initial camera startup.

Attention!

- No error check is performed for incompatibilty with `BinY`, and `StartY`, If these values are incompatible, you will receive an **InvalidValueException** from a subsequent call to `StartExposure()`.

Master Interfaces Reference

Camera.NumY

property `Offset`: int

(Read/Write) Gets or sets the current offset value or index (**see Notes**)

Raises

- **`InvalidValueException`** – If the supplied value is not valid
- **`NotImplementedException`** – If neither **offsets index** mode nor **offsets value** mode are supported.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

The Offset property is used to adjust the offset setting of the camera and has two modes of operation:

- **Offsets-Index:** The Offset property is the selected offset's index within the `Offsets` array of textual offset descriptions.
  - In this mode the `Offsets` method returns a *0-based* array of strings, which describe available offset settings.
  - `OffsetMin` and `OffsetMax` will throw a `NotImplementedException`.
- **Offsets-Value:** The Offset property is a direct numeric representation of the camera's offset.
  - In this mode the `OffsetMin` and `OffsetMax` properties must return integers specifying the valid range for Offset.
  - The `Offsets` array property will throw a `NotImplementedException`.

A driver can support none, one or both offset modes depending on the camera's capabilities. However, only one mode can be active at any one moment because both modes share the Offset property to return the offset value. Your application can determine which mode is operational by reading the `OffsetMin`, `OffsetMax` property and this Offset property. If a property can be read then its associated mode is active, if it throws a `NotImplementedException` then the mode is not active.

Important

The `ReadoutMode` may in some cases affect the offset of the camera; if so, the driver must ensure that the two properties do not conflict if both are used.

Master Interfaces Reference

Camera.Offset

property `OffsetMax`: int

Maximum offset value that this camera supports (see notes and `Offset`)

Raises

- **`NotImplementedException`** – If the `Offset` property is not implemented or is operating in **offsets-index** mode.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

When `Offset` is operating in **offsets-value** mode:

- `OffsetMax` must return the camera's highest valid `Offset` setting
- The `Offsets` property will throw **NotImplementedException**

OffsetMax and `OffsetMin` act together and that either both will return values, or both will throw **NotImplementedException**.

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.OffsetMax

property `OffsetMin`: int

Minimum offset value that this camera supports (see notes and `Offset`)

Raises

- **`NotImplementedException`** – If the `Offset` property is not implemented or is operating in **offsets-index** mode.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

When `Offset` is operating in **offsets-value** mode:

- OffsetMin must return the camera's highest valid `Offset` setting
- The `Offsets` property will throw `NotImplementedException`.

OffsetMin and `OffsetMax` act together and that either both will return values, or both will throw `NotImplementedException`.

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.OffsetMin

property `Offsets`: List[str]

List of Offset *names* supported by the camera (see notes and `Offset`)

Raises

- **`NotImplementedException`** – If the `Offset` property is not imple-

mented or is operating in **offsets-value** mode.

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

When `Offset` is operating in the **offsets-index** mode:

- The `Offsets` property returns a list of available offset setting *names*.
- The `OffsetMax` and `OffsetMin` properties will throw `NotImplementedException`.

The returned offset names could, for example, be a list of ISO settings for a DSLR camera or a list of offset names for a CMOS camera. Typically the application software will display the returned offset names in a drop list, from which the astronomer can select the required value. The application can then configure the required offset by setting the camera's Offset property to the *array index* of the selected description.

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.Offsets

property `PercentCompleted`: int

The percentage completeness of this operation

Raises

- **InvalidOperationException** – When it is inappropriate to ask for a completion percentage.
- **NotImplementedException** – If this optional property is not implemented.
- **NotConnectedException** – If the device is not connected.
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. See Attention below. The device did not *successfully* complete the request.

Note

- If valid, returns an integer between 0 and 100, where 0 indicates 0% progress (function just started) and 100 indicates 100% progress (i.e. completion).
- At the discretion of the device, PercentCompleted may optionally be valid when `CameraState` is in any or all of the following states:
  - `cameraExposing`
  - `cameraWaiting`
  - `cameraReading`
  - `cameraDownloading`

In all other states an `InvalidOperationException` will be raised.

Attention!

- If the camera encounters a problem which prevents or prevented it from *successfully* completing the operation, the driver will raise an exception when you attempt to read PercentComplete.

Master Interfaces Reference

Camera.PercentCompleted

property `PixelSizeX`: float

The width (microns) of the camera sensor elements.

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.PixelSizeX

property `PixelSizeY`: float

The height (microns) of the camera sensor elements.

Raises
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.PixelSizeY

property ReadoutMode: int

(Read/Write) Gets or sets the current camera readout mode (**see Notes**)

Raises

- **InvalidValueException** – If the supplied value is not valid (index out of range)
- **NotImplementedException** – If `CanFastReadout` is True.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- ReadoutMode is an index into the array `ReadoutModes`, and selects the desired readout mode for the camera. Defaults to 0 if not set.
- It is strongly recommended, but not required, that cameras make the 0-index mode suitable for standard imaging operations, since it is the default.

Important

The `ReadoutMode` may in some cases affect the `Gain` and/or `Offset` of the camera; if so, the camera must ensure that the two properties do not conflict if both are used.

Master Interfaces Reference

Camera.ReadoutMode

property ReadoutModes: List[str]

List of ReadoutMode *names* supported by the camera (see notes and `ReadoutMode`)

Raises

- **NotImplementedException** – If the `ReadoutMode` property is not implemented.

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Readout modes may be available from the camera, and if so then `CanFastReadout` will be False. The two camera mode selection schemes are mutually exclusive.
- This property provides an array of strings, each of which describes an available readout mode of the camera. At least one string will be present in the list. Your application may use this list to present to the user a drop-list of modes. The choice of available modes made available is entirely at the discretion of the camera. Please note that if the camera has many different modes of operation, then the most commonly adjusted settings will probably be in ReadoutModes; additional settings may be provided using `SetupDialog()`.
- To select a mode, set ReadoutMode to the index of the desired mode. The index is zero-based.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.ReadoutModes

property SensorName: str

The name of the sensor used within the camera.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Returns the name (data sheet part number) of the sensor, e.g. ICX285AL. The format is to be exactly as shown on manufacturer data sheet, subject to the following rules:
  - All letters will be upper-case.
  - Spaces will not be included.
  - Any extra suffixes that define region codes, package types, temperature range, coatings, grading, colour/monochrome, etc. will not be included.
  - For colour sensors, if a suffix differentiates different Bayer matrix encodings, it will be included.
  - The property will return an empty string if the sensor name is not known
- **Examples:**
  - ICX285AL-F shall be reported as ICX285
  - KAF-8300-AXC-CD-AA shall be reported as KAF-8300
- The most common usage of this property is to select approximate colour balance parameters to be applied to the Bayer matrix of one-shot colour sensors. Application authors should assume that an appropriate IR cut-off filter is in place for colour sensors.
- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.SensorName

property `SensorType`: SensorType

The type of sensor within the camera.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is recommended that this property be retrieved only after a connection is established with the camera hardware, to ensure that the driver is aware of the capabilities of the specific camera model.

Master Interfaces Reference

Camera.SensorType

property `SetCCDTemperature`: float

(Read/Write) Get or set the camera's cooler setpoint (degrees Celsius).

Raises

- **`InvalidValueException`** – If set to a value outside the camera's valid temperature setpoint range.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Camera.SetCCDTemperature

property `StartX`: int

(Read/Write) Set or return the current X-axis subframe start position.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If binning is active, value is in binned pixels.
- Defaults to 0 with `NumX` = `CameraXSize` (full frame) on initial camera startup.

Attention!

- No error check is performed for incompatibilty with `BinX`, and `NumX`, If these values are incompatible, you will receive an **InvalidValueException** from a subsequent call to `StartExposure()`.

Master Interfaces Reference

Camera.StartX

property `StartY`: int

(Read/Write) Set or return the current Y-axis subframe start position.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- If binning is active, value is in binned pixels.
- Defaults to 0 with `NumY` = `CameraYSize` (full frame) on initial camera startup.

Attention!

- No error check is performed for incompatibilty with `BinY`, and `NumY`, If these values are incompatible, you will receive an **InvalidValueException** from a subsequent call to `StartExposure()`.

Master Interfaces Reference

Camera.StartY

property SubExposureDuration: float

(Read/Write) Set or return the camera's sub-exposure interval (sec)

Raises

- **NotImplementedException** – The camera does not support on-board stacking with user-supplied sub-exposure interval.
- **NotConnectedException** – If the device is not connected.
- **InvalidValueException** – The supplied duration is not valid.
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Camera.SubExposureDuration

property SupportedActions: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

3.1.1 ImageMetadata Class

class `alpaca.camera.`**`ImageMetadata`** (*metadata_version: int*, *image_element_type: ImageArrayElementTypes*, *transmission_element_type: ImageArrayElementTypes*, *rank: int*, *num_x: int*, *num_y: int*, *num_z: int*)

Bases: `object`

Metadata describing the returned ImageArray data

Note

- Constructed internally by the library during image retrieval.
- See https://ascom-standards.org/Developer/AlpacaImageBytes.pdf

property `Dimension1`

The first (X) dimension of the image array

property `Dimension2`

The second (Y) dimension of the image array

property `Dimension3`

The third (Z) dimension of the image array (None or 3)

property `ImageElementType`: ImageArrayElementTypes

The data type of the pixels in originally acquired image

> **Note**
>
> Within Python, the returned nested list(s) image pixels themselves will be either int or float.

property `MetadataVersion`
The version of metadata, currently 1

property `Rank`
The matrix rank of the image data (either 2 or 3)

property `TransmissionElementType`: ImageArrayElementTypes
The ddta type of the pixels in the transmitted image bytes stream

> **Note**
>
> Within Python, the returned image pixels themselves will be either int or float.
>
> To save transmission time camera may choose to use a smaller data type than the original image if the pixel values would all be representative in that data type without a loss of precision.

3.1.2 Camera-Related Constants

enum `alpaca.camera.`**`CameraStates`** (*value*)
Bases: `DocIntEnum`
Current condition of the Camera

Member Type `int`
Valid values are as follows:

`cameraIdle` = <CameraStates.cameraIdle: 0>
Inactive

`cameraWaiting` = <CameraStates.cameraWaiting: 1>
Waiting for ??

`cameraExposing` = <CameraStates.cameraExposing: 2>
Acquiring photons

`cameraReading` = <CameraStates.cameraReading: 3>
Reading from the sensor

`cameraDownload` = <CameraStates.cameraDownload: 4>
Downloading the image data

`cameraError` = <CameraStates.cameraError: 5>
An error condition exists

enum `alpaca.camera.`**`SensorType`** (*value*)
Bases: `DocIntEnum`
Type of sensor in the Camera. Names should be self-explanatory.

Member Type `int`
Valid values are as follows:

Monochrome = <SensorType.Monochrome: 0>

Color = <SensorType.Color: 1>

RGGB = <SensorType.RGGB: 2>

CMYG = <SensorType.CMYG: 3>

CMYG2 = <SensorType.CMYG2: 4>

LRGB = <SensorType.LRGB: 5>

enum `alpaca.camera.`**ImageArrayElementTypes** (*value*)
Bases: `DocIntEnum`
The native data type of ImageArray pixels

Member Type `int`
Valid values are as follows:

Unknown = <ImageArrayElementTypes.Unknown: 0>

Int16 = <ImageArrayElementTypes.Int16: 1>

Int32 = <ImageArrayElementTypes.Int32: 2>

Double = <ImageArrayElementTypes.Double: 3>

Single = <ImageArrayElementTypes.Single: 4>
Unused in Alpaca 2022

UInt64 = <ImageArrayElementTypes.UInt64: 5>
Unused in Alpaca 2022

Byte = <ImageArrayElementTypes.Byte: 6>
Unused in Alpaca 2022

Int64 = <ImageArrayElementTypes.Int64: 7>
Unused in Alpaca 2022

UInt16 = <ImageArrayElementTypes.UInt16: 8>
Unused in Alpaca 2022

3.1.3 Example: Acquiring an Image, Creating FITS Image

Using numpy and astropy.io.fits, connect to an Alpaca Camera, acquire a short image, download and make a local FITS file:

```
import os
import time
import array
from alpaca.camera import *     # Sorry Python purists, this has multiple required
 Classes
```

```
import numpy as np
import astropy.io.fits as fits

#
# Set up the camera
#
c = Camera('localhost:32323', 0)    # Connect to the ALpaca Omni Simulator
c.Connected = True
c.BinX = 1
c.BinY = 1
# Assure full frame after binning change
c.StartX = 0
c.StartY = 0
c.NumX = c.CameraXSize // c.BinX    # Watch it, this needs to be an int (typ)
c.NumY = c.CameraYSize // c.BinY
#
# Acquire a light image, wait while printing % complete
#
c.StartExposure(2.0, True)
while not c.ImageReady:
    time.sleep(0.5)
    print(f'{c.PercentCompleted}% complete')
print('finished')
#
# OK image acquired, grab the image array and the metadata
#
img = c.ImageArray
imginfo = c.ImageArrayInfo
if imginfo.ImageElementType == ImageArrayElementTypes.Int32:
    if c.MaxADU <= 65535:
        imgDataType = np.uint16 # Required for BZERO & BSCALE to be written
    else:
        imgDataType = np.int32
elif imginfo.ImageElementType == ImageArrayElementTypes.Double:
    imgDataType = np.float64
#
# Make a numpy array of he correct shape for astropy.io.fits
#
if imginfo.Rank == 2:
    nda = np.array(img, dtype=imgDataType).transpose()
else:
    nda = np.array(img, dtype=imgDataType).transpose(2,1,0)
#
# Create the FITS header and common FITS fields
#
hdr = fits.Header()
hdr['COMMENT'] = 'FITS (Flexible Image Transport System) format defined in
 Astronomy and'
hdr['COMMENT'] = 'Astrophysics Supplement Series v44/p363, v44/p371, v73/p359, v73
/p365.'
hdr['COMMENT'] = 'Contact the NASA Science Office of Standards and Technology for
 the'
hdr['COMMENT'] = 'FITS Definition document #100 and other FITS information.'
if imgDataType ==  np.uint16:
    hdr['BZERO'] = 32768.0
    hdr['BSCALE'] = 1.0
hdr['EXPOSURE'] = c.LastExposureDuration
hdr['EXPTIME'] = c.LastExposureDuration
hdr['DATE-OBS'] = c.LastExposureStartTime
hdr['TIMESYS'] = 'UTC'
```

```
hdr['XBINNING'] = c.BinX
hdr['YBINNING'] = c.BinY
hdr['INSTRUME'] = c.SensorName
try:
    hdr['GAIN'] = c.Gain
except:
    pass
try:
    hdr['OFFSET'] = c.Offset
    if type(c.Offset == int):
        hdr['PEDESTAL'] = c.Offset
except:
    pass
hdr['HISTORY'] = 'Created using Python alpyca-client library'
#
# Create the final FITS from the numpy array and FITS info
#
hdu = fits.PrimaryHDU(nda, header=hdr)

img_file = f"{os.getenv('USERPROFILE')}/Desktop/test.fts"
hdu.writeto(img_file, overwrite=True)
c.Connected = False

print("Booyah! Your FITS image is ready.")
```

Resulting FITS header:

```
Header listing for HDU #1:
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                   16 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 1280
NAXIS2  =                 1024
EXPOSURE=            2.0052547
EXPTIME =            2.0052547
DATE-OBS= '2022-04-15T18:20:50'
TIMESYS = 'UTC     '
XBINNING=                    1
YBINNING=                    1
INSTRUME= 'MyCamera'
BSCALE  =                    1
BZERO   =                32768
COMMENT FITS (Flexible Image Transport System) format defined in Astronomy and
COMMENT Astrophysics Supplement Series v44/p363, v44/p371, v73/p359, v73/p365.
COMMENT Contact the NASA Science Office of Standards and Technology for the
COMMENT FITS Definition document #100 and other FITS information.
HISTORY Created using Python alpyca-client library
END
```

3.2 CoverCalibrator Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master ICoverCalibratorv4 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class `alpaca.covercalibrator.`**CoverCalibrator** (*address: str*, *device_number: int*, *protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard ICoverCalibratorV2 Interface

Initialize CoverCalibrator object.

Parameters

- **address** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **device_number** (`int`) – The index of the device (usually 0)
- **protocol** (`str, optional`) – Only if device needs https. Defaults to "http".

Action (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **ActionName** – A name from `SupportedActions` that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in `SupportedActions`.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`CalibratorOff` ( ) → None

Turns the calibrator off if the device has calibration capability

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – When `CalibratorState` is `NotPresent`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): If the calibrator requires time to safely stabilise after use, `CalibratorChanging` will become `True` and `CalibratorState` will return `NotReady`. When the calibrator is safely off, `CalibratorChanging` will become `False` `CalibratorState` will return `Off`. See How can I tell if my asynchronous request failed after being started?.

Master Interfaces Reference

CoverCalibrator.CalibratorOff()

`CalibratorOn` (*BrightnessVal: int*) → None

Turns the calibrator on if the device has calibration capability

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters **`Brightness`** – The calibrator illumination brightness to be set

Raises

- **`NotImplementedException`** – When `CalibratorState` is `NotPresent`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): If the calibrator requires time to safely stabilise after use, `CalibratorChanging` will become `True` and `CalibratorState` will return `NotReady`. When the calibrator is ready for use, `CalibratorChanging` will become `False` `CalibratorState` will return `Ready`. See How can I tell if my asynchronous request failed after being started?.
- If an error condition arises while turning on the calibrator, `CalibratorState` will be set to `Error` rather than `Unknown`.

Attention!

For devices with both cover and calibrator capabilities, this method may change the `CoverState`, if required. This operation is also **asynchronous** (non-blocking) so you may need to wait for `CoverState` to reach `Open`. See How can I tell if my asynchronous request failed after being started?

Master Interfaces Reference

CoverCalibrator.CalibratorOn()

`CloseCover` ( ) → None

Initiates cover closing if a cover is present

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – When `CoverState` is `NotPresent`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): `CoverState` indicates the status of the operation once CloseCover() returns. It will be `Moving` immediately after the return of CloseCover(), and will remain as long as the operation is progressing successfully. See How can I tell if my asynchronous request failed after being started?
- `Closed` indicates *successful* completion.
- If an error condition arises while moving between states, `CoverState` will be set to `Error` rather than `Unknown`

Master Interfaces Reference

CoverCalibrator.CloseCover()

CommandBlind (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandBool (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandString` (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

`Connect` ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`HaltCover` ( ) → None

Immediately stops an in-progress `OpenCover()` or `CloseCover()`

Raises

- **`NotImplementedException`** – When `CoverState` is `NotPresent`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This will stop any cover movement as soon as possible and set a `CoverState` of `Open`, `Closed` or `Unknown` as appropriate.
- If cover movement cannot be interrupted, a `NotImplementedException` will be thrown.

Master Interfaces Reference

CoverCalibrator.HaltCover()

OpenCover ( ) → None

Initiates cover opening if a cover is present

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **NotImplementedException** – When CoverState is NotPresent
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): CoverState indicates the status of the operation once OpenCover() returns. It will be Moving immediately after the return of OpenCover(), and will remain as long as the operation is progressing successfully. See How can I tell if my asynchronous request failed after being started?
- Open indicates *successful* completion.
- If an error condition arises while moving between states, CoverState will be set to Error rather than Unknown

Master Interfaces Reference

CoverCalibrator.OpenCover()

property Brightness: int

The current calibrator brightness (0 - MaxBrightness)

Raises

- **NotImplementedException** – When CalibratorState is NotPresent
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The brightness value will be 0 when CalibratorState is Off

Master Interfaces Reference

CoverCalibrator.Brightness

property CalibratorChanging: bool

True whenever the Calibrator is **not** ready to be used (illumination not yet stabilized), or not completely shut down.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `CalibratorOn()` or `CalibratorOff()` has completed, at which time it will transition from `True` to `False`.
- Present only in devices with `InterfaceVersion` = 2.

Master Interfaces Reference

CoverCalibrator.CalibratorChanging

property `CalibratorState`: CalibratorStatus

The state of the calibration device

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If no calibrator is present, the state will be `NotPresent`. You will not receive a `NotImplementedException`.
- The brightness value will be 0 when CalibratorState is `Off`
- The `Unknown` state will only be returned if the device is unaware of the calibrator's state e.g. if the hardware does not report the device's state and the calibrator has just been powered on. You do not need to take special action if this state is returned, you must carry on as usual, calling `CalibratorOn()` and `CalibratorOff()` methods as required.
- If the calibrator hardware cannot report its state, the device might mimic this by recording the last configured state and returning that. Driver authors or device manufacturers may also wish to offer users the capability of powering up in a known state and driving the hardware to this state when `Connected` is set True.

Master Interfaces Reference

CoverCalibrator.CalibratorState

property `Connected`: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property CoverMoving: bool

`True` whenever an (async) `OpenCover()` or `CloseCover()` operation is in progress.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `OpenCover()` or `CloseCover()` has completed, at which time it will transition from `True` to `False`.
- Present only in devices with `InterfaceVersion` = `2`.

Master Interfaces Reference

CoverCalibrator.CoverMoving

property CoverState: CoverStatus

The state of the device cover

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If no cover is present, the state will be `NotPresent`. You will not receive a `NotImplementedException`.
- The `Unknown` state will only be returned if the device is unaware of the cover's state e.g. if the hardware does not report the device's state and the cover has just been powered on. You do not need to take special action if this state is returned, you must carry on as usual, calling `OpenCover()` and `CloseCover()` methods as required.
- If the cover hardware cannot report its state, the device might mimic this by recording the last configured state and returning that. Driver authors or device manufacturers may also wish to offer users the capability of powering up in a known state and driving the hardware to this state when connecting.

Master Interfaces Reference

CoverCalibrator.CoverState

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `MaxBrightness`: int

The Brightness value that makes the calibrator deliver its maximum illumination.

Raises

- **`NotImplementedException`** – When `CalibratorState` is `NotPresent`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a mandatory property if a calibrator device is present (`CalibratorState` is other than `NotPresent`)
- The value will always be a positive integer, indicating the available precision.
- Examples: A value of 1 indicates that the calibrator can only be "off" or "on". A value of 10 indicates that the calibrator has 10 discrete illumination levels in addition to "off".

Master Interfaces Reference

CoverCalibrator.MaxBrightness

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `SupportedActions`: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

3.2.1 CoverCalibrator-Related Constants

enum `alpaca.covercalibrator.`**`CoverStatus`** (*value*)

Bases: `DocIntEnum`

Indicates the current status of the cover

Member Type `int`

Valid values are as follows:

`NotPresent` = <CoverStatus.NotPresent: 0>

`Closed` = <CoverStatus.Closed: 1>

`Moving` = <CoverStatus.Moving: 2>

Open = <CoverStatus.Open: 3>

Unknown = <CoverStatus.Unknown: 4>

Error = <CoverStatus.Error: 5>

enum alpaca.covercalibrator.**CalibratorStatus** (*value*)

Bases: DocIntEnum

Indicates the current status of the calibrator

Member Type int

Valid values are as follows:

NotPresent = <CalibratorStatus.NotPresent: 0>

Off = <CalibratorStatus.Off: 1>

NotReady = <CalibratorStatus.NotReady: 2>

Ready = <CalibratorStatus.Ready: 3>

Unknown = <CalibratorStatus.Unknown: 4>

Error = <CalibratorStatus.Error: 5>

3.3 Dome Class

The Dome interface is designed to provide an enclosure-independent way of managing access to the sky for the telescope within. Enclosures vary widely in their design, with roll-off roofs and classic rotating domes being only two of the possibilities.

Thus, this interface focuses on providing the telescope with access to the sky at a given sky location specified by alt/az coordinates. For additional help, see The Dome Interface seems complex and confusing. Help me. For some history, see

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master IDomeV3 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class alpaca.dome.**Dome** (*address: str, device_number: int, protocol: str = 'http'*)

Bases: Device

ASCOM Standard IDomeV2 Interface

Initialize Dome object.

Parameters

- **address** (str) – IP address and port of the device (x.x.x.x:pppp)
- **device_number** (int) – The index of the device (usually 0)
- **protocol** (str, optional) – Only if device needs https. Defaults to "http".

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

AbortSlew ( ) → None

Immediately stops any part of the dome from moving, opening, or closing. See Notes.

Raises
- **NotConnectedException** – If the device is not connected
- **DriverException** – If a communications failure occurs, or if the AbortSlew() request itself fails in some way. This exception may be encountered on any call to the device.

Note

- When this call succeeds, Slewing will become False, and slaving will have stopped as indicate by Slaved becoming False.
- By "any part of the dome" is meant the dome itself, the roof, a shutter, clamshell leaves, a port, etc. Calling AbortSlew() will stop alt/az movement of the opening as well as stopping opening or closing.

Master Interfaces Reference

Dome.AbortSlew()

Action (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters
- **ActionName** – A name from SupportedActions that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises
- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in SupportedActions.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`CloseShutter` ( ) → None

Start to close the shutter or otherwise shield the telescope from the sky
Non-blocking: Returns immediately with `ShutterStatus` = `shutterClosing` after *successfully* starting the operation. See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – If the dome does not have a controllable shutter/roof. In this case `CanSetShutter` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): `ShutterStatus` is the correct property to use for monitoring an in-progress shutter movement. A transition to `shutterClosed` indicates a *successfully completed* closure. If it returns with `ShutterStatus` `shutterClosed`, it means the shutter was already closed, another success. If See How can I tell if my asynchronous request failed after being started?
- If another app calls CloseShutter() while the shutter is already closing, the request will be accepted and you will see `ShutterStatus` = `shutterClosing` as you would expect.

Attention!

This operation is not cross-coupled in any way with the currently requested `Azimuth` and `Altitude`. Opening and closing are used to shield and expose the opening to the sky, wherever it is specified to be.

Master Interfaces Reference

Dome.CloseShutter()

`CommandBlind` (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandBool (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandString (*Command: str, Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

Connect ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note
- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

Disconnect ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

FindHome ( )

Start a search for the dome's home position and synchronize Azimuth.

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **NotImplementedException** – If the dome does not support homing.
- **NotConnectedException** – If the device is not connected
- **SlavedException** – If `Slaved` is True
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `AtHome` property to monitor the operation. When the the home position is has been *successfully* reached, `Azimuth` is synchronized to the appropriate value, `AtHome` becomes True and `Slewing` becomes False. See How can I tell if my asynchronous request failed after being started?
- An app should check `AtHome` before calling FindHome().

Master Interfaces Reference

Dome.FindHome()

OpenShutter ( ) → None

Start to open shutter or otherwise expose telescope to the sky.

Non-blocking: Returns immediately with `ShutterStatus` = `shutterOpening` if the opening has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **NotImplementedException** – If the dome does not have a controllable shutter/roof. In this case `CanSetShutter` will be False.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): `ShutterStatus` is the correct property to use for monitoring an in-progress shutter movement. A transition to `shutterOpen` indicates a *successfully completed* opening. If OpenShutter returns with `ShutterStatus` = `shutterOpen` then the shutter was already open, which is also a success. See How can I tell if my asynchronous request failed after being started?
- If another app calls OpenShutter() while the shutter is already opening, the request will be accepted and you will see `ShutterStatus` = `shutterOpening` as you would expect.

Attention!

This operation is not cross-coupled in any way with the currently requested `Azimuth` and `Altitude`. Opening and closing are used to shield and expose the opening to the sky, wherever it is specified to be.

Master Interfaces Reference

Dome.OpenShutter()

Park ( ) → None

Start slewing the dome to its park position.

Non-blocking: Returns immediately with `Slewing` = True if the park operation has *successfully* been started, or `Slewing` = False which means the dome is already parked (and of course `AtPark` will already be True). See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **NotImplementedException** – If the dome does not support parking. In this case `CanPark` will be False.
- **NotConnectedException** – If the device is not connected
- **ParkedException** – If `AtPark` is True
- **SlavedException** – If `Slaved` is True

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `AtPark` property to monitor the operation. When the the park position has been *successfully* reached, `Azimuth` is synchronized to the park position, `AtPark` becomes True, and `Slewing` becomes False. See How can I tell if my asynchronous request failed after being started?
- An app should check `AtPark` before calling Park().

Master Interfaces Reference

Dome.Park()

SetPark ( ) → None

Set current position of dome to be the park position

Raises

- **NotImplementedException** – If the dome does not support the setting of the park position. In this case `CanSetPark` will be False.
- **NotConnectedException** – If the device is not connected
- **SlavedException** – If `Slaved` is True
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.SetPark()

SlewToAltitude (*Altitude: float*) → None

Start slewing the opening to the given altitude (degrees).

Non-blocking: Returns immediately with `Slewing` = True if the slewing operation has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters **Altitude** – The requested altitude of the opening

Raises

- **NotImplementedException** – If the dome opening does not support vertical (altitude) control. In this case `CanSetAltitude` will be False.
- **NotConnectedException** – If the device is not connected
- **SlavedException** – If `Slaved` is True
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not

successfully complete the request.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When the the requested Altitude has been *successfully* reached, `Slewing` becomes False. If SlewToAltitude() returns with `Slewing` = False then the opening was already at the requested altitude, which is also a success See How can I tell if my asynchronous request failed after being started?
- The specified altitude (*referenced to the dome center/equator*) is of the position of the opening.

Attention!

If the opening is closed, this method must still complete, with the dome controller accepting the requested position as its `Altitude` property. Later, when opening, via `OpenShutter()`, the last received/current `Altitude` is used to position the opening to the sky.

Master Interfaces Reference

Dome.SlewToAltitude()

`SlewToAzimuth` (*Azimuth: float*) → None

Start slewing the opening to the given azimuth (degrees).
Non-blocking: Returns immediately with `Slewing` = True if the slewing operation has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters **`Azimuth`** – The requested azimuth of the opening. See Notes.

Raises

- **`NotImplementedException`** – If the dome does not support rotational (azimuth) control. In this case `CanSetAzimuth` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`SlavedException`** – If `Slaved` is True
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When the the requested Azimuth has been *successfully* reached, `Slewing` becomes False. If SlewToAzimuth() returns with `Slewing` = False then the opening was already at the requested azimuth, which is also a success See How can I tell if my asynchronous request failed after being started?
- Azimuth has the usual sense of True North zero and increasing clockwise i.e. 90 East, 180 South, 270 West.
- The specified azimuth (*referenced to the dome center/equator*) is of the position of the opening.

Attention!

If the shutter is closed, this method will still complete, with the dome controller accepting the requested position as its `Azimuth` property. Later, when the shutter is opened via `OpenShutter()`, the last received/current `Azimuth` is used to re-position the opening to the sky. This may extend the time needed to complete the `OpenShutter()` operation.

Master Interfaces Reference

Dome.SlewToAzimuth()

`SyncToAzimuth` (*Azimuth: float*) → None

Synchronize the current azimuth of the dome (degrees) to the given azimuth.

Raises

- **`NotImplementedException`** – If the shutter does not support azimuth synchronization. In this case `CanSyncAzimuth` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`SlavedException`** – If `Slaved` is True
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.SyncToAzimuth()

property `Altitude`: float

Dome altitude (degrees) of the opening to the sky.

Raises

- **`NotImplementedException`** – If the dome does not support vertical (altitude) control / placement of its observing opening (including a roll-off roof). In this case `CanSetAltitude` will be False.
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The specified altitude (*referenced to the dome center/equator*) is of the opening to the sky through which the optics receive light.
- It is up to the dome control and driver to determine how best to locate the dome aperture in order to expose the specified alt/az area to the sky, including positioning clamshell leaves, split shutters, etc. Your app need not know how this is happening, just that the alt/az area of the sky will be visible.
- Do not use Altitude as a way to determine if a (non-blocking) `SlewToAltitude()` has completed. The Altitude may transit through the requested position before finally settling, and may be slightly off when it stops. Use the `Slewing` property.

Attention!

An ASCOM Dome device does not include transformations for mount/optics to azimuth and altitude. It is prohibited for a stand-alone Dome control device to require cross-linking to query a telescope directly. Your app will need to provide the dome-centered alt/az given the geometry of the mount and optics in use. See also the `Slaved` property for details on slaving (telescope motion tracking). Only an *integrated* mount/dome system will offer both a Telescope and a Dome interface, and be capable of slaving.

Master Interfaces Reference

Dome.Altitude

property `AtHome`: bool

The dome is in the home position.

Note

This is normally used following a findhome() operation. The value is reset with any azimuth slew operation that moves the dome away from the home position. athome() may also become true durng normal slew operations, if the dome passes through the home position and the dome controller hardware is capable of detecting that; or at the end of a slew operation if the dome comes to rest at the home position.

Returns True if dome is in the home position.

Master Interfaces Reference

Dome.AtHome

property `AtPark`: bool

The telescope has *successfully* reached its park position.

Raises

- **`NotImplementedException`** – If the dome does not support parking. In this case `CanPark` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

Set only following a park() operation and reset with any slew operation.

Returns True if the dome is in the programmed park position.

Master Interfaces Reference

Dome.AtPark

property `Azimuth`: float

Dome azimuth (degrees) of the opening to the sky

This this does not include the geometric transformations needed for mount and optics configurations. See The Dome Interface seems complex and confusing. Help me..

Raises

- **`NotImplementedException`** – If the dome does not support directional (azimuth) control / placement of its observing opening (including roll-off roof). In this case `CanSetAzimuth` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Azimuth has the usual sense of True North zero and increasing clockwise i.e. 90 East, 180 South, 270 West.
- The specified azimuth (*referenced to the dome center/equator*) is of the opening to the sky through which the optics receive light.
- You can detect a roll-off roof by `CanSetAzimuth` being False.
- It is up to the dome control and driver to determine how best to locate the dome aperture in order to expose the specified alt/az area to the sky, including positioning clamshell leaves, split shutters, etc. Your app need not know how this is happening, just that the alt/az area of the sky will be visible.
- Do not use Azimuth as a way to determine if a (non-blocking) `SlewToAzimuth()` has completed. The Azimuth may transit through the requested position before finally settling, and may be slightly off when it stops. Use the `Slewing` property.

Attention!

An ASCOM Dome device does not include transformations for mount/optics to azimuth and altitude. It is prohibited for a stand-alone Dome control device to require cross-linking to query a telescope directly. Your app will need to provide the dome-centered alt/az given the geometry of the mount and optics in use. See also the `Slaved` property for details on slaving (telescope motion tracking). Only an *integrated* mount/dome system will offer both a Telescope and a Dome interface, and be capable of slaving.

Master Interfaces Reference

Dome.Azimuth

property `CanFindHome`: bool

The dome can find its home position via `FindHome()`

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.CanFindHome

property `CanPark`: bool

The dome can be programmatically parked via `Park()`

Raises

- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.CanPark

property CanSetAltitude: bool

The opening's altitude can be set via `SetAltitude()`

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.CanSetAltitude

property CanSetAzimuth: bool

The opening's azimuth can be set via `SetAzimuth()`

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.CanSetAzimuth

property CanSetPark: bool

The dome park position can be set via `SetPark()`

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.CanSetPark

property CanSetShutter: bool

The shutter can be opened and closed via `OpenShutter()` and `CloseShutter()`

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Master Interfaces Reference

Dome.CanSetShutter

property CanSlave: bool

The opening can be slaved to the telescope/optics via `Slaved` (see Notes)

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If this is True, then the exposed Dome interface is part of an integrated mount/dome control system that offers automatic slaving.

Attention!

An ASCOM Dome device does not include transformations for mount/optics to azimuth and altitude. It is prohibited for a stand-alone Dome control device to require cross-linking to query a telescope directly. Your app will need to provide the dome-centered alt/az given the geometry of the mount and optics in use. See also the `Slaved` property for details on slaving (telescope motion tracking).

Master Interfaces Reference

Dome.CanSlave

property CanSyncAzimuth: bool

The opening's azimuth position can be synched via `SyncToAzimuth()`.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Dome.CanSyncAzimuth

property Connected: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- `NotConnectedException` – If the device status is unavailable

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the DriverInfo property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for Description, find this specific device's specification, and see Description there.

property DeviceState: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for DeviceState, find this specific device's specification, and see DeviceState there.

property DriverInfo: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `ShutterStatus`: ShutterState

Status of the dome shutter or roll-off roof.

Raises

- **`NotImplementedException`** – If the dome does not have a controllable shutter/roof. In this case `CanSetShutter` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This property is the correct way to monitor an in-progress shutter movement. It will be `shutterOpening' immediately after returning from an :meth:`OpenShutter()` call, and `shutterClosing' immediately after returning from a :meth:`CloseShutter()` call.

Master Interfaces Reference

Dome.ShutterStatus

property `Slaved`: bool

(Read/Write) Indicate or set whether the dome is slaved to the telescope.

Raises

- **`NotImplementedException`** – If the dome controller is not par of an integrated dome/telescope control system which offers controllable dome slaving. In this case `CanSlave` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

An ASCOM Dome device does not include transformations for mount/optics to azimuth and altitude. It is prohibited for a stand-alone Dome control device to require cross-linking to query a telescope directly. Your app will need to provide the dome-centered alt/az given the geometry of the mount and optics in use. See also the `Slaved` property for details on slaving (telescope motion tracking).

Master Interfaces Reference

Dome.Slaved

property `Slewing`: bool

Any part of the dome is moving, opening, or closing. See Notes.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – If the device cannot *successfully complete* a previous movement request. This exception may be encountered on any call to the device.

Note

- This is the correct property to use to determine *successful* completion of a (non-blocking) `SlewToAzimuth()` and/or `SlewToAltitude()` request. Slewing will be True immediately upon returning from either of these calls, and will remain True until *successful* completion, at which time Slewing will become False.
- By “any part of the dome” is meant the roof, a shutter, clamshell leaves, a port, etc. This will be true during alt/az movement of the opening as well as opening or closing.

Master Interfaces Reference

property SupportedActions: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

3.3.1 Dome-Related Constants

enum `alpaca.dome.`**ShutterState** (*value*)

Bases: `DocIntEnum`

Indicates the current state of the shutter or roof

Member Type `int`

Valid values are as follows:

shutterOpen = <ShutterState.shutterOpen: 0>

The shutter or roof is open

shutterClosed = <ShutterState.shutterClosed: 1>

The shutter or roof is closed

shutterOpening = <ShutterState.shutterOpening: 2>

The shutter or roof is opening

shutterClosing = <ShutterState.shutterClosing: 3>

The shutter or roof is closing

shutterError = <ShutterState.shutterError: 4>
The shutter or roof has encountered a problem

3.4 FilterWheel Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master IFilterWheelv3 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class alpaca.filterwheel.**FilterWheel** (*address: str*, *device_number: int*, *protocol: str = 'http'*)

Bases: Device

ASCOM Standard IFilterWheelV2 interface.

Initialize FilterWheel object.

Parameters
- **address** (str) – IP address and port of the device (x.x.x.x:pppp)
- **device_number** (int) – The index of the device (usually 0)
- **protocol** (str, optional) – Only if device needs https. Defaults to "http".

Action (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters
- **ActionName** – A name from SupportedActions that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises
- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in SupportedActions.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`CommandBlind` (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters
- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandBool` (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandString (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

Connect ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

property `Connected`: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `FocusOffsets`: List[int]

List of filter focus offsets for each filter in the wheel

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The offset values in this list are in the same order as the filters in the wheel
- The number of available filters can be determined from the length of the list.
- If focuser offsets are not available, then the list will contain zeroes.

Master Interfaces Reference

FilterWheel.FocusOffsets

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `Names`: List[str]

List of filter names for each filter in the wheel

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The names in this lisat are in the same order as the filters in the wheel
- The number of available filters can be determined from the length of the list.
- If focuser offsets are not available, then the lost will contain generic names of 'Filter 1', 'Filter 2', etc.

Master Interfaces Reference

FilterWheel.Names

property `Position`: int

(Read/Write) Start a change to, or return the filter wheel position (zero-based)
Non-blocking: Returns immediately upon writing to change the filter with Position = -1 if the operation has been *successfully* started. See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`InvalidValueException`** – If an invalid filter number is written to Position.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

Asynchronous (non-blocking): Writing to Position returns as soon as the filter change operation has been *successfully* started. Position will return -1 while the change is in progress. After the requested position has been *successfully* reached and motion stops, Position will return the requested new filter number. See How can I tell if my asynchronous request failed after being started?

Master Interfaces Reference

FilterWheel.Position

property `SupportedActions`: List[str]

The list of custom action names supported by this driver
Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

3.5 Focuser Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master IFocuserv4 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class `alpaca.focuser.`**`Focuser`**(*address: str, device_number: int, protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard IFocuserV3 Interface

Attention!

It is possible to command the focuser to a position exceeding its limits (see notes for `MaxStep`) without receiving an exception. This is by design.

Initialize Focuser object.

Parameters

- **`address`** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **`device_number`** (`int`) – The index of the device (usually 0)

- **protocol** (`str, optional`) – Only if device needs https. Defaults to "http".

Action (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **ActionName** – A name from `SupportedActions` that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in `SupportedActions`.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

CommandBlind (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not

successfully complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandBool` (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandString` (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters
- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **`NotImplementedException`** – If no actions at all are supported

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

Connect ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

Disconnect ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`Halt` ( ) → None

Immediately stop any focuser motion due to a previous `Move()` call.

Raises
- **`NotImplementedException`** – The focuser cannot be programmatically halted.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- You should try to call this method aftr initialization to see if halting is supported by your device. You can use this info to possibly disable a Halt button in your user interface.

Master Interfaces Reference

Focuser.Halt()

`Move` (*Position: int*) → None

Starts moving the focuser to a new position

Non-blocking: Returns immediately after *successfully* starting the focus change with `IsMoving` = True. See Notes, and How can I tell if my asynchronous request failed after being started?

See Notes for details on absolute versus relative focusers

Parameters **`Position`** – Step distance or absolute position, depending on the value of the `Absolute` property.

Raises
- **`InvalidValueException`** – If Position would result in a movement beyond `MaxStep`.
- **`InvalidOperationException`** – **IFocuserV2 and earlier only** Raised if `TempComp` is true and a Move() is attempted. This restriction was removed in IFocuserV3, but you must be prepared

to catch this for older focusers (2018).

- **NotImplementedException** – The focuser cannot be programmatically halted.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): The method returns as soon as the focus change operation has been *successfully* started, with the `IsMoving` property True. After the requested position is *successfully* reached and motion stops, the `IsMoving` property becomes False. See How can I tell if my asynchronous request failed after being started?
- If the `Absolute` property is True, then this is an absolute positioning focuser. The `Move()` method tells the focuser to move to an exact step position, and the Position parameter of the `Move()` method is an integer between 0 and `MaxStep`.
- If the `Absolute` property is False, then this is a relative positioning focuser. The `Move()` method tells the focuser to move in a relative direction. The Position parameter of the `Move()` method is actually a *step distance* and is an integer between minus `MaxIncrement` and plus `MaxIncrement`.

Master Interfaces Reference

Focuser.Move()

property Absolute: bool

The focuser does absolute positioning. See `Move()`.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

True means the focuser is capable of absolute position; that is, being commanded to a specific step location. False means this is a relative positioning focuser.

Master Interfaces Reference

Focuser.Absolute

property Connected: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises
- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `IsMoving`: bool

The focuser is currently moving to a new position

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the correct property to use to determine *successful* completion of a (non-blocking) `Move()` request. IsMoving will be True immediately upon returning from a `Move()` call, and will remain True until *successful* completion, at which time `IsMoving` will become False.

Master Interfaces Reference

Focuser.IsMoving

property `MaxIncrement`: int

Maximum number of steps allowed in one `Move()` operation.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- For most focusers this is the same as the `MaxStep` property. This is normally used to limit the increment display in the host software.

Master Interfaces Reference

Focuser.MaxIncrement

property `MaxStep`: int

Maximum step position permitted.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The focuser can step between 0 and MaxStep. If an attempt is made to move the focuser beyond these limits, it will automatically stop at the limit.

Master Interfaces Reference

Focuser.MaxStep

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `Position`: int

Current focuser position, in steps.

Raises

- **`NotImplementedException`** – The device is a relative focuser (`Absolute` is False)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Do not use this as a way to determine if a (non-blocking) `Move()` has completed. The Position may transit through the requested position before finally settling. Use the `IsMoving` property.

Master Interfaces Reference

Focuser.Position

property `StepSize`: float

Step size (microns) for the focuser.

Raises

- **`NotImplementedException`** – If the device does not intrinsically know what the step size is.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Focuser.StepSize

property `SupportedActions`: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a “discovery” mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

property `TempComp`: bool

(read/write) Set or indicate the state of the focuser's temp compensation.

Raises

- **`NotImplementedException`** – On writing to TempComp, if `TempCompAvailable` is False, indicating that this focuser does not have temperature compensation. In that case reading TempComp will always return False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Setting TempComp to True puts the focuser into temperature tracking mode; setting it to False will turn off temperature tracking.
- If `TempCompAvailable` is False this property will always return False.

Master Interfaces Reference

Focuser.TempComp

property `TempCompAvailable`: bool

If focuser has temperature compensation available.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Focuser.TempCompAvailable

property `Temperature`: float

Current **ambient** temperature (deg. C).

Raises

- **`NotImplementedException`** – The temperature is not available for this device.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of

the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Historically (prior to 2019) no units were specified for this property. You should assume this this is in degrees Celsius but old devices may supply temperature in other units. By now (2022) however devices should be providing degrees celsius.

Master Interfaces Reference

Focuser.Temperature

3.6 ObservingConditions Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master IObservingConditionsv2 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class alpaca.observingconditions.**ObservingConditions** (*address: str, device_number: int, protocol: str = 'http'*)

Bases: Device

ASCOM Standard IObservingConditions Interface

Provides measurements of meterological conditions as apply to astronomy. Determination of safe/unsafe is made by a separate SafetyMonitor device.

Initialize the ObservingConditions object.

Parameters

- **address** (str) – IP address and port of the device (x.x.x.x:pppp)
- **device_number** (int) – The index of the device (usually 0)
- **protocol** (str, optional) – Only if device needs https. Defaults to "http".

Action (*ActionName: str, *Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **ActionName** – A name from SupportedActions that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **NotImplementedException** – If no actions at all are supported

- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in SupportedActions.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with SupportedActions, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for Action(), find this specific device's specification, and see Action() there.

CommandBlind (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in NotImplementedException

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for CommandBlind(), find this specific device's specification, and see CommandBlind() there.

CommandBool (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandString (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

`Connect` ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`Refresh` ( ) → None

Forces the device to immediately query its attached hardware to refresh sensor values

Raises

- **`NotImplementedException`** – This method is not supported.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.Refresh()

`SensorDescription` (*SensorName: str*) → str

Description of the sensor providing the requested property

Parameters **`SensorName`** – A string containing the name of the ObservingConditions meterological property for which the sensor description is desired. For example "WindSpeed" (for `WindSpeed`) would retrieve a description of the sensor used to measure the wind speed.

Raises

- **`NotImplementedException`** – This method is not supported.
- **`NotConnectedException`** – If the device is not connected.
- **`InvalidValueException`** – The supplied SensorName is not valid.
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.SensorDescription

`TimeSinceLastUpdate` (*SensorName: str*) → str

Elapsed time (sec) since last update of the sensor providing the requested property

Parameters **`SensorName`** – A string containing the name of the ObservingConditions meterological property for which the time since last update is desired. For example "WindSpeed" (for `WindSpeed`) would retrieve the time since the wind speed was last updated by its sensor.

Raises

- **`NotImplementedException`** – This method is not supported.

- **NotConnectedException** – If the device is not connected.
- **InvalidValueException** – The supplied SensorName is not valid.
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.TimeSinceLastUpdate

property AveragePeriod: float

(read/write) Gets And sets the time period (hours) over which observations will be averaged

Raises

- **InvalidValueException** – If the value set is out of bounds for this device. All devices must accept 0.0 to specify that an instantaneous value is to be made available.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- AveragePeriod returns the time period (hours) over which sensor readings will be averaged. If the device is delivering instantaneous sensor readings this property will return a value of 0.0.
- Though discouraged in the specification, possible you will receive an exception if you read a sensor property when insufficient time has passed to get a true average reading.

Master Interfaces Reference

ObservingConditions.AveragePeriod

property CloudCover: float

Amount of sky obscured by cloud (0.0-1.0)

Raises

- **NotImplementedException** – This property is not available.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.CloudCover

property Connected: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property Connecting: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DewPoint`: float

Atmospheric dew point temperature (deg C) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.DewPoint

property DriverInfo: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property DriverVersion: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `Humidity`: float

Atmospheric relative humidity (0-100%) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.Humidity

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request.

This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `Pressure`: float

Atmospheric pressure (hPa) at the observatory altitude

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

Not "corrected to sea level" as often encountered in weather reports. The ConvertPressure() method may be used to get "sea level" pressure

Master Interfaces Reference

ObservingConditions.Pressure

property `RainRate`: float

Rain rate (mm/hr) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.RainRate

property `SkyBrightness`: float

Sky brightness (Lux) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.SkyBrightness

property `SkyQuality`: float

Sky quality (mag per sq-arcsec) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.SkyQuality

property `SkyTemperature`: float

Sky temperature (deg C) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.SkyTemperature

property `StarFWHM`: float

Seeing (FWHM in arc-sec) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.StarFWHM

property `SupportedActions`: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

property `Temperature`: float

Atmospheric temperature (deg C) at the observatory

Raises

- **NotImplementedException** – This property is not available.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.Temperature

property `WindDirection`: float

Direction (deg) from which the wind is blowing at the observatory

Raises

- **NotImplementedException** – This property is not available.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Meterological standards** Wind direction is that from which the wind is blowing, measured in degrees clockwise from *true* North=0.0, East=90.0, South=180.0, West=270.0 If the wind velocity is 0 then direction is reported as 0.

Master Interfaces Reference

ObservingConditions.WindDirection

property `WindGust`: float

Peak 3 second wind gust (m/s) at the observatory over the last 2 minutes

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.WindGust

property `WindSpeed`: float

Wind speed (m/s) at the observatory

Raises

- **`NotImplementedException`** – This property is not available.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

ObservingConditions.WindSpeed

3.7 Rotator Class

The Rotator V3 interface provides for a common offset between its mechanical angle, plus the angle at which an attached imager may be mounted, and the equatorial position angle (PA) on the sky. By calling `Sync()` with a known current PA (from plate solving etc.), you can cause the rotator (and imager) to work in PA for you as well as other apps that might be using the rotator.

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master IRotatorv4 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class `alpaca.rotator.`**`Rotator`** (*address: str, device_number: int, protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard IRotatorV3 interface.

Initialize the Rotator object.

Parameters

- **`address`** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **`device_number`** (`int`) – The index of the device (usually 0)
- **`protocol`** (`str, optional`) – Only if device needs https. Defaults to "http".

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

`Action` (*ActionName: str, *Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **`ActionName`** – A name from `SupportedActions` that represents the action to be carried out.
- **`*Parameters`** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`ActionNotImplementedException`** – If the driver does not support the
- **`requested`** – ActionName. The supported action names are listed in `SupportedActions`.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`CommandBlind` (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters

- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandBool` (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters

- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandString` (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

`Connect` ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`Halt` ( ) → None

Immediately stop any rotator motion due to a previous movement call.

Raises

- **`NotImplementedException`** – The rotator cannot be programmatically halted.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- You should try to call this method aftr initialization to see if halting is supported by your device. You can use this info to possibly disable a Halt button in your user interface.

Master Interfaces Reference

Rotator.Halt()

Move (*Position: float*) → None

Starts rotation relative to the current position (degrees)

Non-blocking: Returns immediately with `IsMoving` = True if the operation has *successfully* been started, or if it returns with `IsMoving` = False, it will already be at the requested position, also a success. See Notes, and How can I tell if my asynchronous request failed after being started?

Also See Notes for details on absolute versus relative movement.

Parameters **Position** – The angular amount (degrees) to move relative to the current position.

Raises

- **InvalidValueException** – The given position change results in a position outside 0 <= position < 360.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous**: The method returns as soon as the rotation operation has been successfully started, with the `IsMoving` property True. After the requested angle is successfully reached and motion stops, the `IsMoving` property becomes False. See How can I tell if my asynchronous request failed after being started?
- Calling Move causes the TargetPosition property to change to the sum of the current angular position and the value of the Position parameter (modulo 360 degrees), then starts rotation to TargetPosition. Position includes the effect of any previous Sync() operation.

Master Interfaces Reference

Rotator.Move()

MoveAbsolute (*Position: float*) → None

Starts rotation to the new position (degrees)

Non-blocking: Returns immediately with `IsMoving` = True if the operation has *successfully* been started, or if it returns with `IsMoving` = False, it will already be at

the requested position, also a success. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters **Position** – The requested angle, degrees.

Raises

- **InvalidValueException** – The given position is 0 <= position < 360.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous**: The method returns as soon as the rotation operation has been successfully started, with the `IsMoving` property True. After the requested angle is successfully reached and motion stops, the `IsMoving` property becomes False. See How can I tell if my asynchronous request failed after being started?
- Calling Move causes the TargetPosition property to change to the value of the Position parameter (modulo 360 degrees), then starts rotation to TargetPosition. Position includes the effect of any previous Sync() operation.

Master Interfaces Reference

Rotator.MoveAbsolute()

MoveMechanical (*Position: float*) → None

Starts rotation to the given mechanical position (degrees)

Non-blocking: Returns immediately with `IsMoving` = True if the operation has *successfully* been started, or if it returns with `IsMoving` = False, it will already be at the requested position, also a success. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters **Position** – The requested angle, degrees.

Raises

- **InvalidValueException** – The given position is 0 <= position < 360. [or does it just apply modulo 360? Then what is an "invalid" value?]
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous**: The method returns as soon as the rotation operation has been successfully started, with the `IsMoving` property True. After the requested angle is successfully reached and motion stops, the `IsMoving` property becomes False. See How can I tell if my asynchronous request failed after being started?
- Calling MoveMechanical causes the TargetPosition property to change to the value of the Position parameter then starts rotation to TargetPosition. This moves without regard to the SyncOffset, that is, to the mechanical rotator angle.
- This method is to address requirements that need a physical rotation angle such as taking sky flats.

Master Interfaces Reference

Rotator.MoveMechanical()

Sync (*Position: float*) → None

Syncs the rotator to the specified position angle (degrees) without moving it.

Parameters **Position** – The requested angle, degrees.

Raises

- **InvalidValueException** – The given position is 0 <= position < 360. [or does it just apply modulo 360? Then what is an "invalid" value?]
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Once this method has been called and the sync offset determined, both the `MoveAbsolute()` method and the `Position` property will function in synced coordinates rather than mechanical coordinates. The sync offset will persist across driver starts and device reboots.

Master Interfaces Reference

Rotator.Sync()

property CanReverse: bool

The rotator supports the Reverse method (see Notes)

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- For IRotatorV3 drivers and later(`InterfaceVersion` >= 3) CanReverse is always True.
- For more info on reversal see the `Reverse` property.

Master Interfaces Reference

Rotator.CanReverse

property `Connected`: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises
- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `IsMoving`: bool

The rotator is currently moving to a new position

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the correct property to use to determine *successful* completion of a (non-blocking) `Move()` request. IsMoving will be True immediately upon returning from a `Move()` call, and will remain True until *successful* completion, at which time IsMoving will become False.

Master Interfaces Reference

Rotator.IsMoving

property `MechanicalPosition`: bool

The raw mechanical position (deg) of the rotator

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

Value is in degrees counterclockwise from the rotator's mechanical index.

Master Interfaces Reference

Rotator.MechanicalPosition

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `Position`: bool

This returns the position (deg) of the rotator allowing for sync offset

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Position is in degrees counterclockwise
- The `Sync()` method may used to make Position indicate equatorial position angle. This can account for not only an offset in the rotator's mechanical position, but also the angle at which an attached imager is mounted.
- If `Sync()` has never been called, Position will be equal to `MechanicalPosition`. Once called, however, the offset will remain across driver starts and device reboots.

Master Interfaces Reference

Rotator.Position

property `Reverse`: bool

(Read/Write) Set or indicate rotation direction reversal.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

Rotation is normally in degrees counterclockwise as viewed from behind the rotator, looking toward the sky. This corresponds to the direction of equatorial position angle. Set this property True to cause rotation opposite to equatorial PositionAngle, i.e. clockwise.

Master Interfaces Reference

Rotator.Reverse

property `StepSize`: float

The minimum rotation step size (deg)

Raises

- **`NotImplementedException`** – If this property is not available from the device
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Rotator.StepSize

property SupportedActions: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

property TargetPosition: float

The destination angle for `Move()` and MoveAbsolute().

Note

This will contain the new Position, including any `Sync()` offset, immediately upon return from a call to `Move()` or `MoveAbsolute()`.

Master Interfaces Reference

Rotator.TargetPosition

3.8 SafetyMonitor Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master ISafetyMonitorv3 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class `alpaca.safetymonitor.`**SafetyMonitor** (*address: str*, *device_number: int*, *protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard ISafetyMonitor V1 Interface.

Provides a single property that indicates whether it is safe to expose the observatory instruments to the outside environment, or not. The measurements of meterological conditions that your application (or a separate weather monitoring system) uses to make this decision will most often come from sensors that are accessed through the `ObservingConditions` interface.

Initialize the SafetyMonitor object.

Parameters
- **`address`** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **`device_number`** (`int`) – The index of the device (usually 0)
- **`protocol`** (`str, optional`) – Only if device needs https. Defaults to "http".

`Action` (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters
- **`ActionName`** – A name from `SupportedActions` that represents the action to be carried out.
- **`*Parameters`** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises
- **`NotImplementedException`** – If no actions at all are supported
- **`ActionNotImplementedException`** – If the driver does not support the
- **`requested`** – ActionName. The supported action names are listed in `SupportedActions`.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`CommandBlind` (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters
- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandBool` (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandString (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

Connect ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

property `Connected`: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `IsSafe`: bool

The monitored state is safe for use.

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

SafetyMonitor.IsSafe

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `SupportedActions`: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

3.9 Switch Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master ISwitchv3 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class `alpaca.switch.`**`Switch`** (*address: str, device_number: int, protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard ISwitch V2 Interface

Initialize the Switch device.

Parameters

- **`address`** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **`device_number`** (`int`) – The index of the device (usually 0)
- **`protocol`** (`str, optional`) – Only if device needs https. Defaults to "http".

`Action` (*ActionName: str, *Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **`ActionName`** – A name from `SupportedActions` that represents the action to be carried out.

- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises
- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in SupportedActions.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note
- This method, combined with SupportedActions, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for Action(), find this specific device's specification, and see Action() there.

CanAsync (*Id: int*) → bool

The specified switch can operate asynchronously. See SetAsync() and SetAsyncValue().

Parameters **Id** – the specified switch number (see Notes)

Raises
- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note
- Switches are numbered from 0 to MaxSwitch - 1.
- Examples of switches that cannot be written to include a limit switch or a sensor.

Master Interfaces Reference

Switch.CanAsync()

CanWrite (*Id: int*) → bool

The specified switch can be written to.

Parameters **Id** – the specified switch number (see Notes)

Raises

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to MaxSwitch - 1.
- Examples of witches that cannot be written to include a limit switch or a sensor.

Master Interfaces Reference

Switch.CanWrite()

CancelAsync (*Id: int*) → None

Cancels an in-progress asynchronous state change operation. See SetAsync() and SetAsyncValue() for details of asynchronous switch operations.

Parameters **Id** – the specified switch number (see Notes)

Raises

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- On return, the next call to StateChangeComplete() for this switch will raise an OperationCancelledException; thereafter calls to StateChangeComplete() for the switch will return False.
- Switches are numbered from 0 to MaxSwitch - 1.

Master Interfaces Reference

Switch.CancelAsync()

CommandBlind (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandBool (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandString` (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

`Connect` ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`GetSwitch` (*Id: int*) → bool

The state of the specified switch.

Parameters **`Id`** – the specified switch number (see Notes)

Raises

- **`InvalidValueException`** – The Id is out of range (see `MaxSwitch`)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.
- On is True, Off is False.

Master Interfaces Reference

Switch.GetSwitch()

GetSwitchDescription (*Id: int*) → str

The textual description of the specified switch.

Parameters **Id** – the specified switch number (see Notes)

Raises

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to MaxSwitch - 1.

Master Interfaces Reference

Switch.GetSwitchDescription()

GetSwitchName (*Id: int*) → str

The textual name of the specified switch.

Parameters **Id** – the specified switch number (see Notes)

Raises

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to MaxSwitch - 1.

Master Interfaces Reference

Switch.GetSwitchName()

GetSwitchValue (*Id: int*) → float

The value of the specified switch as a float.

Parameters **Id** – the specified switch number (see Notes)

Raises

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by

one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.

Master Interfaces Reference

Switch.GetSwitchValue()

`MaxSwitchValue` (*Id: int*) → float

The maximum value of the specified switch as a double.

Parameters **`Id`** – the specified switch number (see Notes)

Raises

- **`InvalidValueException`** – The Id is out of range (see `MaxSwitch`)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.

Master Interfaces Reference

Switch.MaxSwitchValue()

`MinSwitchValue` (*Id: int*) → float

The minimum value of the specified switch as a double.

Parameters **`Id`** – the specified switch number (see Notes)

Raises

- **`InvalidValueException`** – The Id is out of range (see `MaxSwitch`)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.

Master Interfaces Reference

Switch.MinSwitchValue()

SetAsync (*Id: int*, *State: bool*) → None

Asynchronouly Set a switch to the specified boolean on/off state.

Parameters

- **Id** – the specified switch number (see Notes)
- **State** – The required control state

Raises

- **NotImplementedException** – If `CanAsync() = False` for switch `Id`
- **InvalidValueException** – The `Id` is out of range (see `MaxSwitch`)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): The method returns as soon as the state change operation has been successfully started, with `StateChangeComplete()` for switch `Id = False`. After the state change has completed `StateChangeComplete()` becomes True.
- Switches are numbered from 0 to `MaxSwitch - 1`.
- On is True, Off is False.

Master Interfaces Reference

Switch.SetAsync()

SetAsyncValue (*Id: int*, *Value: float*) → None

Asynchronouly Set a switch to the specified value

Parameters

- **Id** – the specified switch number (see Notes)
- **Value** – The value to be set, between `MinSwitchValue`()` and `MaxSwitchValue()` for switch `Id`

Raises

- **NotImplementedException** – If `CanAsync() = False` for switch `Id`
- **InvalidValueException** – The Id is out of range (see `MaxSwitch`), or if the given value is not between `MinSwitchValue()` and `MaxSwitchValue()` for the given switch `Id`.
- **NotConnectedException** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): The method returns as soon as the state change operation has been successfully started, with `StateChangeComplete()` for switch `Id = False`. After the state change has completed `StateChangeComplete()` becomes True.
- Switches are numbered from 0 to `MaxSwitch - 1`.
- On is True, Off is False.

Master Interfaces Reference

Switch.SetAsyncValue()

SetSwitch (*Id: int*, *State: bool*) → None

Set a switch to the specified state

Parameters

- **Id** – the specified switch number (see Notes)
- **State** – The required control state

Raises

- **InvalidValueException** – The Id is out of range (see `MaxSwitch`)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.
- On is True, Off is False.

Master Interfaces Reference

Switch.SetSwitch()

SetSwitchName (*Id: int*, *Name: str*) → None

Set a switch name to the specified value.

Parameters

- **Id** – the specified switch number (see Notes)
- **Name** – The desired (new) name for the switch

Raises

- **InvalidValueException** – The Id is out of range (see `MaxSwitch`)

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.
- On is True, Off is False.

Master Interfaces Reference

Switch.SetSwitchName()

SetSwitchValue (*Id: int*, *Value: float*) → None

Set a switch value to the specified value.

Parameters

- **Id** – the specified switch number (see Notes)
- **Value** – Value to be set, between `MinSwitchValue` and `MinSwitchValue`.

Raises

- **InvalidValueException** – The Id is out of range (see `MaxSwitch`), or the Value is out of range, not between `MinSwitchValue` and `MinSwitchValue`.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to `MaxSwitch - 1`.
- On is True, Off is False.

Master Interfaces Reference

Switch.SetSwitchValue()

StateChangeComplete (*Id: int*) → bool

True if the last `SetAsync()` or `SetAsyncValue()` has completed and the switch is in the requested state.

Parameters **Id** – the specified switch number (see Notes)

Raises

- **NotImplementedException** – If `CanAsync()` is `False` for switch `Id`
- **OperationCancelledException** – If an in-progress state change is cancelled by a call to `CancelAsync()` call for switch

Id

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Switches are numbered from 0 to MaxSwitch - 1.

Master Interfaces Reference

Switch.StateChangeComplete()

SwitchStep (*Id: int*) → float

The step size of the specified switch (see Notes).

Parameters **Id** – the specified switch number (see Notes)

Raises

- **InvalidValueException** – The Id is out of range (see MaxSwitch)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Step size is the difference between successive values of the device.
- Switches are numbered from 0 to MaxSwitch - 1.

Master Interfaces Reference

Switch.SwitchStep()

property Connected: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `MaxSwitch`: int

Count of switches managed by this device.

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Number of switches managed by this device. Switches are numbered from 0 to MaxSwitch - 1.

Master Interfaces Reference

Switch.MaxSwitch

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **`DriverException`** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `SupportedActions`: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

3.10 Telescope Class

Master Interfaces Reference

These green boxes in each interface member each have a link to the corresponding member definition in the Master ITelescopev4 Interface document. The information in this Alpyca document is provided *for your convenience*. If there is any question, the info in ASCOM Master Interfaces is the official specification.

class `alpaca.telescope.`**`Telescope`** (*address: str, device_number: int, protocol: str = 'http'*)

Bases: `Device`

ASCOM Standard ITelescope V3 Interface

Initialize the Telescope object.

Parameters

- **`address`** (`str`) – IP address and port of the device (x.x.x.x:pppp)
- **`device_number`** (`int`) – The index of the device (usually 0)
- **`protocol`** (`str, optional`) – Only if device needs https. Defaults to "http".

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

AbortSlew ( ) → None

Immediatley stops an asynchronous slew in progress.

Raises

- **InvalidOperationException** – If the mount is parked (AtPark = True)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Effective only after an asynchronous slew/move call to SlewToTargetAsync(), SlewToCoordinatesAsync(), SlewToAltAzAsync(), or MoveAxis().
- Does nothing if no slew/motion is in progress.
- Tracking is returned to its pre-slew state.

Master Interfaces Reference

Telescope.AbortSlew()

Action (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **ActionName** – A name from SupportedActions that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in SupportedActions.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

`AxisRates` (*Axis: TelescopeAxes*) → List[Rate]

Angular rates at which the mount may be moved with `MoveAxis()`. See Notes.

Returns A list of `Rate` objects, each of which specifies a minimum and a maximum angular rate at which the given axis of the mount may be moved.

Raises

- **`InvalidValueException`** – An invalid axis value is specified.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `MoveAxis()` for details.
- An empty list will be returned if `MoveAxis()` is not supported.
- Returned rates will always be positive, it is up to you to choose the positive or negative rate for your call to `MoveAxis()`.

Master Interfaces Reference

Telescope.AxisRates()

`CanMoveAxis` (*Axis: TelescopeAxes*) → bool

The mount can be moved about the given axis

Raises

- **`InvalidValueException`** – An invalid axis value is specified.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.CanMoveAxis()

CommandBlind (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandBool (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters
- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises
- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

`CommandString` (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **`Command`** – The literal command string to be transmitted.
- **`Raw`** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **`NotImplementedException`** – If no actions at all are supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

`Connect` ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`DestinationSideOfPier` (*RightAscension: float*, *Declination: float*) → PierSide

Predicts the pointing state (PierSide) after a GEM slews to given coordinates at this instant.

Provided so apps can manage GEM flipping during an image sequence. See `SideOfPier`, What is DestinationSideOfPier and why would I want to use it?, and What is the meaning of "pointing state" in the docs for SideOfPier?

Raises

- **`InvalidValueException`** – An invalid axis value is specified.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.DestinationSideOfPier()

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

`FindHome` ( ) → None

Start moving the mount to the "home" position.

Non-blocking: Returns immediately with `Slewing` = True if the homing operation has *successfully* been started, or `Slewing` = False which means the mount is

already at its home position (and of course `AtHome` will already be True). See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – If this feature is not implemented (`CanFindHome` = False)
- **`InvalidOperationException`** – If the mount is parked (`AtPark` = True)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `AtHome` property to monitor the operation. When the mount has *successfully* reached its home position, `Slewing` becomes False and `AtHome` becomes True. See How can I tell if my asynchronous request failed after being started?

Master Interfaces Reference

Telescope.FindHome()

`MoveAxis` (*Axis: TelescopeAxes*, *Rate: float*) → None

Move the mount about the given axis at the given angular rate.

Non-blocking: Returns immediately with `Slewing` = True after *successfully* starting the axis rotation operation. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters

- **`Axis`** – `TelescopeAxes`, the axis about which rotation is desired
- **`Rate`** – The rate or rotation desired (deg/sec)

Raises

- **`NotImplementedException`** – If this feature is not implemented (`CanMoveAxis` = False)
- **`InvalidOperationException`** – If the mount is parked (`AtPark` = True)
- **`InvalidValueException`** – If the axis or rate value is not valid.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to determine if the mount is moving, however you must explicitly call MoveAxis() with a zero rate to stop motion about the given axis.
- This is a complex feature, see What does MoveAxis() do and how do I use it?

Master Interfaces Reference

Telescope.MoveAxis()

`Park` ( ) → None

Start slewing the mount to its park position.

Non-blocking: Returns immediately with `Slewing` = True if the park operation has *successfully* been started, or `Slewing` = False which means the mount is already parked (and of course `AtPark` will already be True). See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – If the mount does not support parking. In this case `CanPark` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`ParkedException`** – If `AtPark` is True
- **`SlavedException`** – If `Slaved` is True
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `AtPark` property to monitor the operation. When the the park position has been *successfully* reached, `AtPark` becomes True, and `Slewing` becomes False. See How can I tell if my asynchronous request failed after being started?
- An app should check `AtPark` before calling Park().

Master Interfaces Reference

Telescope.Park()

`PulseGuide` (*Direction: GuideDirections*, *Duration: int*) → None

Pulse guide in the specified direction for the specified time (ms).

Non-blocking: See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters

- **`Direction`** – `GuideDirections`

- **Interval** – duration of the guide move, milliseconds

Raises

- **InvalidValueException** – If either the direction or the duration are invalid
- **NotImplementedException** – If the mount does not support pulse guiding (CanPulseGuide property is False)
- **NotConnectedException** – If the device is not connected.
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous**: The method returns as soon the pulse-guiding operation has been *successfully* started, with IsPulseGuiding property True. However, you may find that IsPulseGuiding is False when you get around to checking it if the 'pulse' is short. This is still a success if you get False back and not an exception. See How can I tell if my asynchronous request failed after being started?
- Some mounts have implemented this as a Synchronous (blocking) operation. This is deprecated and will be prohbited in the future.
- GuideDirections for North and South have varying interpretations by German Equatorial mounts. Some GEM mounts interpret North to be the same rotation direction of the declination axis regardless of their pointing state ("side of the pier"). Others truly implement North and South by reversing the dec-axis rotation depending on their pointing state. **Apps must be prepared for either behavior**.

Master Interfaces Reference

Telescope.PulseGuide()

SetPark ( ) → None

Set the telescope's park position to its current position.

Raises

- **NotImplementedException** – If the mount does not support the setting of the park position. In this case CanSetPark will be False.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.SetPark()

SlewToAltAz (*Azimuth: float*, *Altitude: float*) → None

DEPRECATED - Do not use this via Alpaca

SlewToAltAzAsync (*Azimuth: float*, *Altitude: float*) → None

Start a slew to the given local horizontal coordinates. See Notes.

Non-blocking: Returns immediately with `Slewing` = True if the slewing operation has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters

- **Azimuth** – Azimuth coordinate (degrees, North-referenced, positive East/clockwise).
- **Altitude** – Altitude coordinate (degrees, positive up).

Raises

- **ParkedException** – If `AtPark` is True
- **InvalidValueException** – If either of the coordinates are invalid
- **NotImplementedException** – If the mount does not support alt /az slewing. In this case `CanSlewAltAz` will be False.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When the the requested coordinates have been *successfully* reached, `Slewing` becomes False. If SlewToAltAzAsync() returns with `Slewing` = False then the mount was already at the requested coordinates, which is also a success See How can I tell if my asynchronous request failed after being started?

Master Interfaces Reference

Telescope.SlewToAltAzAsync()

SlewToCoordinates (*RightAscension: float*, *Declination: float*) → None

DEPRECATED - Do not use this via Alpaca

SlewToCoordinatesAsync (*RightAscension: float*, *Declination: float*)

Start a slew to the given equatorial coordinates. See Notes.

Non-blocking: Returns immediately with `Slewing` = True if the slewing operation has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Parameters

- **RightAscension** – Right Ascension coordinate (hours).
- **Declination** – Declination coordinate (degrees).

Raises

- **ParkedException** – If AtPark is True
- **NotImplementedException** – If the mount does not support async slewing to equatorial coordinates. In this case CanSlewAsync will be False.
- **InvalidValueException** – If either of the coordinates are invalid
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the Slewing property to monitor the operation. When the the requested coordinates have been *successfully* reached, Slewing becomes False. If SlewToCoordinatesAsync() returns with Slewing = False then the mount was already at the requested coordinates, which is also a success See How can I tell if my asynchronous request failed after being started?
- The given coordinates must match the mount's EquatorialSystem.
- The given coordinates are copied to the TargetRightAscension and TargetDeclination properties.

Master Interfaces Reference

Telescope.SlewToCoordinatesAsync()

SlewToTarget ( ) → None

DEPRECATED - Do not use this via Alpaca

SlewToTargetAsync ( ) → None

Start a slew to the coordinates in TargetRightAscension and TargetDeclination.. See Notes.

Non-blocking: Returns immediately with Slewing = True if the slewing operation has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **ParkedException** – If AtPark is True
- **NotImplementedException** – If the mount does not support async slewing to equatorial coordinates. In this case CanSlewAsync will be False.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When the the target coordinates have been *successfully* reached, `Slewing` becomes False. If SlewToCoordinatesAsync() returns with `Slewing` = False then the mount was already at the target coordinates, which is also a success See How can I tell if my asynchronous request failed after being started?

Master Interfaces Reference

Telescope.SlewToTargetAsync()

`SyncToAltAz` (*Azimuth: float*, *Altitude: float*) → None

Match the mount's alt/az coordinates with the given alt/az coordinates

Parameters

- **`Azimuth`** – Corrected Azimuth coordinate (degrees, North-referenced, positive East/clockwise).
- **`Altitude`** – Corrected Altitude coordinate (degrees, positive up).

Raises

- **`ParkedException`** – If `AtPark` is True
- **`InvalidValueException`** – If either of the coordinates are invalid
- **`NotImplementedException`** – If the mount does not support alt /az sync. In this case `CanSyncAltAz` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.SyncToAltAz()

`SyncToCoordinates` (*RightAscension: float*, *Declination: float*) → None

Match the mount's equatorial coordinates with the given equatorial coordinates

Parameters

- **`RightAscension`** – Corrected Right Ascension coordinate (hours).
- **`Declination`** – Corrected Declination coordinate (degrees).

Raises

- **`ParkedException`** – [REVIEW] If `AtPark` is True
- **`NotImplementedException`** – If the mount does not support equatorial coordinate synchronization. In this case `CanSync` will be False.
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.SyncToCoordinates()

SyncToTarget ( ) → None

Match the mount's equatorial coordinates with :attr:TargetRightAscension and TargetDeclination.

Raises

- **ParkedException** – If AtPark is True
- **NotImplementedException** – If the mount does not support equatorial coordinate synchronization. In this case CanSync will be False.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.SyncToTarget()

Unpark ( ) → None

Takes the mount out of parked state

Raises

- **NotImplementedException** – If this method is not implemented. In this case CanUnpark will be False.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Unparking a mount that is not parked is harmless and will always be successful.

Master Interfaces Reference

Telescope.Unpark()

property AlignmentMode: AlignmentModes

The current mount alignment mode.

Raises

- **NotImplementedException** – If the mount cannot report its alignment mode.

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.AlignmentMode

property Altitude: float

The mount's current Altitude (degrees) above the horizon.

Raises

- **NotImplementedException** – Alt-Az not implemented by the device
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.Altitude

property ApertureArea: float

The telescope's aperture area (square meters).

Raises

- **NotImplementedException** – Not implemented by the device
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The area takes into account any obstructions; it is the actual light-gathering area.

Master Interfaces Reference

Telescope.ApertureArea

property ApertureDiameter: float

Return the telescope's effective aperture (meters).

Raises

- **NotImplementedException** – Alt-Az not implemented by the device
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.ApertureDiameter

property `AtHome`: bool

The mount is at the home position.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the correct property to use to determine *successful* completion of the (non-blocking) `FindHome()` operation. See How can I tell if my asynchronous request failed after being started?
- True if the telescope is stopped in the Home position. Can be True only following a FindHome() operation.
- Will become False immediately upon any slewing operation
- Will always be False if the telescope does not support homing. Use `CanFindHome` to determine if the mount supports homing.

Master Interfaces Reference

Telescope.AtHome

property `AtPark`: bool

The telescope is at the park position.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the correct property to use to determine *successful* completion of the (non-blocking) `Park()` operation. See How can I tell if my asynchronous request failed after being started?
- True if the telescope is stopped in the Park position. Can be True only following successful completion of a `Park()` operation.
- When parked, the telescope will be stationary or restricted to a small safe range of movement. `Tracking` will be False.
- You must take the telescope out of park by calling `Unpark()`; attempts to slew enabling tracking while parked will raise a `ParkedException`.
- Will always be False if the telescope does not support parking. Use `CanPark` to determine if the mount supports parking.

Master Interfaces Reference

Telescope.AtPark

property `Azimuth`: float

The azimuth (degrees) at which the telescope is currently pointing.

Raises

- **`NotImplementedException`** – Alt-Az not implemented by the device
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Azimuth is per the usual alt/az coordinate convention: degrees North-referenced, positive East/clockwise.

Master Interfaces Reference

Telescope.Azimuth

property `CanFindHome`: bool

The mount can find its home position.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `FindHome()`

Master Interfaces Reference

Telescope.CanFindHome

property `CanPark`: bool

The mount can be parked.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `Park()`

Master Interfaces Reference

Telescope.CanPark

property `CanPulseGuide`: bool

The mount can be pulse guided.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `PulseGuide`

Master Interfaces Reference

Telescope.CanPulseGuide

property `CanSetDeclinationRate`: bool

The Declination tracking rate may be offset.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `DeclinationRate`

Master Interfaces Reference

Telescope.CanSetDeclinationRate

property `CanSetGuideRates`: bool

The guiding rates for `PulseGuide()` can be adjusted

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `PulseGuide()`.

Master Interfaces Reference

Telescope.CanSetGuideRates

property `CanSetPark`: bool

The mount's park position can be set.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `SetPark()`

Master Interfaces Reference

Telescope.CanSetPark

property `CanSetPierSide`: bool

The mount can be force-flipped via setting `SideOfPier`.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `SideOfPier`.
- Will always be False for non-German mounts

Master Interfaces Reference

Telescope.CanSetPierSide

property `CanSetRightAscensionRate`: bool

The Right Ascension tracking rate may be offset

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `RightAscensionRate`.

Master Interfaces Reference

Telescope.CanSetRightAscensionRate

property `CanSetTracking`: bool

The mount's sidereal tracking may be turned on and off

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `Tracking`.

Master Interfaces Reference

Telescope.CanSetTracking

property `CanSlew`: bool

The mount can slew to equatorial coordinates.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `SlewToCoordinates()`, `SlewToCoordinatesAsync()` `SlewToTarget()`, and `SlewToTargetAsync()`.

Attention!

Do not use synchronous methods unless the mount cannot do asynchronous slewing (`CanSlewAsync` = False). Synchronous methods will be deprecated in the next version of ITelescope.

Master Interfaces Reference

Telescope.CanSlew

property `CanSlewAltAz`: bool

The mount can slew to alt/az coordinates.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `SlewToAltAz()` and `SlewToAltAzAsync()`.

Attention!

Do not use synchronous methods unless the mount cannot do asynchronous slewing (`CanSlewAltAzAsync` = False). Synchronous methods will be deprecated in the next version of ITelescope.

Master Interfaces Reference

Telescope.CanSlewAltAz

property `CanSlewAltAzAsync`: bool

The mount can slew to alt/az coordinates asynchronously.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- `CanSlewAltAz` will be True if CanSlewAltAzAsync is True.
- See `SlewToAltAzAsync()`.

Attention!

Always use asynchronous slewing if at all possible (CanSlewAltAzAsync = True). Synchronous methods will be deprecated in the next version of ITelescope.

Master Interfaces Reference

Telescope.CanSlewAltAzAsync

property `CanSlewAsync`: bool

The mount can slew to equatorial coordinates synchronously.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- `CanSlew` will be True if CanSlewAsync is True.
- See `SlewToCoordinatesAsync()` and `SlewToTargetAsync()`.

Attention!

Always use asynchronous slewing if at all possible (CanSlewAsync = True). Synchronous methods will be deprecated in the next version of ITelescope.

Master Interfaces Reference

Telescope.CanSlewAsync

property `CanSync`: bool

The mount can be synchronized to equatorial coordinates.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `SyncToCoordinates()`.

Master Interfaces Reference

Telescope.CanSync

property CanSyncAltAz: bool

The mount can be synchronized to alt/az coordinates.

Raises
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See SyncToAltAz().

Master Interfaces Reference

Telescope.CanSyncAltAz

property CanUnpark: bool

The mount can be unparked

Raises
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See Unpark() and Park().

Master Interfaces Reference

Telescope.CanUnpark

property Connected: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Declination`: float

The mount's current Declination (degrees) in the current `EquatorialSystem` (see Notes)

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- Declination will be in the equinox given by the current value of `EquatorialSystem`.

Master Interfaces Reference

Telescope.Declination

property `DeclinationRate`: float

(Read/Write) The mount's declination tracking rate (see Notes).

Raises

- **`NotImplementedException`** – If `CanSetDeclinationRate` is False,
- **`yet an attempt is made to write to this property.`** –
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- DeclinationRate is an offset from 0 (no change in declination), given in arc seconds per SI (atomic) second. (Please note that the units of `RightAscensionRate` are in (sidereal) seconds of RA per *sidereal* second).
- The supported range for this property is mount-specific.
- Offset tracking is most commonly used to track a solar system object such as a minor planet or comet.
- Offset tracking may also be used (less commonly) as a method for reducing dynamic mount errors.
- If offset tracking is in effect (non-zero), and a slew is initiated, the mount will continue to update the slew destination coordinates at the given offset rate.

Master Interfaces Reference

Telescope.DeclinationRate What are RightAscensionRate and DeclinationRate and how are they used?

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property DeviceState: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property DoesRefraction: bool

(Read/Write) The mount applies atmospheric refraction to corrections

Raises

- **NotImplementedException** – If either reading or writing of this property is not implemented
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- If the driver does not know whether the attached telescope does its own refraction, and if the driver does not itself calculate refraction, this property (if implemented) will raise `DriverException` when read.
- If the mount indicates that it can apply refraction, yet you wish to calculate your own (more accurate) correction, try setting this to False then, if successful, supply your own refracted coordinates.
- If you set this to True, and the mount (already) does refraction, or if you set this to Fales, and the mount (already) does not do refraction, no exception will be raised.

Master Interfaces Reference

Telescope.DoesRefraction

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**
Common to all devices

Returns Python list of strings (see Notes)

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.
Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `EquatorialSystem`: EquatorialCoordinateType

The current equatorial coordinate system used by the mount

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- See `EquatorialCoordinateType`.
- Most mounts use topocentric coordinates. Some high-end research mounts use J2000 coordinates.

Master Interfaces Reference

Telescope.EquatorialSystem

property `FocalLength`: float

Return the telescope's focal length in meters.

Raises

- **`NotImplementedException`** – Focal length is not available from the mount
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.FocalLength

property GuideRateDeclination: float

(Read/Write) The current Declination rate offset (deg/sec) for guiding.

Raises

- **InvalidValueException** – If an invalid guide rate is set
- **NotImplementedException** – Rate cannot be set, CanSetGuideRates = False
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the rate for both hardware/relay guiding and for PulseGuide().
- The mount may not support separate right ascension and declination guide rates. If so, setting either rate will set the other to the same value.
- This value will be set to a default upon startup.

Master Interfaces Reference

Telescope.GuideRateDeclination

property GuideRateRightAscension: float

(Read/Write) The current Right Ascension rate offset (deg/sec) for guiding.

Raises

- **InvalidValueException** – If an invalid guide rate is set
- **NotImplementedException** – Rate cannot be set, CanSetGuideRates = False
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the rate for both hardware/relay guiding and for PulseGuide().
- The mount may not support separate right ascension and declination guide rates. If so, setting either rate will set the other to the same value.
- This value will be set to a default upon startup.

Master Interfaces Reference

Telescope.GuideRateRightAscension

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `IsPulseGuiding`: bool

The mount is currently executing a `PulseGuide()` command.

Use this property to determine when a (non-blocking) pulse guide command has completed. See Notes and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – Pulse guiding is not supported
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- A pulse guide command may be so short that you won't see this equal to True. If you can read False after calling `PulseGuide()`, then you know it completed *successfully*. See How can I tell if my asynchronous request failed after being started?

Master Interfaces Reference

Telescope.IsPulseGuiding

property Name: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises **DriverException** – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

> **Master Interfaces Reference**
>
> Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for Name, find this specific device's specification, and see Name there.

property RightAscension: float

The mount's current right ascension (hours) in the current EquatorialSystem. See Notes.

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

> **Master Interfaces Reference**
>
> Telescope.RightAscension

property RightAscensionRate: float

(Read/Write) Read or set a secular rate of change to the mount's RightAscension (seconds of RA per **sidereal** second). See

Raises

- **NotImplementedException** – If CanSetRightAscensionRate is False,
- **yet an attempt is made to write to this property.** –
- **InvalidOperationException** – If the current TrackingRate is not driveSidereal.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- RightAscensionRate is an offset from 0 (no change in Right Ascension). Note that a mount that is tracking sidereally is pointing to an *unchanging* right ascension. See
- Right Ascension is a *time* coordinate not an angular coordinate. Seconds are not arc seconds.
- To convert a given rate in units of sidereal *seconds* per UTC (clock) second, multiply the value by 0.9972695677 (the number of UTC seconds in a sidereal second) then set the RightAscensionRate property.
- The supported range for this property is mount-specific.
- Offset tracking is most commonly used to track a solar system object such as a minor planet or comet.
- If offset tracking is in effect (non-zero), and a slew is initiated, the mount will continue to update the slew destination coordinates at the given offset rate.
- Use the `Tracking` property to stop and start tracking.

Master Interfaces Reference

Telescope.RightAscensionRate What are RightAscensionRate and DeclinationRate and how are they used?

property `SideOfPier`: PierSide

(Read/Write) Start a change of, or return, the mount's pointing state. See What is the meaning of "pointing state" in the docs for SideOfPier?

Non-blocking: Writing to *change* pointing state returns immediately with `Slewing` = True if the state change (e.g. GEM flip) operation has *successfully* been started. See Notes, and How can I tell if my asynchronous request failed after being started?

Raises

- **`NotImplementedException`** – If the mount does not report its pointing state, at all, or if it doesn't support changing pointing state (e.g.force-flipping) by writing to SideOfPier (`CanSetPierSide` = False).
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Asynchronous** (non-blocking) if writing SideOfPier to force a pointing state change (e.g. GEM flip): Use the `Slewing` property to monitor the operation. When the pointing state change has been *successfully* completed, `Slewing` becomes False. If writing SideOfPier returns with `Slewing` = False then the mount was already in the requested pointing state, which is also a success. See How can I tell if my asynchronous request failed after being started?
- May optionally be written-to to force a flip on a German mount
- See What is the meaning of "pointing state" in the docs for SideOfPier?

Master Interfaces Reference

Telescope.SideOfPier

property `SiderealTime`: float

Local apparent sidereal time (See Notes)

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- It is required for a driver to calculate this from the system clock if the mount has no accessible source of sidereal time.
- Local Apparent Sidereal Time is the sidereal time used for pointing telescopes, and thus must be calculated from the Greenwich Mean Sidereal time, longitude, nutation in longitude and true ecliptic obliquity.

Master Interfaces Reference

Telescope.SiderealTime

property `SiteElevation`: float

(Read/Write) The observing site's elevation (meters) above mean sea level.

Raises

- **`NotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is outside the range -300 through 10000 meters.
- **`InvalidOperationException`** – If the application must set the SiteElevation before reading it, but has not. See Notes.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of

the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Some mounts supply this via input to their control systems, in other scenarios the application will set this on initialization.
- If a change is made via SiteElevation, most mounts will save the value persistently across power off/on.
- If the value hasn't been set by any means, an InvalidOperationException will be raised.

Master Interfaces Reference

Telescope.SiteElevation

property `SiteLatitude`: float

(Read/Write) The latitude (degrees) of the observing site. See Notes.

Raises

- **`NotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is outside the range -90 through 90 degrees.
- **`InvalidOperationException`** – If the application must set the SiteLatitude before reading it, but has not. See Notes.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is geodetic (map) latitude, degrees, WGS84, positive North.
- Some mounts supply this via input to their control systems, in other scenarios the application will set this on initialization.
- If a change is made via SiteLatitude, most mounts will save the value persistently across power off/on.
- If the value hasn't been set by any means, an InvalidOperationException will be raised.

Master Interfaces Reference

Telescope.SiteLatitude

property `SiteLongitude`: float

(Read/Write) The longitude (degrees) of the observing site. See Notes.

Raises

- **`NotImplementedException`** – If the property is not implemented

- **InvalidValueException** – If the given value is outside the range -180 through 180 degrees.
- **InvalidOperationException** – If the application must set the SiteLatitude before reading it, but has not. See Notes.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is geodetic (map) longitude, degrees, WGS84, **positive East**.
- Some mounts supply this via input to their control systems, in other scenarios the application will set this on initialization.
- If a change is made via SiteLongitude, most mounts will save the value persistently across power off/on.
- If the value hasn't been set by any means, an InvalidOperationException will be raised.

Attention!

West longitude is negative.

Master Interfaces Reference

Telescope.SiteLongitude

property SlewSettleTime: int

(Read/Write) The post-slew settling time (seconds).
Artificially lengthen all slewing operations. Useful for mounts or buildings that require additional mechanical settling time after a slew to stabilize.

Raises

- **NotImplementedException** – If the property is not implemented
- **InvalidValueException** – If the given settling time is invalid (negative or ridiculously high)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Telescope.SlewSettleTime

property Slewing: bool

The mount is in motion resulting from a slew or a move-axis. See How can I tell if my

asynchronous request failed after being started?

Raises

- **NotImplementedException** – If the property is not implemented (none of the CanSlew properties are True, this is a manual mount)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is the correct property to use to determine *successful* completion of a (non-blocking) `SlewToCoordinatesAsync()`, `SlewToTargetAsync()`, `SlewToCoordinatesAsync()`, or by writing to `SideOfPier` to force a flip. See How can I tell if my asynchronous request failed after being started?
- Slewing will be True immediately upon returning from any of these calls, and will remain True until *successful* completion, at which time Slewing will become False.
- You might see Slewing = False on returning from a slew or move-axis if the operation takes a very short time. If you see False (and not an exception) in this state, you can be certain that the operation completed *successfully*.
- Slewing will not be True during pulse-guiding or application of tracking offsets.

Master Interfaces Reference

Telescope.Slewing

property SupportedActions: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

property `TargetDeclination`: float

(Read/Write) Set or return the target declination in the current `EquatorialSystem`. See Notes.

Raises

- **`NotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is outside the range -90 through 90 degrees.
- **`InvalidOperationException`** – If the application must set the TargetDeclination before reading it, but has not. See Notes.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a pre-set target coordinate for `SlewToTargetAsync()` and `SyncToTarget()`
- Target coordinates are for the current `EquatorialSystem`.

Master Interfaces Reference

Telescope.TargetDeclination

property `TargetRightAscension`: float

(Read/Write) Set or return the target right ascension (hours) in the current `EquatorialSystem`

Raises

- **`NotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is outside the range 0 to 24 hours.
- **`InvalidOperationException`** – If the application must set the TargetRightAscension before reading it, but has not. See Notes.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a pre-set target coordinate for `SlewToTargetAsync()` and `SyncToTarget()`
- Target coordinates are for the current `EquatorialSystem`.

Master Interfaces Reference

Telescope.TargetRightAscension

property `Tracking`: bool

(Read/Write) The on/off state of the mount's sidereal tracking drive. See Notes.

Raises

- **`NotImplementedException`** – If writing to the property is not implemented. `CanSetTracking` will be False.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- When on, the mount will use the last selected `TrackingRate`.
- Even if the mount doesn't support changing this, it will report the current state.

Master Interfaces Reference

Telescope.Tracking

property `TrackingRate`: DriveRates

(Read/Write) The current (sidereal) tracking rate of the mount, from `DriveRates`.

See Notes.

Raises

- **InvalidValueException** – If value being written is not one of the DriveRates, or if the requested rate is not supported by the mount (not all are).
- **NotImplementedException** – If the mount doesn't support writing this property to change the tracking rate.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Even if the mount doesn't support changing this, it will report the current state.
- If this is any rate other than driveSidereal then RightAscensionRate and DeclinationRate will raise InvalidOperationException.

Master Interfaces Reference

Telescope.TrackingRate

property TrackingRates: List[DriveRates]

Return a list of supported DriveRates values

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- At a minimum, this list will contain an item for driveSidereal

Master Interfaces Reference

Telescope.TrackingRates

property UTCDate: datetime

(Read/Write) The UTC date/time of the mount's internal clock. See Notes.
You may write either a Python datetime (tz=UTC) or an ISO 8601 string for example: 2022-04-22T20:21:01.123 Z

Raises

- **InvalidValueException** – if an illegal ISO 8601 string or a bad Python datetime value is written to change the time. See Notes.
- **NotImplementedException** – If the mount doesn't support writing this property to change the UTC time

- **InvalidOperationException** – When UTCDate is read and the mount cannot provide this property itslef and a value has not yet be established by writing to the property.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Changing time by writing to this property can be done with either a Python datetime value or an ISO 8601 string, for example `2022-04-22T20:21:01.123Z`. Note that the *Z* is required; numerical offset forms may be confused between local and UTC times.
- The date/time must be UTC not Local.
- Even if the mount doesn't support changing this, it will report the current UTC date/time. The value may be derived from the system clock by the driver if the mount doesn't provide it.
- If the mount's UTC date/time is being derived from the system clock, you will not be able to write this (you'll get `NotImplementedException`).

Master Interfaces Reference

Telescope.UTCDate

3.10.1 Rate Class

class `alpaca.telescope.`**`Rate`** (*maxv: float*, *minv: float*)

Bases: `object`

Describes a range of rates supported by the `MoveAxis()` method

property `Maximum`: float

The maximum rate (degrees per second)

property `Minimum`: float

The minimum rate (degrees per second)

3.10.2 Telescope-Related Constants

enum `alpaca.telescope.`**`AlignmentModes`** (*value*)

Bases: `DocIntEnum`

The geometry of the mount

Member Type `int`

Valid values are as follows:

`algAltAz` = <AlignmentModes.algAltAz: 0>

Altitude-Azimuth alignment

algPolar = <AlignmentModes.algPolar: 1>
Polar (equatorial) mount other than German equatorial.

algGermanPolar = <AlignmentModes.algGermanPolar: 2>
German equatorial mount.

enum alpaca.telescope.**DriveRates** (*value*)
Bases: DocIntEnum
Well-known telescope tracking rates

Member Type int
Valid values are as follows:

driveSidereal = <DriveRates.driveSidereal: 0>
Sidereal tracking rate (15.041 arcseconds per second).

driveLunar = <DriveRates.driveLunar: 1>
Lunar tracking rate (14.685 arcseconds per second).

driveSolar = <DriveRates.driveSolar: 2>
Solar tracking rate (15.0 arcseconds per second).

driveKing = <DriveRates.driveKing: 3>
King tracking rate (15.0369 arcseconds per second).

enum alpaca.telescope.**EquatorialCoordinateType** (*value*)
Bases: DocIntEnum
Equatorial coordinate systems used by telescopes.

Member Type int
Valid values are as follows:

equOther = <EquatorialCoordinateType.equOther: 0>
Custom or unknown equinox and/or reference frame.

equTopocentric = <EquatorialCoordinateType.equTopocentric: 1>
Topocentric coordinates. Coordinates of the object at the current date having allowed for annual aberration, precession and nutation. This is the most common coordinate type for amateur telescopes.

equJ2000 = <EquatorialCoordinateType.equJ2000: 2>
J2000 equator/equinox. Coordinates of the object at mid-day on 1st January 2000, ICRS reference frame.

equJ2050 = <EquatorialCoordinateType.equJ2050: 3>
J2050 equator/equinox, ICRS reference frame.

equB1950 = <EquatorialCoordinateType.equB1950: 4>
B1950 equinox, FK4 reference frame.

enum alpaca.telescope.**GuideDirections** (*value*)
Bases: DocIntEnum

The direction in which the guide-rate motion is to be made.

Member Type `int`
Valid values are as follows:

`guideNorth` = <GuideDirections.guideNorth: 0>
North (+ declination/altitude).

`guideSouth` = <GuideDirections.guideSouth: 1>
South (- declination/altitude).

`guideEast` = <GuideDirections.guideEast: 2>
East (+ right ascension/azimuth).

`guideWest` = <GuideDirections.guideWest: 3>
West (- right ascension/azimuth).

enum `alpaca.telescope.`**`PierSide`** (*value*)
Bases: `DocIntEnum`
The pointing state of the mount

Member Type `int`
Valid values are as follows:

`pierEast` = <PierSide.pierEast: 0>
Normal pointing state - Mount on the East side of pier (looking West)

`pierWest` = <PierSide.pierWest: 1>
Through the pole pointing state - Mount on the West side of pier (looking East)

`pierUnknown` = <PierSide.pierUnknown: -1>
Unknown or indeterminate.

enum `alpaca.telescope.`**`TelescopeAxes`** (*value*)
Bases: `DocIntEnum`

Member Type `int`
Valid values are as follows:

`axisPrimary` = <TelescopeAxes.axisPrimary: 0>
Primary axis (e.g., Right Ascension or Azimuth).

`axisSecondary` = <TelescopeAxes.axisSecondary: 1>
Secondary axis (e.g., Declination or Altitude).

`axisTertiary` = <TelescopeAxes.axisTertiary: 2>
Tertiary axis (e.g. imager rotator/de-rotator).

3.11 Device Superclass

This contains methods and properties that are shared by all ASCOM/Alpaca classes. Its members appear within the ASCOM/Alpaca class documentation as well as here.

In addition, this class contains the low-level HTTP I/O used to communicate within Alpaca devices.

class alpaca.device.**Device** (*address: str*, *device_type: str*, *device_number: int*, *protocol: str*)

Bases: object

Common interface members across all ASCOM Alpaca devices.

Initialize Device object.

address

Domain name or IP address of Alpaca server. Can also specify port number if needed. localhost forced to be IPv4 127.0.0.1

device_type

One of the recognised ASCOM device types e.g. telescope (must be lower case).

device_number

Zero based device number as set on the server (0 to 4294967295).

protocol

Protocol (http vs https) used to communicate with Alpaca server.

api_version

Alpaca API version.

base_url

Basic URL to easily append with commands.

Note

- Sets a random number for ClientID that lasts
- 'localhost' is forced to use IPv4 for device usage

Action (*ActionName: str*, **Parameters*) → str

Invoke the specified device-specific custom action

Common to all devices

Parameters

- **ActionName** – A name from SupportedActions that represents the action to be carried out.
- ***Parameters** – List of required parameters or [] if none are required.

Returns String result of the action.

Raises

- **NotImplementedException** – If no actions at all are supported
- **ActionNotImplementedException** – If the driver does not support the
- **requested** – ActionName. The supported action names are listed in SupportedActions.
- **NotConnectedException** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Action()`, find this specific device's specification, and see `Action()` there.

CommandBlind (*Command: str*, *Raw: bool*) → None

Transmit an arbitrary string to the device and does not wait for a response.

Common to all devices

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandBool (*Command: str*, *Raw: bool*) → bool

Transmit an arbitrary string to the device and wait for a boolean response.

Common to all devices

Returns The True/False response from the command

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandBlind()`, find this specific device's specification, and see `CommandBlind()` there.

CommandString (*Command: str*, *Raw: bool*) → str

Transmit an arbitrary string to the device and wait for a string response.

Common to all devices

Returns The string response from the command

Parameters

- **Command** – The literal command string to be transmitted.
- **Raw** – If true, command is transmitted 'as-is'. If false, then protocol framing characters may be added prior to transmission.

Raises

- **NotImplementedException** – If no actions at all are supported
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Attention!

Deprecated, will most likely result in `NotImplementedException`

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `CommandString()`, find this specific device's specification, and see `CommandString()` there.

Connect ( ) → None

Connect to the device **asynchronously**.

Common to all devices

Returns Nothing

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the . To see the reference info for `Connect()`, find this specific device's specification, and see `Connect()` there.

`Disconnect` ( ) → None

Disconnect from the device **asynchronously**.

Common to all devices

Returns Nothing

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- **Non-Blocking** Use `Connecting` to indicate completion.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Disconnect()`, find this specific device's specification, and see `Disconnect()` there.

property `Connected`: bool

(Read/Write) Retrieve or set the connected state of the device.

Common to all devices

Set True to connect to the device hardware. Set False to disconnect from the device hardware. You can also read the property to check whether it is connected. This reports the current hardware state. See Notes below.

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- The `Connected` property sets and reports the state of connection to the device hardware. For a hub this means that `Connected` will be `True` when the first driver connects and will only be set to False when all drivers have disconnected. A second driver may find that `Connected` is already `True` and setting `Connected` to False does not report `Connected` as False. This is not an error because the physical state is that the hardware connection is still `True`.
- Multiple calls setting `Connected` to `True` or false will not cause an error.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connected`, find this specific device's specification, and see `Connected` there.

property `Connecting`: bool

Returns `True` while the device is undertaking an asynchronous `Connect()` or `Disconnect()` operation.

Common to all devices

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- Use this property to determine when an (async) `Connect()` or `Disconnect()` has completed, at which time it will transition from `True` to `False`.
- Natively present only in Platform 7 (2024) devices, but this library emulates Connect()/Disconnect()/Connecting mechanic for older devices.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Connecting`, find this specific device's specification, and see `Connecting` there.

property `Description`: str

Description of the **device** such as manufacturer and model number.

Common to all devices

Raises

- **`NotConnectedException`** – If the device status is unavailable
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully*

complete the request.

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length will be a maximum of 64 characters so that it can be used in FITS image headers, which are limited to 80 characters including the header name.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Description`, find this specific device's specification, and see `Description` there.

property `DeviceState`: List[dict]

List of key-value pairs representing the operational properties of the device

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DeviceState`, find this specific device's specification, and see `DeviceState` there.

property `DriverInfo`: List[str]

Descriptive and version information about the ASCOM **driver**

Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This describes the *driver* not the device. See the `Description` property for information on the device itself
- The return is a Python list of strings, the total length of which may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM (COM or Alpaca) driver, including version and copyright data. . To get the driver version in a parse-able string, use the `DriverVersion` property.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverInfo`, find this specific device's specification, and see `DriverInfo` there.

property `DriverVersion`: str

String containing only the major and minor version of the *driver*.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This must be in the form "n.n". It should not to be confused with the `InterfaceVersion` property, which is the version of this specification supported by the driver. **Note:** on systems with a comma as the decimal point you may need to make accommodations to parse the value.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `DriverVersion`, find this specific device's specification, and see `DriverVersion` there.

property `InterfaceVersion`: int

ASCOM Device interface definition version that this device supports.

Common to all devices

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This is a single integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV3, this will be 3. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- This value is cached internally after first retrieval since it is repeatedly used if emulating Connect/Disconnect semantics on older (pre - Platform 7) devioces.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `InterfaceVersion`, find this specific device's specification, and see `InterfaceVersion` there.

property `Name`: str

The short name of the *driver*, for display purposes.

Common to all devices

Raises `DriverException` – If the driver cannot *successfully* complete the request. This exception may be encountered on any call to the device.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `Name`, find this specific device's specification, and see `Name` there.

property `SupportedActions`: List[str]

The list of custom action names supported by this driver

Common to all devices

Returns Python list of strings (see Notes)

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. The device did not *successfully* complete the request.

Note

- This method, combined with `Action()`, is the supported mechanic for adding non-standard functionality.
- SupportedActions is a "discovery" mechanism that enables clients to know which Actions a device supports without having to exercise the Actions themselves. This mechanism is necessary because there could be people / equipment safety issues if actions are called unexpectedly or out of a defined process sequence. It follows from this that SupportedActions must return names that match the spelling of `Action()` names exactly, without additional descriptive text. However, returned names may use any casing because the ActionName parameter of `Action()` is case insensitive.

Master Interfaces Reference

Alpyca uses a common Device class but this is not available in the ASCOM Master Interface Definitions (external). To see the reference info for `SupportedActions`, find this specific device's specification, and see `SupportedActions` there.

ASCOM Alpaca Exception Classes

These exception classes are defined in the Alpaca API Reference and ASCOM Master Interfaces (Alpaca and COM) on the ASCOM main web site.

4.1 Exception Characteristics

When these ASCOM Alpaca specific exceptions are raised by the device, the internal logic reconstructs them as Python exceptions from the incoming device response JSON. Thus you get the same exceptions by name and value as defined in the ASCOM API.

Internal Arguments

It's common to access the message of a built-in Python exception as provided by `catch ex:` as an `ex.args` array. These custom exceptions define `ex.args[0]` as the error number and `ex.args[1]` as the error message.

Properties

The exceptions not only have number and message attributes like built-in Python exception objects, but they also have `number` and `message` properties. Within an exception as provided by a `catch ex:` you can get the error number as `ex.number` and the error message as `ex.message`.

String Conversion

Like built-in Python exceptions, you can print out the exception parts via `str(ex)` which produces a string with the the error message then the error number in parentheses. For example, a `DriverException` with code `0x506`:

```
Datalink connect failure (Error Code: 0x506)
```

As usual you can get the exception name via `type(ex).__name__`.

Default Uncaught Exceptions

Like built-in Python exceptions, the console output for uncaught exceptions prints the name of the exception and its error number and message as formatted by `str(ex)`, for example:

```
alpaca.exceptions.DriverException: Datalink connect failure
(Error Code: 0x506)
```

4.2 Exception Definitions

exception alpaca.exceptions.**ActionNotImplementedException** (*message: str*)

Bases: Exception

Exception thrown by a device when it receives an unknown command through the Action method.

Properties:

- number (int): Constant 0x040C (1036)
- message (str): Text of the error message

exception alpaca.exceptions.**AlpacaRequestException** (*number: int*, *message: str*)

Bases: Exception

Raised by the device's Alpaca server for unknown or illegal requests.

Properties:

- number (int): The HTTP response code (4xx or 5xx)
- message (str): The concatenation of the server's response text and the URL.

exception alpaca.exceptions.**DriverException** (*number: int*, *message: str*)

Bases: Exception

Generic driver exception. See note below.

Properties:

- number (int): Assigned by the device and will be a number from 0x500 - 0xFFF
- message (str): Text of the error message

> **Note**
>
> This is the generic driver exception. Drivers are permitted to directly throw these exceptions. This exception should only be thrown if there is no other more appropriate exception as listed here are already defined. These specific exceptions should be thrown where appropriate rather than using the more generic DriverException. Conform will not accept DriverExceptions where more appropriate exceptionsare already defined.

exception alpaca.exceptions.**InvalidOperationException** (*message: str*)

Bases: Exception

Thrown by the device to reject a command from the client.

Properties:

- number (int): Constant 0x40B (1035)
- message (str): Text of the error message

exception alpaca.exceptions.**InvalidValueException** (*message: str*)

Bases: Exception

Exception to report an invalid value supplied to a device.

Properties:

- number (int): Constant 0x401 (1025)
- message(str): Text of the error message

exception alpaca.exceptions.**NotConnectedException** (*message: str*)

Bases: Exception

An operation is attempted that requires communication with the device, but the device is disconnected.

Properties:

- number (int): Constant 0x407 (1031)
- message(str): Text of the error message

This refers to the driver not being connected to the device. It is not for network outages or bad URLs.

exception alpaca.exceptions.**NotImplementedException** (*message: str*)

Bases: Exception

Property or method is not implemented in the device

Properties:

- number (int): Constant 0x400 (1024)
- message(str): Text of the error message

exception alpaca.exceptions.**OperationCancelledException** (*message: str*)

Bases: Exception

An (asynchronous) in-progress operation has been cancelled.

Properties:

- number (int): Constant 0x40E (1038)
- message(str): Text of the error message

exception alpaca.exceptions.**ParkedException** (*message: str*)

Bases: Exception

Movement (or other invalid operation) was attempted while the device was in a parked state.

Properties:

- number (int): Constant 0x408 (1032)
- message(str): Text of the error message

exception alpaca.exceptions.**SlavedException** (*message: str*)

Bases: Exception

Movement (or other invalid operation) was attempted while the device was in slaved mode. This applies primarily to Dome drivers.

Properties:

- number (int): Constant 0x409 (1033)
- message(str): Text of the error message

exception alpaca.exceptions.**ValueNotSetException** (*message: str*)

Bases: Exception

No value has yet been set for this property.

Properties:

- number (int): Constant 0x402 (1026)
- message(str): Text of the error message

Alpaca Device Server Discovery

This module provides Alpaca device server discovery service. Search your local network segment (or VLAN) for Alpaca device *servers*, returning a list consisting of `ipaddress:port` strings for each one found. Each Alpaca device server may provide access to multiple Alpaca device types, and multiple Alpaca devices of a given type.

Note

Use the Alpaca Device Server Management functions to learn the details of the served device(s). See the example there.

Example:

```
from alpaca import discovery

svrs = discovery.search_ipv4()  # Note there is an IPv6 function as well
print(svrs)
```

Output:

```
['127.0.0.1:32323', '192.168.1.12:11111', '192.168.1.31:11111']
```

This example shows one Alpaca server on the local host, two Alpaca servers on the LAN.

`alpaca.discovery.`**`search_ipv4`** (*numquery: int = 2, timeout: int = 2, trace: bool = False*) → List[str]

Discover Alpaca device servers on the IPV4 LAN/VLAN

Returns a list of strings of the form `ipaddress:port`, each corresponding to a discovered Alpaca device server. Use Alpaca Device Server Management functions to enumerate the devices.

Parameters

- **`numquery`** – Number of discovery queries to send (default 2)
- **`timeout`** – Time (sec.) to allow for responses to each discovery query. Optional, defaults to 2 seconds.
- **`trace`** – Output a trace of the discovery process. Optional, defaults to False.

Raises **`To be determined.`** –

Note

- This function uses IPV4
- UDP protocol restricted to the LAN/VLAN is used to perform the query.
- Adapters which show an APIPA address (169.254.*.*) are skipped in order to avoid a guaranteed timeout trying to receive the datagram replies. See Why Do I Have the 169.254 IP Address?
- See section 4 of the Alpaca API Reference for Discovery details.

`alpaca.discovery.`**`search_ipv6`** (*numquery: int = 2*, *timeout: int = 2*, *trace: bool = False*) → List[str]

Discover Alpaca device servers on the IPV6 LAN/VLAN

Returns a list of strings of the form `[ipv6address%intfc]:port`, each corresponding to a discovered Alpaca device server. Use Alpaca Device Server Management functions to enumerate the devices.

Parameters

- **`numquery`** – Number of discovery queries to send (default 2)
- **`timeout`** – Time (sec.) to allow for responses to the discovery query. Optional, defaults to 2 seconds.
- **`trace`** – Output a trace of the discovery process. Optional, defaults to False.

Raises **`To be determined.`** –

Note

- This function uses IPV6
- UDP protocol, restricted link-local addresses to the LAN/VLAN attached to each interface, is used to perform the query. Does not query global IPv6.
- ISATAP addresses are specifically excluded.
- See section 4 of the Alpaca API Reference for Discovery details.

Alpaca Device Server Management

Provides information about an Alpaca device server found via Alpaca Device Server Discovery, and the devices which are provided by that server. For more information see the .

Example using the Management functions:

```
svrs = discovery.search_ipv4()
print(svrs)
for svr in svrs:
    print(f"At {svr}")
    print (f"  V{management.apiversions(svr)} server")
    print (f"  {management.description(svr)['ServerName']}")
    devs = management.configureddevices(svr)
    for dev in devs:
        print(f"    {dev['DeviceType']}[{dev['DeviceNumber']}]:
 {dev['DeviceName']}")
```

Output:

```
['127.0.0.1:32323', '127.0.0.1:11111']
At 127.0.0.1:32323
    V[1] server
    ASCOM Alpaca Simulators
        Camera[0]: Alpaca Camera Sim
        CoverCalibrator[0]: Alpaca CoverCalibrator Simulator
        Dome[0]: Alpaca Dome Sim
        FilterWheel[0]: Alpaca Filter Wheel Sim
        Focuser[0]: Alpaca Focuser Sim
        ObservingConditions[0]: Alpaca Observing Conditions Sim
        Rotator[0]: Alpaca Rotator Sim
        SafetyMonitor[0]: Alpaca SafetyMonitor Sim
        Switch[0]: Alpaca Switch V2 Sim
        Telescope[0]: Alpaca Telescope Sim
At 127.0.0.1:11111
    V[1] server
    ASCOM Remote Server
        Rotator[0]: Rotator Simulator
        Telescope[0]: Telescope Simulator for .NET
        Focuser[0]: ASCOM Simulator Focuser Driver
```

`alpaca.management.`**`apiversions`**(*addr: str*) → List[int]

Returns a list of supported Alpaca API version numbers

Parameters **addr** – An *address:port* string from discovery

Raises **AlpacaRequestException** – Method or parameter error, internal Alpaca server error

Note

- Currently (April 2022) this will be [1]

`alpaca.management.`**`configureddevices`** (*addr: str*) → List[dict]

Return a list of dictionaries describing each device served by this Alpaca Server
Each element of the returned list is a dictionary of properties of each Alpaca device served by the server at *addr*. The dictionaries consist of the following elements:

DeviceName The name of the device

DeviceType The ASCOM standard name for the type of device

DeviceNumber The index of the device among devices of the same type. See Notes.

UniqueID A "globally unique ID" identifying this device

Parameters **`addr`** – An *address:port* string from discovery

Raises **`AlpacaRequestException`** – Method or parameter error, internal Alpaca server error

`alpaca.management.`**`description`** (*addr: str*) → str

Return a description of the device as a whole (the server)

Parameters **`addr`** – An *address:port* string from discovery

Raises **`AlpacaRequestException`** – Method or parameter error, internal Alpaca server error

Note

- This is the description of the server at the given *address:port*, which may serve multiple Alpaca devices.

Frequently Asked Questions

7.1 How can I tell if my asynchronous request failed after being started?

See Status of This Document

All asynchronous (non-blocking) methods in ASCOM are paired with corresponding properties that allow you to determine if the operation (running in the background) has finished. There are two places where an async operation can fail:

1. When you call the method that starts the operation, for example `Focuser.Move`. If you get an exception here, it means the device couldn't *start* the operation, for whatever reason. Common reasons include an out-of-range request or an unconnected device.
2. Later you read the property that tells you whether the async operation has finished, for example `Focuser.IsMoving`. If you see the value change to indicate that the operation has finished, you can be *100% certain that it completed successfully*. On the other hand, if you get an exception here (usually `DriverException`), it means the device *failed to finish the operation successfully*. In this case, the device is compromsed and requires special attention.

Tip

Have a look at this article Why exceptions in async methods are "dangerous" in C#. While the article uses the C# language and acync/await to illustrate the so-called "dangers" (failing to await), the exact same principles apply here. In the example above, you really must use `Focuser.IsMoving` to determine completion. It is the 'await' in this cross-language/cross-platform environment. If you ignore `Focuser.IsMoving` and instead "double-check" the results by comparing your request with the results, you run several risks, including

1. A lost exception (an integrity bust),
2. a false completion indication if the device passes through the requested position on its way to settling to its final place, and
3. needing to decide what "close enough" means.

Plus it needlessly complicates your code. We have to design for, and require, trustworthy devices/drivers.

7.2 The Dome Interface seems complex and confusing. Help me.

[Q] **How can I tell if I'm connected to a roll-off roof or a "dumb" clamshell?**

[A] Look for `CanSetAzimuth` to be False. This means that there is no way to move the opening to the sky at all. The only functions available will be those related to opening and closing the roof or clamshell to provide access to the entire sky (or not).

[Q] **How do I control a rotating dome with a simple shutter?**

[A] If `CanSetAltitude` is False, then you have a common dome with a rotatable opening (e.g., a slit). You can `SlewToAzimuth()` to position the slit, and of course `OpenShutter()` and `CloseShutter()`.

[Q] **How can I adjust the location of the opening (slit, port, clamshell leaves) to account for the geometry and offset of the optics?**

[A] The Dome interface does not provide for this, as it requires current pointing information from the mount/telescope, as well as mount configuration and measurements. This is a composite task requiring information about two devices, and is thus out of scope for a Dome device by itself. Your application is responsible for transforming the telescope alt/az to the alt/az needed for the dome.

There are, however, a few integrated/combined telescope/mount/dome control systems (COMSOFT PC/TCS, DFM TCS, for example) which expose both `Telescope` and `Dome` interfaces. The slaving properties in the ASCOM Dome interface are provided for these types of control systems.

7.3 What is the meaning of "pointing state" in the docs for SideOfPier?

In the docs for `Telescope.SideOfPier` and `Telescope.DestinationSideOfPier()`, for historical reasons, the name `SideOfPier` does not reflect its true meaning. The name will *not* be changed (so as to preserve compatibility), but the meaning has since become clear. *All* conventional mounts (German, fork, etc) have two pointing states for a given equatorial (sky) position. Mechanical limitations often make it impossible for the mount to position the optics at given HA/Dec in one of the two pointing states, but there are places where the same point can be reached sensibly in both pointing states (e.g. near the pole and close to the meridian). In order to understand these pointing states, consider the following (thanks to TPOINT author Patrick Wallace for this info):

All conventional telescope mounts have two axes nominally at right angles. For an equatorial, the longitude axis is mechanical hour angle and the latitude axis is mechanical declination. Sky coordinates and mechanical coordinates are two completely separate arenas. This becomes rather more obvious if your mount is an altaz, but it's still true for an equatorial. Both mount axes can in principle move over a range of 360 deg. This is distinct from sky HA /Dec, where Dec is limited to a 180 deg range (+90 to -90). Apart from practical limitations, any point in the sky can be seen in two mechanical orientations. To get from one to the other the HA axis is moved 180 deg and the Dec axis is moved through the pole a distance twice the sky codeclination (90 - sky declination).

Mechanical zero HA/Dec will be one of the two ways of pointing at the intersection of the celestial equator and the local meridian. In order to support Dome slaving, where it is

important to know which side of the pier the mount is actually on, ASCOM has adopted the convention that the Normal pointing state will be the state where a German Equatorial mount is on the East side of the pier, looking West, with the counterweights below the optical assembly and that pierEast will represent this pointing state.

Move your scope to this position and consider the two mechanical encoders zeroed. The two pointing states are, then:

Normal (pierEast)	Where the mechanical Dec is in the range -90 deg to +90 deg
Beyond the pole (pierWest)	Where the mechanical Dec is in the range -180 deg to -90 deg or +90 deg to +180 deg

"Side of pier" is a *consequence* of the former definition, not something fundamental. Apart from mechanical interference, the telescope can move from one side of the pier to the other without the mechanical Dec having changed: you could track Polaris forever with the telescope moving from west of pier to east of pier or vice versa every 12h. Thus, "side of pier" is, in general, not a useful term (except perhaps in a loose, descriptive, explanatory sense). All this applies to a fork mount just as much as to a GEM, and it would be wrong to make the "beyond pole" state illegal for the former. Your mount may not be able to get there if your camera hits the fork, but it's possible on some mounts. Whether this is useful depends on whether you're in Hawaii or Finland.

To first order, the relationship between sky and mechanical HA/Dec is as follows:

Normal state

- HA_sky = HA_mech
- Dec_sky = Dec_mech

Beyond the pole

- HA_sky = HA_mech + 12h, expressed in range ± 12h
- Dec_sky = 180d - Dec_mech, expressed in range ± 90d

Astronomy software often needs to know which which pointing state the mount is in. Examples include setting guiding polarities and calculating dome opening azimuth/altitude. The meaning of the `Telescope.SideOfPier` property, then is:

pierEast	Normal pointing state
pierWest	Beyond the pole pointing state

If the mount hardware reports neither the true pointing state (or equivalent) nor the mechanical declination axis position (which varies from -180 to +180), a driver cannot calculate the pointing state, and *must not* implement SideOfPier. If the mount hardware reports only the mechanical declination axis position (-180 to +180) then a driver can calculate SideOfPier as follows:

- **pierEast** = abs(mechanical dec) <= 90 deg
- **pierWest** = abs(mechanical Dec) > 90 deg

It is allowed (though not required) that SideOfPier may be written to force the mount to flip. Doing so, however, may change the right ascension of the telescope. During flipping, Telescope.Slewing must return True.

7.3.1 Pointing State and Side of Pier - Help for Driver Developers

A further document published on the ASCOM website, Pointing State and Side of Pier (PDF), is also installed in the Developer Documentation folder by the ASCOM Developer Components installer. This further explains the pointing state concept and includes diagrams illustrating how it relates to physical side of pier for German equatorial telescopes. It also includes details of the tests performed by Conform to determine whether the driver correctly reports the pointing state as defined above.

7.4 What is DestinationSideOfPier and why would I want to use it?

The `DestinationSideOfPier` property is provided for applications to manage pier flipping during automated image sequences. Basically you provide it with an RA and Dec, and it comes back telling you the pointing state `SideOfPier` that would result from a slew-to *at the present time*. Looking at the current SideOfPier and DestinationSideOfPier tells you if the mount would flip on a slew to those coordinates. This info is based on the given RA/Dec at the given time, so is not a static function.

The mount knows where all of its settings are, how they are applied, and what their effects are. All it needs to do is tell the app the outcome of a slew to a point. Obviously if trash RA /Dec are given the mount would raise an exception for invalid coordinates.

As your image sequence progresses, at the beginning of each image you add the exposure interval to the RA (RA is a time coordinate, right?) and if you're really picky adjust by the 0.27% difference from sidereal to solar time, then call DestinationSideOfPier(RA + image, Dec). If it tells you the flip point will be reached before the end of the exposure, then you have some choices to make:

1. Will the mount track past the flip point far enough to allow the image to proceed "from here" and complete, so you could do the flip at the end while the image downloads?
2. If the mount is hard limited at the flip point then you would have to wait until the target drifts past the flip point, flip, then proceed. Not many mounts are hard limited against tracking past their flip points.

The tricky parts are

1. For #1 above, knowing whether, and how far, the mount can track past its flip point. My own experience is that most German mounts can track at least one "typical" exposure interval past their flip points. In the old days this would be 1800 seconds for grungy CCDs with bad read noise and a narrowband filter, but nowadays, especially with CMOS, even narrowband exposures are significantly shorter. Even at the celestial equator, 1800 seconds is only 7.5 degrees, and less as declination increases (by cos(dec)). Tracking 7.5 degrees or less past a flip point seems within the capability of most GEMs. Also, if you can image past the flip point, you can download the image in parallel with flipping the mount, so the penalty for flipping is the flip time minus the image download time.
2. For #2 above, how long to wait before flipping? To handle this, stop tracking for safety, then periodically call DestinationSideOfPier(RA, Dec) for your target's coordinates while the target itself drifts towards, then past, the flip point (which you don't know but who cares?). Wait until it tells you that the mount will flip Turn on tracking, slew to your

target, the mount will flip, and off you go toward the west with your image sequence.

7.5 What does MoveAxis() do and how do I use it?

This method supports control of the mount about its mechanical axes. Upon successful return, the telescope will start moving at the specified rate about the specified axis and continue *indefinitely*. This method must be called for each axis separately. The axis motions may run concurrently, each at their own rate. Set the rate for an axis to zero to restore the motion about that axis to the rate set by the `TrackingRate` property. Tracking motion (if enabled) is suspended during this mode of operation.

Notes

- The movement rate must be within the value(s) obtained from a `Rate` object in the `AxisRates()` list for the desired axis.
- The rate is a signed value with negative rates moving in the oposite direction to positive rates.
- The values specified in `AxisRates()` are absolute, unsigned values and apply to both directions, determined by the sign used in this command.
- The value of `Slewing` will be True if the mount is moving about any of its axes as a result of this method being called. This can be used to simulate a handbox by initiating motion with the MouseDown event and stopping the motion with the MouseUp event.
- When the motion is stopped by setting the rate to zero the mount will be set to the previous `TrackingRate` or to no movement, depending on the state of the `Tracking` property.
- It may be possible to implement satellite tracking by using the `MoveAxis()` method to move the scope in the required manner to track a satellite.

a

A

B

C

D

E

F

G

H

N

O

P

R

S

T

U

V

W