

ASCOM Master Interfaces

Release 1.1.0

Robert B. Denny <rdenny@dc3.com>

Oct 28, 2026

1 ASCOM Master Interfaces (Alpaca and COM) 1

2 Essential Information 3
2.1 Introduction . . . 3
2.2 Essential Information for Developers . . . 3
2.3 Data Types Used in This Document . . . 3
2.4 Status of This Document (1.1.0 as of 28-Oct-2026) . . . 4

3 Release Notes for Interfaces as of ASCOM Platform 7 5
3.1 Non-Blocking (Asynchronous) Behavior - Documentation . . . 5
3.2 Non-Blocking (Asynchronous) Enhancement - Device Connections . . . 5
3.3 Expanded Application - Camera.ReadoutMode . . . 5
3.4 Non-Blocking (Asynchronous) Enhancement - CoverCalibrator . . . 6
3.5 Non-Blocking (Asynchronous) Enhancement - Dome . . . 6
3.6 Non-Blocking (Asynchronous) Enhancement - Switch . . . 6
3.7 Non-Blocking (Asynchronous) and Other Enhancements - Telescope . . . 6
3.8 Enhancement - Aggregated Reading of Operational Properties . . . 7

4 Frequently Asked Questions 9
4.1 Asynchronous Operations in ASCOM . . . 9
4.2 What is the “read all” feature and what are its rules? . . . 10
4.3 TimeStamp Value . . . 11
4.4 MaxADU, ElectronsPerADU, and FullWellCapacity in the Camera Interface . . . 11
4.5 State Transition Diagram for `Camera.CameraState` . . . 12
4.6 The Dome Interface seems complex and confusing. Help me. . . . 13
4.7 What are the Angles that Rotator uses, and How Do They Relate? . . . 14
4.8 Synchronous Slewing in the Telescope Interface . . . 16
4.9 Managing a Telescope Mount’s Knowledge of Time and Place . . . 17
4.10 What is the meaning of “pointing state” in the docs for `SideOfPier` . . . 18
4.11 What does `MoveAxis()` do and how do I use it? . . . 20
4.12 What are the equatorial coordinate reference frames? . . . 21
4.13 What is `DestinationSideOfPier` and why would I want to use it? . . . 22
4.14 What are `RightAscensionRate` and `DeclinationRate` and how are they used? 23
4.15 What Does `PulseGuide()` Do? I’m Confused. . . . 24
4.16 The Switch Interface seems complex and confusing. Help me. . . . 25
4.17 What do we mean by the terms Mandatory, Optional and Deprecated? . . . 27

5 ASCOM Master Interface Definitions **29**
5.1 ICameraV4 Interface . . . 29
5.2 ICoverCalibratorV2 Interface . . . 81
5.3 IDomeV3 Interface . . . 99
5.4 IFilterWheelV3 Interface . . . 123
5.5 IFocuserV4 Interface . . . 136
5.6 IObservingConditionsV2 Interface . . . 154
5.7 IRate Interface . . . 175
5.8 IRotatorV4 Interface . . . 176
5.9 ISafetyMonitorV3 Interface . . . 195
5.10 IStateValue Interface . . . 206
5.11 ISwitchV3 Interface . . . 206
5.12 ITelescope V4 Interface . . . 227
5.13 Exception Classes . . . 278

Index **283**

ASCOM Master Interfaces (Alpaca and COM)

Attention!

This is a frozen document. These specifications are subject to clarification and correction, and occasionally revision. To get the most current information please go to ASCOM Master Interfaces (HTML). The latest version of this PDF version is available at ASCOM Master Interfaces (PDF).

Authors: Robert B. Denny, Peter W. G. Simpson - *ASCOM Initiative*

This document contains the official ASCOM Alpaca and COM application programming interface (API) specifications for Camera, CoverCalibrator, Dome, FilterWheel, Focuser. ObservingConditions, Rotator, SafetyMonitor, Switch, and Telescope astronomy devices. By design, these interfaces are functionally identical on classic COM ASCOM and Alpaca. This permits interoperation between apps and devices whether they use COM or Alpaca.

Important

The specifications here are written in a generic form, without dependence on any specific language or operation system. Thus datatypes here are generic in nature. For specifics, see the ASCOM documentation for COM or Alpaca, and for your language and operating system.

- Essential Information
- Release Notes for Interfaces as of ASCOM Platform 7
- ASCOM Master Interface Definitions
- Frequently Asked Questions

- ASCOM Initiative web site
- About Alpaca and ASCOM
- ASCOM Alpaca Developers Information
- ASCOM COM Developers Information
- ASCOM Driver and Application Development Support Forum



Essential Information

2.1 Introduction

This is intended to be our master generic document or the ASCOM interfaces. The interface specifications are written using language and OS independent syntax.For more information see the ASCOM Initiative web site specifically the ASCOM Implementation Neutral Interface Definitions.

2.2 Essential Information for Developers

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 a development project, you may benefit from reviewing the following design principles that are *foundational*:

- The General Principles
- Best Practices
- Asynchronous APIs
- Exceptions in ASCOM

2.3 Data Types Used in This Document

Throughout the interface definitions, parameter and other numeric values are assigned generic datatypes `integer`, `float`, `list`, and `array`. Please see ASCOM Implementation Neutral Interface Definitions for COM and Alpaca API Reference for Alpaca implementation specific detail.

In this document the scalar types `integer` and `float` are considered signed and do not imply restrictions on their permitted values. If there are value restrictions, they are either commonly understood (e.g., range of angles) or specifically mentioned in the value's documentation. The specifics of an interface in a particular language may differentiate between similar scalar type in that language such as 16-, 32-, or 64-bit signed or unsigned integers.

A `list` is a single string of comma-separated substrings. An `array` is an indexable and enumerable set of values. Array syntax and usage is language specific.

2.4 Status of This Document (1.1.0 as of 28-Oct-2026)

This is an update to the first production release of this document. Camera image array shaping and encoding has been clarified. Rotator operation was clarified with diagrams. A requirement for internal wrap management has been added as a result of new rotators without this appearing and causing confusion and limitations. Formatting and datatype corrections were made. **No other changes were made**. The clarifying FAQ articles represent the ASCOM Platform 7 specifications. A new FAQ Section 4.17 has been added. Datatype specifics have been included. Consider this a living document. Inevitably, corrections and requests for clarifications will be applied during its lifetime. Significant corrections will be logged here in the future.

Release Notes for Interfaces as of ASCOM Platform 7

This document contains the official ASCOM Alpaca and COM application programming interface (API) specifications as of the release of the ASCOM Platform 7 and the corresponding Alpaca interfaces. Platform 7 represents a significant upgrade to all of the interfaces, hence their `InterfaceVersion` properties have been increased by 1.

3.1 Non-Blocking (Asynchronous) Behavior - Documentation

Since Alpaca is a network protocol, it is important that methods be non-blocking (asynchronous). Most of the Platform 6 interface methods are either explicitly or implicitly asynchronous. In addition, these methods already have existing properties which permit detecting their successful completion.

However the Platform 6 interface documentation did not make this explicitly clear in many cases, and was simply incorrect in a few cases. This document contains formalized and clarified documentation for those methods that are already asynchronous including specifying the relevant completion properties.

3.2 Non-Blocking (Asynchronous) Enhancement - Device Connections

A fundamental problem with all of the Platform 6 device interfaces is the use of the synchronous `Connected` property to effect connection and disconnection. Since a property cannot be made asynchronous, the upgraded interfaces now have new `Connect()` and `Disconnect()` methods, which are asynchronous, and a new `Connecting` property with which clients can detect completion of these operations.

Attention!

These changes are non-breaking. However, writing to `Connected` to effect connection and disconnection is now explicitly deprecated for `clients` but must still be implemented by `drivers` to ensure backward compatibility with earlier software.

3.3 Expanded Application - Camera.ReadoutMode

The description of `ReadoutMode` now includes a generalization that it is meant to be used for any sort of mode the camera supports such as manufacturer-recommended combina-

tions of gain and offset, changes in sensor size, etc. in addition to pixel data readout modes /speeds.

3.4 Non-Blocking (Asynchronous) Enhancement - CoverCalibrator

The `CoverCalibrator V1` interface doesn't have separate properties to undertake the state and status roles. Instead, it uses multi-purpose enum state/status properties that combine these functions. Using this approach it is possible to report an error state but it is not possible to return a message indicating the nature of the issue.

To address this, poll-able boolean completion properties have been added for cover and calibrator operations that can return text error descriptions through exceptions / errors when necessary. `CalibratorChanging` was added for `CalibratorOn()` and `CalibratorOff()`, and `CoverMoving` was added for `OpenCover()` and `CloseCover()`

3.5 Non-Blocking (Asynchronous) Enhancement - Dome

`AbortSlew()` has been newly declared as asynchronous with `Slewing()` as the completion property.

3.6 Non-Blocking (Asynchronous) Enhancement - Switch

Switch state changes are currently synchronous, which limits their application to short duration activities. Two new asynchronous set methods `SetAsync()` and `SetAsyncValue()` have been added to provide greater flexibility and enable switches to control long running activities.

These are optional, and clients may discover if the device has these capabilities by testing the new `CanAsync` property. Completion detection is provided by a new `StateChangeComplete` property. Finally an in-progress switch state change may be cancelled via the new `CancelAsync()` method.

3.7 Non-Blocking (Asynchronous) and Other Enhancements - Telescope

- `FindHome()`, `Park()`, `MoveAxis()`, and `PulseGuide()` are specifically documented as asynchronous correcting earlier errors and as a clarification.
- If a mount can slew at all, it now must support both `synchronous` *and* `asynchronous` slewing. This means that both `CanSlew` and `CanSlewAsync` must have the same value, and separately, both `CanSlewAltAz` and `CanSlewAltAzAsync` must have the same value.
- Synchronous slewing has been deprecated for `clients` but must be implemented by `drivers` for backward compatibility.
- `AbortSlew()` has been newly declared as asynchronous with `Slewing` as the completion property.
- Attempts to set `RightAscensionRate` and/or `DeclinationRate` now require

raising `InvalidOperationException` if `TrackingRate` is not `driveSidereal`.

- Several methods have been specified to throw a new `ParkedException` if they are not appropriate for a parked mount.
- Equatorial slewing methods must throw `InvalidOperationException` if `Tracking` is False or if the requested slew would fail due to exceeding a hardware limit of the mount.

3.8 Enhancement - Aggregated Reading of Operational Properties

See Section 4.2

Frequently Asked Questions

In order not to inflate the documentation of interfaces with details and best practices, these articles provide information and requirements that relate to one or more interfaces.

4.1 Asynchronous Operations in ASCOM

As of this release of the ASCOM Interfaces (both COM and Alpaca), asynchronous methods are provided for all operations. Some older methods are not asynchronous, and they are now deprecated. For details on how the transition to "all asynchronous" was accomplished, and the changes that made this possible see Release Notes for Interfaces as of ASCOM Platform 7.

4.1.1 Background for Developers

An asynchronous operation uses two interface members: an initiator method to start the operation and a completion property that can be polled to monitor the operation's progress.

Fundamentally, an asynchronous method initiates an operation and returns immediately, enabling the caller (an app) to read properties, including the completion property, and possibly initiate other operations in parallel. If, during initiation, the device knows that it cannot successfully complete the requested operation, it must raise an exception instead of returning from the method call.

There are subtitles and consequences to this mode of operation. To help developers, we have provided developer information on the details of asynchronous operations in ASCOM on our main website in the document Asynchronous Programming and Exceptions. Please consider looking at this.

4.1.2 How can I tell if my asynchronous request fails?

This section is written from an *app developer's* point of view.

All asynchronous (non-blocking) methods in ASCOM are paired with corresponding completion properties that allow you to determine if the operation (running in the background) has finished. There are *two places* where a requested async operation can fail:

1. When you call the method that starts the operation, for example `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 completion property that tells you whether the async operation has finished, for example `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 (often a `DriverException`), it means the device *failed to finish the operation successfully*. In this case, the device is compromised and requires special attention.

4.1.3 What do I do if something goes wrong?

This section is written from an *device / driver developer's* point of view.

If you determine that the operation cannot be started when the initiator method is called, e.g. a supplied parameter is incorrect or the device is not connected, the method must raise an exception rather then returning to the caller.

If something goes wrong after the operation has been initiated, report this to the client by raising an exception when the client tries to read the completion property. The exception should continue to be raised on each attempt to read the completion property until it is reset at the start of the next operation that uses that completion property.

Tip

Have a look at this article Why exceptions in async methods are "dangerous" in C#. While the article uses the C# language and async/await to illustrate the so-called "dangers" (failing to await), the exact same principles apply here. For example, you really must use `Focuser.IsMoving` to determine completion of a `Focuser.Move()`. `Focuser.IsMoving` is the 'await' in this cross-language/cross-platform environment. If you ignore `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.

4.2 What is the "read all" feature and what are its rules?

Each device interface has a new `DeviceState` property which will return, in a single read, a list `StateValue` objects, each of which is a name-value pair. The list must contain all of the device's **operational** properties. Configuration information is not included because this is either set and known by the application or can be read once at the beginning of an operation.

Each device's specified **operational** properties are defined in the documentation for that device, for example, `Telescope.DeviceState`. From both the client's and the device's perspective, `DeviceState` is a "best endeavours" call. This is to ensure that the maximum amount of available data is returned by the device to the client.

Important

Applications must expect that, from time to time, some operational state values may not be present in the device response and must implement a strategy to deal with such "missing" values.

Important

If you wish to report additional values to clients, beyond those defined as operational, implement an `Action` e.g. via `Telescope.Action` and `Telescope.SupportedActions` and return your items in this way rather than adding them to the `DeviceState` response.

This is to ensure that the `DeviceState` call is as performant as possible for both client and device and is not burdened with information that unduly increases its size and transmission time.

Conform will report non-standard `StateValue` items found in the `DeviceState` response as Issues.

Note

- If a particular operational property is not available for any reason, its `StateValue` object must simply be omitted from the `DeviceState` list - do not throw a `DriverException` in this circumstance.
- If no operational states are available for the device, an empty list (with no `TimeStamp`) must be returned.
- The `DeviceState` property should only throw exceptions under the most exceptional circumstances such as losing connection to the physical device.

4.3 TimeStamp Value

Each `DeviceState` list must include a `TimeStamp` element so that the device can record the time at which the operational states were was measured, if known. The (string) ISO-8601 time format must be used to report:

- An unqualified local time.
- A local time including a time offset.
- A UTC time using the Z time-zone designator

4.4 MaxADU, ElectronsPerADU, and FullWellCapacity in the Camera Interface

The `MaxADU`, `ElectronsPerADU`, and `FullWellCapacity` are characteristic properties which describe the camera sensor and its analog-to-digital converter (ADC). The last two are essential parameters for scientific imaging.

4.4.1 Values Must Reflect Current Camera Modes

The primary application use case for these properties is (1) Establish the operating modes, (2) Acquire the image, and (3) Query the interface to determine these properties. However, these properties must be available before the first image is acquired, in other words, immediately on startup (with default values). Also, after making a change to relevant properties such as `BinX`, `BinY`, `Gain`, `Offset`, or `ReadoutMode`, or via the camera's setup dialog, the mode change(s) must be immediately reflected in these properties

4.4.2 Purpose of MaxADU

`MaxADU` is provided for imaging applications so they may know the *maximum* range of ADU values to expect from a camera in `ImageArray` and therefore may establish their display scaling, etc.

4.4.3 Meaning of MaxADU

`MaxADU` is a characteristic of the camera, not of the current image (if any). It must be the maximum ADU value that can *ever* be output by the camera sensor and its analog-to-digital converter (ADC) *in its current operating modes*. Usually this would correspond to the current bit depth as `2^bitdepth - 1`. However if, by design, a camera is limited *in its current modes* to an ADU value that is *significantly* lower than `2^bitdepth - 1`, it must report the maximum possible pixel value (the upper limit).

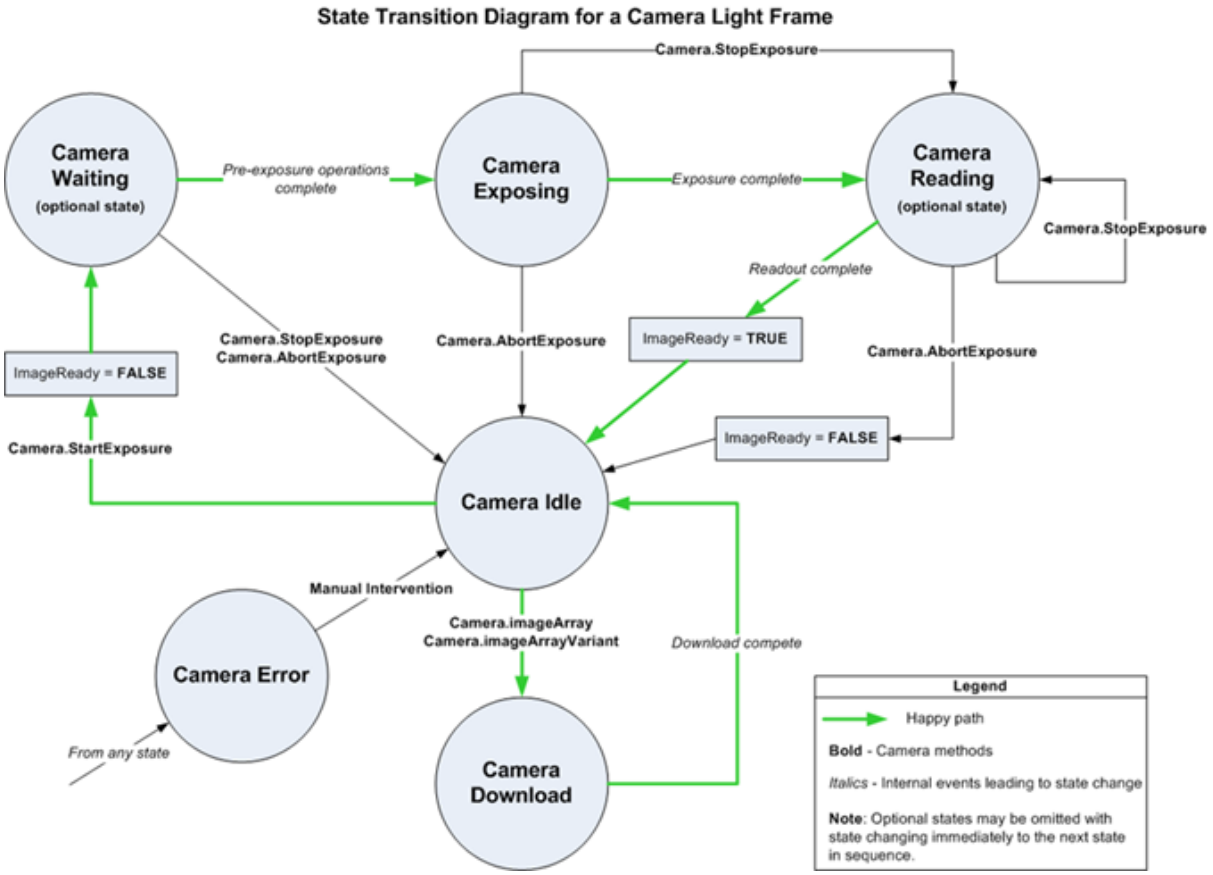
For example if the camera's ADC is 12 bit, but by design, and *in the camera's current modes*, the sensor can only ever produce a 12-bit digitized pixel value of 3800 (a significantly lower value), then `MaxADU` must report 3800, and not `2^12 - 1 = 4095`.

4.4.4 Inflation of Data Values - Upscaling to Higher Bit depth

This specification strongly discourages inflating the pixel values coming from the analog to digital converter (ADC) to create the illusion that the camera is capable of a higher bit depth than it actually is. This includes not interpolating between real ADU values to smooth the inflated data between inflated values. ADUs should increase and decrease in steps of 1. Cameras that inflate values are essentially useless for science.

4.5 State Transition Diagram for `Camera.CameraState`

This specification assumes that a camera transitions between operational states `Camera.CameraStates` as shown in the following diagram:



4.6 The Dome Interface seems complex and confusing. Help me.

The `Dome` interface is designed to handle any type of enclosure including classic domes with a slit having a single shutter, split shutter, clamshells, rotatable clamshells, rolloffs, split rolloffs, well really anything. The concept is that the Dome must provide an "aperture to the sky" and a way to open and close the aperture.

The dome controller can optionally receive both azimuth and altitude commands from the application. For example, a clamshell controller that gets both azimuth and altitude could optimize the position of the leaves, minimizing exposure to wind. This could be even better if the clamshell rotates. The Dome Interface allows a driver to advertise this by returning True for both `CanSetAzimuth` and `CanSetAltitude`.

[Q] How do I provide for a roll-off roof?

[A] Return `CanSetAzimuth` = False and `CanSetAltitude()` = False. This tells the client that there is no way to adjust 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 provide for a rotating dome with a simple shutter?

[A] Return `CanSetAltitude` = False if you have a common dome with a rotatable opening (e.g., a slit). The client can use `SlewToAzimuth()` to position the slit, and of course `OpenShutter()` and `CloseShutter()`.

[Q] What are the exact meanings of `Azimuth` and `Altitude`?

[A] The specified azimuth and altitude (*referenced to the dome center/equator*) give the position on the sky that the observer wishes to observe. It is up to the device to determine how best to locate the dome aperture in order to expose that part of the sky to the telescope.This means that the mechanical position to which the Dome moves may not correspond exactly to requested observing azimuth and altitude because the device must coordinate multiple shutters, clamshell segments or roof mechanisms to provide the required aperture on the sky.

[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.

Normally this must be done by an application which connects to a `Telescope` and a `Dome`, stores relevant geometric info about both, then calculates the dome slit azimuth (and maybe altitude) for slews, as well as periodically adjusting the dome's azimuth ("slaving") as the telescope tracks across the sky.

[Q] OK, where can I find info on the calculations?

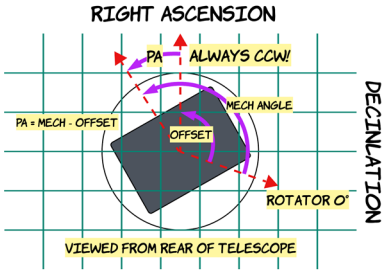
[A] The most general and easy to understand paper is Dome Calculation by Nicolas de Hilster. It includes provisions for side-by-side multiple OTAs. Included is a live workbench which you can use to test your calculations for a mount with side-by-side multiple OTAs. There is a discussion of this on the ASCOM Developers Forum.

[Q] I see that the `Dome` interface has `CanSlave` and `Slaved` properties. Why are they present if a dome controller can't slave?

[A] There are 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.

4.7 What are the Angles that Rotator uses, and How Do They Relate?

An ASCOM instrument rotator is intended to position an imager at a given angle on the sky in the *equatorial coordinate system*. Adding an instrument rotator to a telescope effectively turns it into a 3-axis system, with the imager being positioned in Right Ascension, Declination, and Position Angle. Refer to this figure:



[Q] What is Equatorial Position Angle?

[A] Looking from behind the imager, Position Angle (PA) is the angle from North (in the equatorial coordinate system) rotating in a *counterclockwise* direction. It is always positive.

[Q] What is the difference between Position and MechanicalPosition?

[A] It is difficult to mount a camera within a rotator so that the camera's position angle is exactly the same as the mechanical angle of the rotator. Therefore an ASCOM rotator keeps an angular offset from its `MechanicalPosition` to the equatorial `Position` (PA) that the imager sees at that mechanical position. Thus the device's client app can write the equatorial PA to `Position` and get the PA it wants, regardless of the mechanical angle at which the imager is mounted in the rotator or the 0-position of the rotator as configured. See the next question.

[Q] What is the purpose of Sync()?

[A] This allows the client app to tell the rotator at what PA it is currently positioned. The client app will typically do a plate solution on an image, yielding the true equatorial PA, then immediately call `Sync()` with this PA. This establishes the angular offset from its `MechanicalPosition` to its equatorial `Position` (PA).

Note

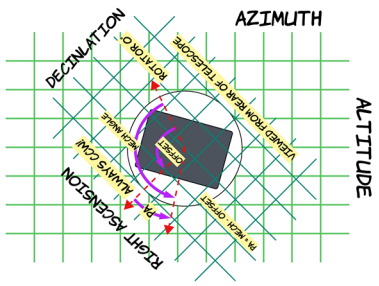
The rotator should store this offset internally, allowing *any application* to set the camera to the desired PA.

4.7.1 Rotator on an Alt-Az Mount

The above assumes that the rotator is mounted to a telescope on an equatorial mount (fork, German, etc.). When a telescope is carried on an alt-az mount [1], imaging on the sky will result in image field rotation because the optics are not aligned with the celestial sphere. Such mounts will often include a *field derotator* which will attempt to compensate for this field rotation.

This derotator will be an integrated part of the mount control system because its angle and angular rate are related to the current RightAscension and Declination by a transform from equatorial to local horizontal coordinates. As the mount "tracks" its azimuth, altitude, and field rotation angle continuously change in order to keep the RA, Dec, and PA constant (still against the celestial sphere).

It's beyond the scope of this specification to go into more detail on this. However, there are two consequences that affect the ASCOM Rotator operation in alt-az mounts. Refer to this diagram:



1 This refers to a *mount* that is physically horizontal with the ground, whose axes are vertical and horizontal. Don't confuse this with operating the mount in alt-az coordinates as opposed to equatorial coordinates. All astronomical mounts, regardless of their mechanical geometry, can be positioned with alt-az or equatorial coordinate inputs, and can display their current position in either coordinate system.

1. The derotation system must operate *below* the ASCOM Rotator. At all times `Position` must be the target *equatorial* PA, and it must remain constant against the sky while the derotator is turning continuously to provide tracking.
2. The `MechanicalPosition` property is called *field position* and it must be *relative to the optics*. This is what the client needs to do flat fields. With an equatorial mount, the optics stay at a fixed equatorial PA so the mechanical offset is a static value. With an alt-az mount, the offset between `MechanicalPosition` (field position) and `Position` (equatorial PA) is the sum of the equatorial PA offset resulting from calling `Sync()` and the continuously varying angle needed to remove the field rotation.

4.8 Synchronous Slewing in the Telescope Interface

Over the life of ASCOM, devices have provided both synchronous and asynchronous slewing. With the introduction of Alpaca, synchronous operations are badly mismatched to the communication medium, TCP/IP. Therefore in ITelescope V4 use of synchronous slewing by clients has been strongly discouraged, and thus deprecated as noted in the specifications.

It is a client author design decision as to whether, in the absence of driver asynchronous support, they will use synchronous methods or report that the driver is not compatible with their application.

In order to maximize compatibility with older ASCOM COM clients, the ITelescopeV4 and later interface specifications require COM `Telescope` drivers to support synchronous slewing via `SlewToCoordinates()`, `SlewToTarget()`, `SlewToAltAz()`, and the capability flags `CanSlew` and `CanSlewAltAz`.

Important

All ITelescopeV4 and later devices (COM and Alpaca) that can be programmatically slewed must support asynchronous slewing and must therefore report True for `CanSlewAsync` and `CanSlewAltAzAsync`.

Important

ASCOM COM drivers for mounts that can be programmatically slewed *must* support synchronous slewing to ensure backward compatibility with older clients. This is in addition to supporting asynchronous slewing as described above.

Important

Alpaca devices cannot provide reliable *synchronous* slewing operations over the network, where a method call could take several minutes to complete before returning to the client. Therefore, for mounts that can slew, Alpaca devices must always return False for `CanSlew` and `CanSlewAltAz`, and raise `MethodNotImplementedException` for the synchronous slewing methods.

4.9 Managing a Telescope Mount's Knowledge of Time and Place

In order for an astronomical mount to operate in equatorial coordinates, it must know its location on Earth (and ideally its ground elevation) and the current time. Location is easy, it doesn't change over the observing session. either the mount's hand-box can set the lat/long, or an ASCOM-based application can use `SiteLatitude`, `SiteLongitude`, and `SiteElevation` to supply this info to the mount. Time is not so easy, yet it is *critical* for pointing accuracy.

4.9.1 Simple Internal Clock

Mount designs vary in their methods of getting the current time. Most commonly, a mount will have an internal clock, which may vary in accuracy and stability, whether it remembers the time through power-down and power-up cycles, and which may be set via the mount's hand-box. In this case the mount can provide for an ASCOM-based application to update its clock to remove long term drift or just setting it after power up.

This is provided for apps by their writing to `UTCDate` to provide a time update. The usual source for a time update is a PC/Mac/Linux system ("system clock"). While most systems have reasonably stable clocks, for astronomical uses it is desirable for the *system* to have some source of time updating like Network Time Protocol, or an external GPS-based time source with an app that periodically updates the *system* clock. In this case `UTCDate` should be made both readable (required) and writeable (to provide the time updates to the mount).

Important

For mounts that depend on a host system clock, the mount designer should strongly suggest that the host's clock be synchronized with a precise time source such as NTP or GPS.

4.9.2 Internal Precision Time Source

It's also possible for a mount to have its own internal precision time source, such as a GPS receiver. The mount designer may choose to prohibit external time updates and always use the mount's GPS time source (e.g. the system clock does not have a precise source of time). In this case the mount should refuse to accept writing to `UTCDate`, and instead raise a `PropertyNotImplementedException` on writing to it. This tells the application "I don't want you to change my time, I know better." Always include a meaningful error message with the exception.

4.9.3 Internal Source of Position

A mount with a GPS receiver will always provide geodetic position (obviously), and probably ground elevation. Similar to having a precision time source, the mount designer may choose to prohibit external position updates and always use the mount's GPS site location and elevation. In this case the mount should refuse to accept writing to `SiteLatitude`, `SiteLongitude`, and `SiteElevation`, and instead raise a `PropertyNotImplementedException` on writing to them. This tells the application "I don't want you to change my location or elevation, I know better." Always include a meaningful error message with the exception.

4.9.4 Unavailability of Internal Precision Time or Location Source

At Initialization: Mount designers should consider what the mount should do in case a normally available *internal* source of precision time and location is not available or becomes unavailable after a successful initialization. If it was not available at initialization, then the mount should act as though no input of time and/or place has ever taken place, and allow both the hand-box and an application to set these parameters. Reading these uninitialized properties must raise `InvalidOperationException` (never set). Writing to these parameters should be allowed since the precision sources are not available.

After Initialization: If, on the other hand, the *internal* precision time/place becomes unavailable after initialization what to do? If the position was successfully initialized then the mount should refuse updating `SiteLatitude`, `SiteLongitude`, and `SiteElevation` since they would not change during the session. Or if the mount's internal clock would drift "too much", it may decide that, without a "recent" update from the precision time source, its performance would be degraded "too much". In this case it may choose to raise exceptions on not only reading time/position, but also on equatorial coordinate reads and writes and even slew operations. "I lost my time source so I have no idea where I am pointing or where to go". Give this scenario some thought.

4.10 What is the meaning of "pointing state" in the docs for `SideOfPier`

In the docs for `SideOfPier` and `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, please see the Context section below. Thanks to TPOINT author Patrick Wallace for this information.

4.10.1 ASCOM Convention

In order to support Dome slaving for German equatorial mounts, where it is important to know on which side of the pier the mount is physically located, ASCOM has adopted the convention that the `Normal` pointing state pertains when the mount is on the `East` side of pier, counterweights below the optical assembly, observing a target in the West at hour angle `+3.0` on the `celestial equator`.

4.10.2 Context

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 alt/az, 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 `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 declination) <= 90 deg`
- **pierWest** = `abs(mechanical declination) > 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.

4.10.3 Pointing State and Side of Pier - Help for Driver Developers

A more detailed document is available on the ASCOM website, Pointing State and Side of Pier (PDF). A copy of this document is also available in the `Start / ASCOM Platform for Developers` folder when the ASCOM Platform is installed.

The document 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.

4.11 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 (degrees/second) about the specified axis and continue *indefinitely*. This method must be called for each axis separately without affecting any other axis. 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 its previous state (tracking with or without offsets). Tracking motion (if enabled) is suspended on the specified axis only during this mode of operation. Other axes must not be affected.

This API permits the motion of the telescope about its **mechanical** axes (up to three, see `TelescopeAxes`). In addition, motion about each axis may be at a separate rate (degrees /second) via `MoveAxis()`. Furthermore, the mount may support multiple independent allowable ranges of rates about each axis via `AxisRates()`. The meaning of positive vs negative values as applies to rotation directions about the axes is purposely left undefined. App developers need to provide adaptation to various mount geometries and control systems.

4.11.1 Behavior of the `Tracking` Property

We see `MoveAxis()` as being in a different class to the higher level operations because it just says move this axis in this direction at this rate. Like the `CommandXXX` methods, the driver has no idea what objective the client is trying to achieve and must blindly follow the instructions it is given. In effect the client is now providing the high level functions of the driver / mount control system and is just commandeering the driver to communicate the desired axis movement rate to the mount.

For this reason it is our view that, when `MoveAxis()` is in effect on any axis, clients should not rely on the Tracking value reported by drivers. Since the client is providing high level control and directing use of `MoveAxis()`, only the client knows whether the mount is moving to its target coordinates at a fast rate or whether it is tracking a target at some arbitrary multi-axis rate.

Attention!

If you are looking for movements in the equatorial coordinate system, this is *not* the method for you. Guiding uses either the `PulseGuide()` method, or old fashioned "guider cables". For tracking solar system objects like comets and asteroids, the `RightAscensionRate` and `DeclinationRate` properties may be set to cause "creep" in those equatorial axes to follow the object in the sky. Typically you will have an ephemeris with right ascension and declination "creep" rates and these values may be directly used with those other properties, not `MoveAxis()`.

Note

- You must call `MoveAxis()` once to start motion about the selected axis at the selected `Rate` and once again to stop the motion and restore the previous state.
- 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 opposite direction to positive rates.
- The `Rate` values specified in `AxisRates()` are absolute, unsigned values and apply to both directions, determined by the sign used in this command.
- The meaning of positive vs negative values as applies to rotation directions about the axes is purposely left undefined. App developers need to provide adaptation to various mount geometries and control systems and their rotation directions. See Section 4.11.
- The value of `Slewing` must 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 hand-box by initiating motion with the MouseDown event and stopping the motion with the MouseUp event.
- When the motion is stopped the scope will be set to the previous `TrackingRate` or to no movement, depending on the previous 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.

4.12 What are the equatorial coordinate reference frames?

The `EquatorialSystem` property refers to the "flavor" of equatorial coordinates that the telescope (mount) uses for slewing input and current coordinate readout.

Not all equatorial coordinates are equal. For aiming a telescope (and neglecting atmospheric refraction), the Right Ascension and Declination refer to the rotational state of the Earth *at the current time*, and at the *location* of the observer on the earth. You can think of these as a simple transformation from Alt/Az to RA/Dec using the current *local* Sidereal Time, the latitude of the observer, and the current tilt of the earth's axis. The proper name for this flavor of celestial coordinates is **topocentric**. In `EquatorialCoordinateType` it is `equTopocentric`. This is often referred to as "JNow", but this is a slang term.

`equTopocentric` is the `EquatorialSystem` that the vast majority of commercial and DIY mounts use.

Catalogs of stars and deep space objects obviously cannot refer to topocentric equatorial coordinates since the time and location of the observer are unknown. Therefore, the listed coordinates are instead referred to a specific time and place, which defines the earth rotation state and other things like parallax, light deflection aberration, and space motion of the object. Most common catalogs use a reference system based on the mean pole and equinox for the standard epoch J2000.0 (The Gregorian date January 1, 2000, at 12:00 Terrestrial Time.). In `EquatorialCoordinateType` J2000 is `equJ2000`. The other coordinate types are less common.

The most precise system is the International Celestial Reference System (ICRS). For a deep dive into this, see Standards of Fundamental Astrometry (SOFA) Tools (PDF) specifically *Chapter 2, The Supported Coordinate Systems*.

Note

The ASCOM Initiative has published .NET / COM implementations of the high level methods in the SOFA library and the full Naval Observatory Vector Astrometry Software (NOVAS) library. These are both available in the ASCOM Platform Developer Components and in the cross platform ASCOM Library.

4.13 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.

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()` 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.

4.14 What are `RightAscensionRate` and `DeclinationRate` and how are they used?

These read/write properties are used most commonly to provide a mount with a way to track solar system objects such as asteroids and comets. They both apply a constant rate of change to the mount's `RightAscension` and `Declination`, respectively. This FAQ is provided to help clear up misunderstandings that have historically created problems especially for mount developers.

From a user's perspective, solar system objects have an *ephemeris* which shows the coordinates at one or more times, as well as the rate of change of the coordinates at that time. The rates described here are those coordinate rates. `DeclinationRate` should be easy to understand. The mount's Declination should change by so many (angular) arc seconds per second, depending on sign of the value.

On the other hand, there are several subtle and confusing aspects of `RightAscensionRate`. Keep in mind that RightAscension is a *time* coordinate, not an angular one. So *seconds* of RA are *not* arc-seconds. They are seconds of *time*. Furthermore, RightAscension is a sidereal time not a UTC (clock, solar) time. We'll get back to this in a bit.

If you are a mount developer, it's easy to get confused by the sidereal tracking rotation of your mount's mechanical drive and the Right Ascension of the point at which your mount is pointing. If the mount is tracking, *its Right Ascension is not changing*. That's what the mount's job is, to hold the optics at an unchanging RightAscension (and Declination). Suppose the user wants to apply a *positive* rate of change to Right Ascension. This means the mount's Right Ascension must *increase* (later time) at the requested positive `RightAscensionRate`. RightAscension increases to the East, right? What does this mean down in the mount's RA drive? It must *slow down* so the its pointing drifts toward the east in order to have its RightAscension *increase*. Perhaps counter-intuitive.

4.14.1 Units of `RightAscensionRate`

Due to an unfortunate early design choice, the units of `RightAscensionRate` are in (RA) seconds per *sidereal* second while Ephemerides all specify the rate per unit of UTC (atomic clock) time. Your driver must accept the rate in these (awkward) *RA seconds per sidereal*

second units since we never make breaking changes. To convert the given rate to (the more common) units of sidereal seconds per *UTC (atomic clock) second*, multiply the incoming value by 1.00273791 (the number of sidereal seconds in a UTC second).

4.14.2 `InvalidOperationException` When `TrackingRate` is not `driveSidereal`

These applied rates of change of equatorial coordinates don't apply when the mount is not tracking to follow the equatorial coordinate system. Both `RightAscensionRate` and `DeclinationRate` must raise an `InvalidOperationException` if `TrackingRate` is not set to `driveSidereal`.

4.15 What Does `PulseGuide()` Do? I'm Confused.

`PulseGuide()` creates small incremental *equatorial* movements and is usually used to make fine adjustments to the mount in order to keep the target location centered.

The `Direction` parameter (see `GuideDirections`) determines in which *equatorial* coordinate and direction the movement is to be made, and the `Duration` parameter specifies the length of time that the appropriate guide rate set by the mount's `GuideRateRightAscension` and `GuideRateDeclination` properties will be applied. Each call to `PulseGuide()` moves the mount in a single increment of `GuideRate * Duration = Distance`.

`PulseGuide()` always moves the scope on the equatorial coordinate axes (N/S/E/W) regardless of the mount's `AlignmentMode` (which describes the mechanical construction of the mount).

Finally, `PulseGuide()` is asynchronous. If the mount supports it, an app may call for simultaneous pulse guiding in both RA and Dec axes.

Attention!

Unfortunately some German Equatorial mounts make North and South movements in opposite directions depending on their Section 4.10 (flip state). Most commonly, the error is when the mount is "on the east" looking west ("flipped"), where the North /South directions are backwards. This is a result of the mount behaving like it's using "guider cables" which just reverse the Declination motor.

Application developers will need to implement a Declination reversal option for German mounts that have the above described behaviour. Implement this by first reading `SideOfPier` and then depending on its value, reverse the `guideNorth` versus `guideSouth` parameter in the call to `PulseGuide`.

4.15.1 Magnitude of Move

Obviously the magnitude of the movement for a `PulseGuide` call is dependent not only on the given `Duration` and `Direction` but also on the mount's guiding rates of movement as set by `GuideRateRightAscension` and `GuideRateDeclination`.

Important

The mount's guide rates are *not* related at all to `RightAscensionRate` and `DeclinationRate`, which initiate and stop secular (long running) motions in RA and Dec. The `RightAscensionRate` and `DeclinationRate` methods enable a mount to track objects that move relative to the 'fixed' star background.

4.16 The Switch Interface seems complex and confusing. Help me.

The `Switch` interface is designed to provide control of a variety of sources of power including ordinary power outlets as well as variable output power sources. There are a few confusing aspects of this interface, and this article will try to shed some light on them. The interface provides support for one or more "switches".

In the specification, an on/off outlet type, an on/off sensor type, a variable output rheostat type, and a variable value sensor type, are all referred to as a *switch*.

- An on-off outlet is controlled by `SetSwitch()` and read back by `GetSwitch()`.
- Rheostat switches are controlled by `SetSwitchValue()` and read back by `GetSwitchValue()`.
- If the `CanWrite` property of a switch is `False` then it is not a controllable switch, it is a sensor, and can only be read back via `GetSwitch()` and `GetSwitchValue()`.

Note

Despite its name, `MaxSwitch` is not the maximum value of the `Id` for a switch, it is the total number of switches supported by the device. The maximum legal value for `Id` for a switch device is `MaxSwitch - 1`. Think of `MaxSwitch` as the size of an array with indices ranging from 0 to `MaxSwitch - 1`.

4.16.1 For Application Developers

[Q] How do I turn an outlet on and off?

Given the switch number (or `Id`), call `SetSwitch()` with the `Id` and `True` or `False` for on or off respectively. You can read back the state of an outlet by calling `GetSwitch()` with the `Id`. The allowable range of `Id` values is from `0` through `MaxSwitch - 1`.

[Q] How do I control the output level of a rheostat?

Given the switch number (or `Id`), call `SetSwitchValue()` with the `Id` and the output level you want. You can read back the output level by calling `GetSwitchValue()` with the `Id`. The allowable range of output levels is given by `MinSwitchValue()` and `MaxSwitchValue()`. The allowable range of `Id` values is from `0` through `MaxSwitch - 1`.

[Q] How can I tell if a switch is an outlet type (on/off) or a rheostat?

When the difference between `MinValue` and `MaxValue` equals the value of `SwitchStep` you can assume that the switch is a binary on/off switch.

[Q] How do I tell if a switch is controllable or just a sensor?

Read the switch's `CanWrite()` value using the switch's `Id`.

[Q] I can't set the variable value I want. It keeps coming back different

Besides `MinSwitchValue()` and `MaxSwitchValue()`, the switch's output values may be restricted to successive steps given by `SwitchStep()` increments. Unfortunately, setting the switch to an unsupported value (not one of the steps) does not result in an exception, instead it will be set by rounding up to next higher legal value. See the related question for developers below.

4.16.2 For Device Developers

The behavior of your switch device can be inferred from the above section.

> **Note**
>
> A confusing aspect of this interface is that the `Value` methods must be implemented even for on/off switches, and the on/off methods must be implemented even for output value switches.

[Q] What must I do with the Value methods for my simple on/off switches?

Make the `Value` methods use a value of 0.0 to 1.0. For `MinSwitchValue()` return 0, and for `MaxSwitchValue()` return 1.0. Then make `SetSwitchValue()` take `1.0` or `0.0` for on and off respectively. `GetSwitchValue()` must return `1.0` for on and `0.0` for off. Implied by this is that `SwitchStep()` should return 1.0, just one step from 0.0 to 1.0.

[Q] What happens to a variable value switch when SetSwitchValue sets a value between steps as defined by SwitchStep?

The output value should be rounded to the nearest step. For example say `MinSwitchValue() = 5.0`, `MaxSwitchValue() = 8.0`, and `SwitchStep() = 1.0`, then

- Values in the range of 5.0 to 5.499999 => 5.0
- Values in the range of 5.5 to 6.499999 => 6.0
- Values in the range of 6.0 to 7.499999 => 7.0
- Values in the range of 7.5 to 8.0 => 8.0

[Q] What must I do with SetSwitch() for my variable value switches?

A call to `SetSwitch() True` must set the value to `MaxSwitchValue()`. A call to `SetSwitch() False` must set the value to `MinSwitchValue()`.

[Q] What must I do with GetSwitch() for my variable value switches?

A call to `GetSwitch()` must return `False` if the value is at `MinSwitchValue()`, otherwise `True`.

[Q] Conform is reporting an error for my SwitchStep() value. What could be the problem?

The specs require that `(int(MaxSwitchValue) - int(MinSwitchValue))` must be an exact multiple of `int(SwitchStep)`.

[Q] My variable output device has continuous values. What must I return for SwitchStep()?

Since `SwitchStep()` cannot be 0, return a "small" step that relates to the resolution of the A/D converter or to practical usefulness. As just mentioned, be sure to choose a `SwitchStep()` value that gives rise to an even number of steps within the range.

[Q] How can I "turn off" a variable value switch with a non-zero MinSwitchValue()?

Use a separate switch for on/off. An example might be a dew heater with 5 - 10 volts output. Make one on/off switch with `SwitchName()` of "DewOnOff", with `SwitchDescription()` of "Dew Heater Power On/Off". Then label the variable output switch "DewPower" with `SwitchDescription()` of "Dew Heater Output Power Level (low to high)".

[Q] How can I create a "momentary switch" effect?

A momentary switch activates for a short period when triggered before self-resetting, but there is no direct analogue of this behaviour in the ASCOM switch device. Momentary action can be implemented using an ASCOM switch by activating the momentary action on only one of the `SetSwitch()` operations, either `True` or `False` leaving the other to have no effect.

4.17 What do we mean by the terms Mandatory, Optional and Deprecated?

Mandatory

Interface members flagged as mandatory must be functionally implemented i.e. they must perform and return results in accordance with the interface definition.

Important

Mandatory members must always be present in the interface and must never return a not implemented error.

Optional

Implementation of interface members that are not flagged as mandatory is optional i.e. these members can return a not implemented error if the device author chooses not to implement that functionality.

Important

Even when functionality is not implemented, optional members must be present in the interface to avoid clients receiving missing member or similar errors.

Deprecated

ASCOM's definition of deprecated is: *There are other ways to implement this functionality, different members or mechanics are* **now** *preferred over using this member.*

Important

Deprecated does **not** mean "will be removed at a later date", it only implies that an alternate approach is recommended. **ASCOM will never remove a member from an interface definition because it will break clients that use that member**.



ASCOM Master Interface Definitions

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

- ICameraV4 Interface
- ICoverCalibratorV2 Interface
- IDomeV3 Interface
- IFilterWheelV3 Interface
- IFocuserV4 Interface
- IObservingConditionsV2 Interface
- IRate Interface
- IRotatorV4 Interface
- ISafetyMonitorV3 Interface
- IStateValue Interface
- ISwitchV3 Interface
- ITelescope V4 Interface
- Exception Classes



5.1 ICameraV4 Interface

class Camera

Bases: ASCOM.DeviceInterface

ASCOM Standard ICamera V4 Interface

5.1.1 Methods

Camera.**AbortExposure**( )

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

Returns Nothing

Raises
- **MethodNotImplementedException** – If CanAbortExposure is False.
- **InvalidOperationException** – If abort is not currently possible (e.g. during download).
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

May throw a MethodNotImplementedException if CanAbortExposure is false.

Note

- Unlike StopExposure() this method simply discards any partially-acquired image data and returns the camera to idle.
- Must throw an InvalidOperationException if the camera is not idle and abort is unsuccessful (or not possible, e.g. during download).
- Must throw a DriverException if a hardware or communications error occurs.
- Must *not* throw an exception if the camera is already idle.

Camera.**Action** (*ActionName: str, ActionParameters*)

Added in version 2: Recommended over (now) deprecated CommandBlind(), CommandBool(), and CommandString() as a more flexible extension mechanic.
Invoke the specified device-specific custom action

Parameters
- **ActionName** (*str*) – A name from SupportedActions that represents the action to be carried out.
- **ActionParameters** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises
- **MethodNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`Camera.`**`CommandBlind`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device without waiting for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

`Camera.`**`CommandBool`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns True/False response from the command

Return type boolean

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

`Camera.`**`CommandString`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

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

Returns String response from the command

Return type string

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

async `Camera.Connect` ( )

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

async `Camera.Disconnect` ( )

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

async `Camera.`**`PulseGuide`** (*Direction: GuideDirections*, *Duration: int*)

Activates the Camera's guiding signals to instruct the mount to move in a particular direction for a given period of time. This duplicates the function of `Telescope.PulseGuide()`. See Section 4.15.

Non-blocking: Returns with `IsPulseGuiding` False. See Notes.

Parameters
- **`Direction`** (`GuideDirections`) – The direction of the move.
- **`Duration`** (*`int`*) – duration of the guide move, milliseconds

Returns Nothing

Raises
- **`MethodNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

May throw a `MethodNotImplemented` exception if the camera does not support pulse guiding (`CanPulseGuide` property is False)

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.
- Some older cameras have implemented this as a Synchronous (blocking) operation.
- Directions are nominal and may depend on exact mount wiring. `guideNorth` must be opposite `guideSouth`, and `guideEast` must be opposite `guideWest`.
- In addition, `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 declination axis rotation depending on their pointing state. **Apps must be prepared for either behavior**.
- This duplicates the function of `Telescope.PulseGuide()` but with physical connections ("guider cables"). See Section 4.15

`Camera.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

async `Camera.`**`StartExposure`**`(` *Duration: float, Light: bool* `)`

Start an exposure. **Non-blocking**: Returns with `ImageReady` = False if exposure has *successfully* been started. Use `ImageReady` to check when the exposure is

complete and ready for access via ImageArray. See Note.

Parameters
- **Duration** (*float*) – The duration of exposure in seconds.
- **Light** (*bool*) – True for light frame, False for dark frame (ignored if no shutter)

Returns Nothing

Raises
- **InvalidValueException** – If Duration parameter, or any of BinX, BinY, StartX, StartY, NumX, and NumY have invalid or incompatible combinations of values.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

If your device throws an InvalidValueException here, it is vital that the error message explains which of the binning, subframe, or SubExposureDuration values are illegal or combine to produce an illegal combination. **Help your user solve the problem!**

Attention!

Must be implemented, must not throw a MethodNotImplementedException.

Note

- **Asynchronous** (non-blocking): Use ImageReady to determine if the exposure has been *successfully* completed and the image data is ready for access via ImageArray. Refer to ImageReady for additional info.
- The combination of values for BinX, BinY, StartX, and StartY must be checked when a call to StartExposure() is received. If any of these values is illegal, or if a limitation is imposed by CanAsymmetricBin being False. If any of these conditions exist, an appropriate exception must be thrown as described above.
- A dark frame or bias exposure may be shorter than the ExposureMin value and for a bias frame can be zero. Check the value of Light and allow exposures down to 0 seconds if Light is false. If the hardware will not support an exposure duration of zero then, for dark and bias frames, set it to the minimum that is possible.
- Some applications will set an exposure time of zero for bias frames so it's important that the driver allows this.

async `Camera.StopExposure` ( )

Stop the current exposure, if any, and make available the image data already acquired. **Non-blocking**: Returns with `ImageReady` = False if an exposure is actually in progress. Use `ImageReady` to check when the exposure has been stopped and existing image data is ready for access via `ImageArray`. See Note.

Returns Nothing

Raises
- **`MethodNotImplementedException`** – 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
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Optional, may throw a `MethodNotImplementedException`.

Note

- **Asynchronous** (non-blocking): Use `ImageReady` to determine if the exposure has been *successfully* stopped and the image data is ready for access via `ImageArray`. Refer to `ImageReady` for additional info.
- Unlike `AbortExposure()` this method must cut an exposure short while preserving the image data acquired so far, making it available to the client.
- If an exposure is in progress, the readout process is initiated. Ignored if readout is already in process.
- Must not raise an exception if the camera is idle.

5.1.2 Properties

property `Camera.BayerOffsetX`: Integer

Added in version 2: For color imaging

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

Return type Integer

Raises
- **`PropertyNotImplementedException`** – If the camera is monochrome. See first Note.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Attention!

Must be implemented by colour cameras, monochrome cameras must throw a `PropertyNotImplementedException`.

Note

- Since monochrome cameras don't have a Bayer colour matrix by definition, such cameras must throw a `PropertyNotImplementedException`. Colour cameras must always return a value and must not throw a `PropertyNotImplementedException`.
- The value returned will be in the range 0 to M-1 where M is the X-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.

property `Camera.BayerOffsetY`: Integer

Added in version 2: For color imaging

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

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If the camera is monochrome. See first Note.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented by colour cameras, monochrome cameras must throw a `PropertyNotImplementedException`.

Note

- Since monochrome cameras don't have a Bayer colour matrix by definition, such cameras must throw a `PropertyNotImplementedException`. Colour cameras must always return a value and must not throw a `PropertyNotImplementedException`.
- The value returned will be in the range 0 to M-1 where M is the Y-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.

property `Camera.BinX: Integer`

Read/Write Gets or sets the binning factor for the X axis.

Returns The binning factor for the X axis.

Return type Integer

Raises

- **`InvalidValueException`** – An invalid binning value is written to the property 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Should default to 1 when the camera connection is established.
- If `CanAsymmetricBin` is False, then the binning values must be the same. Setting this property must 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()`.

property `Camera.BinY: Integer`

Read/Write Gets or sets binning factor for the Y axis.

Returns The binning factor for the Y axis.

Return type Integer

Raises

- **`InvalidValueException`** – An invalid binning value is written to the property 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Note

- Should default to 1 when the camera connection is established.
- If `CanAsymmetricBin` is False, then the binning values must be the same. Setting this property must result in `BinX` being the same value.
- Camera does not check for compatible subframe values when this property is set; rather they are checked upon `StartExposure()`.

property `Camera.`**`CameraState`**: enum CameraStates

Current operational state of the camera. See Section 4.5.

Returns The camera's operational state

Return type `CameraStates`

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Camera.`**`CameraXSize`**: Integer

Returns The width of the camera sensor in unbinned pixels

Return type Integer

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Camera.`**`CameraYSize`**: Integer

Returns The height of the camera sensor in unbinned pixels

Return type Integer

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property Camera.CanAbortExposure: Boolean

Returns True if the camera can abort exposures with `AbortExposure()`, else False. See Note.

Return type Boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 `CanStopExposure` and "`StopExposure()`

property Camera.CanAsymmetricBin: Boolean

Returns True if the camera supports asymmetric binning, else False. See Note.

Return type Boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

If True, the camera can have different binning on the X and Y axes, as determined by `BinX` and `BinY`. If False, writing to either `BinX` or `BinY` results in the other binning value being set to the same value.

property Camera.CanFastReadout: Boolean

Added in version 2: For imaging mode support

Returns True if the camera supports a fast readout mode. If this is False then the camera may offer the preferred `ReadoutModes` mechanic.

Return type Boolean

Deprecation Notice

`FastReadout` is an obsolete mechanic for controlling camera mode(s). The `ReadoutMode` mechanic should be used.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- If the camera offers `ReadoutMode` and `ReadoutModes`, then this property must be False.
- 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.

property `Camera.`**`CanGetCoolerPower`**: **Boolean**

Returns True if the camera's cooler power level is available via `CoolerPower`, else False.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Camera.`**`CanPulseGuide`**: **Boolean**

Returns True if the camera supports pulse guiding via `PulseGuide()`, else False.

Return type Boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- Returns true if the camera can directly send auto-guider pulses to the telescope mount via an electrical connection ("guider cables").
- This does not provide any indication of whether the auto-guider cable is actually connected.
- 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.
- The action is analogous to the function of `Telescope.CanPulseGuide`. See Section 4.15.

property `Camera.`**CanSetCCDTemperature: Boolean**

Returns True if the camera's cooler temperature can be controlled via `SetCCDTemperature`, else False.

Return type Boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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.

property `Camera`.**`CanStopExposure`**: Boolean

Returns True if the camera can stop exposures via `StopExposure()` else False.

Return type Boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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()` and `CanAbortExposure`

property `Camera`.**`CCDTemperature`**: Float

Returns The current CCD cooler temperature in degrees Celsius.

Return type float

Raises
- **`InvalidOperationException`** – If data unavailable.
- **`PropertyNotImplementedException`** – If not supported (can't report cooler temperature)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Camera`.**`Connected`**: boolean

Changed in version 4: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.

Read/Write Gets or sets the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See Notes below.

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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of Camera V4. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `Camera.`**Connecting: Boolean**

Added in version 4: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.

property `Camera.CoolerOn`: **Boolean**

Read/Write Turns the camera cooler on and off and returns the current cooler on /off state.

Returns The current cooler on/off state.

Return type Boolean

Raises
- **`PropertyNotImplementedException`** – If not supported (can't report cooler temperature)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Warning

For some cameras, 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.

property `Camera.CoolerPower`: **Float**

Returns The current cooler power level in percent.

Return type Float

Raises
- **`PropertyNotImplementedException`** – If not supported (can't report cooler temperature)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

Must return zero if `CoolerOn` is False.

property `Camera.Description`: String

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `Camera.DeviceState`: List[StateValue]

Added in version 4: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

This device must return the following operational properties if they are known:

- `CameraState`
- `CCDTemperature`
- `CoolerPower`
- `HeatSinkTemperature`
- `ImageReady`
- `IsPulseGuiding`
- `PercentCompleted`
- Section 4.3

Note

- For more info see Section 4.2.

property `Camera.DriverInfo`: String

Added in version 2: Member added.

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

Return type string

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parseable string, use the `DriverVersion` property.

property `Camera.DriverVersion`: String

Added in version 2: Member added.

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

Return type string

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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.
- On systems with a comma as the decimal point you may need to make accommodations to parse the value.

property `Camera.ElectronsPerADU: Float`

Returns The gain of the camera in photoelectrons per A/D unit *in its current mode*. See Notes and Section 4.4.

Return type Float

Raises
- **`PropertyNotImplementedException`** – If this property is not available for the camera.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- At run-time, a change to `BinX`, `BinY`, `Gain`, `Offset`, or `ReadoutMode` which results in this value changing must be reflected immediately in this property's value. Similarly, if a change via the camera's `SetupDialog()` settings changes the value, this property must be updated immediately.
- See Section 4.4.

property `Camera.ExposureMax: Float`

Added in version 3: Member added.

Returns The maximum exposure time (seconds) supported by `StartExposure()`.

Return type Float

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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.

property `Camera.ExposureMin`: Float

Added in version 3: Member added.

Returns The minimum exposure time (seconds) supported by `StartExposure()`.

Return type Float

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This must be a *non-zero* number representing the shortest possible exposure time supported by the camera model.
- For bias frame acquisition an even shorter exposure may be possible; please see `StartExposure()` for more information.
- 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.

property `Camera.ExposureResolution`: Float

Added in version 3: Member added.

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

Return type Float

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

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 must choose the closest available value.
- In some cases the resolution may not be constant over the full range of exposure times; in this case the smallest increment must be chosen by the driver.
- A value of 0.0 shall indicate 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.

property `Camera.FastReadout`: **Boolean**

Added in version 2: For imaging mode support

Read/Write Enables the camera's FastReadout mode if available, and gets the current state of FastReadout mode. See Notes.

Returns True if the camera is in FastReadout mode, else False if it is in normal mode (in which the preferred `ReadoutModes` mechanic may be supported. See Notes.).

Return type Boolean

Deprecation Notice

`FastReadout` is an obsolete mechanic for controlling camera mode(s). The `ReadoutMode` mechanic should be used henceforth.

Raises

- **PropertyNotImplementedException** – If FastReadout is not supported (`CanFastReadout` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by

someone other than yourself. Includes communication errors.

Note

- Some older cameras have a single "fast mode" intended for use in focusing. When set to True, the camera will operate in Fast mode; when False, the camera will operate normally. This property, if implemented, must default to False.
- This mode mechanic is deprecated, and is included only for older cameras and drivers. The `ReadoutMode` mechanic should be used henceforth.

property `Camera.FullWellCapacity`: Float

Returns The maximum number of photoelectrons that can be held by a single pixel *in the camera's current modes*. See Notes below and Section 4.4.

Return type Float

Raises

- **`PropertyNotImplementedException`** – If this property is not available for the camera.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- At run-time, a change to `BinX`, `BinY`, `Gain`, `Offset`, or `ReadoutMode` which results in this value changing must be reflected immediately in this property's value. Similarly, if a change via the camera's `SetupDialog()` settings changes the value, this property must be updated immediately.
- See Section 4.4.
- 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.

property `Camera.Gain`: Integer

Added in version 2: Member added.

Read/Write Gets or sets the current gain value or gains index per its current gain-setting operating mode (**see description below**)

Returns The current gain value or gains index

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `Gain` is not supported at all (neither **Gains Index** nor **Gain Value** mode is supported).

- **InvalidValueException** – If the supplied 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

If the camera can support several different operating modes where gain and other parameters are different, it is better to use `ReadoutMode` and `ReadoutModes` to change all parameters in one go. If this approach is used, Gain and any other affected parameters must be kept in sync with operational values established by the readout mode.

Gain Modes Description

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 names.
  - The `Gains` property returns a *0-based* array of string gain names which should describe available gain settings e.g. "ISO 200", "ISO 1600"
  - `GainMin` and `GainMax` will throw a `PropertyNotImplementedException`.
- **Gain 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 `PropertyNotImplementedException`.

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 `PropertyNotImplementedException` then the mode is not active.

Note

- If a driver supports both modes the astronomer must be able to select the required mode through the driver Setup dialogue.
- During driver initialisation the driver must set Gain to a valid value.

property `Camera.GainMax: Integer`

Added in version 2: Member added.

Returns The maximum gain value that this camera supports in **Gain Value** mode (see Notes and `Gain`)

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `Gain` is not supported at all or if the camera is operating in the **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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

For Gain Value mode

When `Gain` is operating in Gain Value mode:

- `GainMax` must return the camera's highest valid `Gain` setting
- `GainMax` must be equal to or greater than `GainMin`
- The `Gains` property must throw `PropertyNotImplementedException`

Note

- `GainMin` and `GainMax` act together; either both must return values, or both must throw `PropertyNotImplementedException`.
- 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.

property `Camera.GainMin: Integer`

Added in version 2: Member added.

Returns The minimum gain value that this camera supports in **Gain Value** mode (see notes and `Gain`)

Return type Integer

Raises

- **PropertyNotImplementedException** – If `Gain` is not supported at all or if the camera is operating in the **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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

For Gain Value mode

When `Gain` is operating in Gain Value mode:

- `GainMin` must return the camera's lowest valid `Gain` setting
- `GainMin` must be equal to or less than `GainMax`
- The `Gains` property must throw `PropertyNotImplementedException`

Note

- `GainMin` and `GainMax` act together; either both must return values, or both must throw `PropertyNotImplementedException`.
- 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.

property `Camera.Gains`**: Array[String]**

Added in version 2: Member added.

Returns Array (0-based) of Gain *names* supported by the camera when in **Gains Index** mode (see Notes and `Gain`)

Return type Array of strings

Raises

- **PropertyNotImplementedException** – If `Gain` is not supported at all or if the camera is operating in the **Gain 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

For Gains Index mode

When `Gain` is operating in the Gains Index mode:

- The `Gains` property must return a 0-based array of available gain setting *names*.
- The `GainMax` and `GainMin` properties must throw `PropertyNotImplementedException`.

Note

- Typically the application software will display the returned gain names in a drop list, from which the astronomer can select the desired gain's name. The application can then configure the selected gain by setting the camera's `Gain` property to the *array index* of the selected gain name.
- 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.

property `Camera.HasShutter`: Boolean

Returns If True, indicates that the camera has a mechanical shutter, else False.

Return type Boolean

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`.

Note

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

property `Camera.HeatSinkTemperature`: Float

Returns The current heat sink (aka "ambient") temperature (degrees Celsius).

Return type Float

Raises

- **`PropertyNotImplementedException`** – If the camera has 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Camera`.**ImageArray: Array (multi-dimensional see below)**

Returns Array containing the exposure pixel values in ADU. See notes for array dimensionality and row vs column ordering, and `SensorType`

Return type Integer ADU values

Raises

- **InvalidOperationException** – If no image data is available (`ImageReady` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

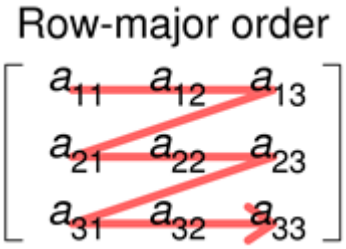
Note

- The array will have dimensions of `NumX` by `NumY` as set at the time `StartExposure()` was called. Full color images will include a "plane" for each color, with the additional plane index following the `X` and `Y` indices. See below for details.
- This is a synchronous call and clients should be prepared for it to take a long time to complete when large images are being transferred.
- Drivers written in C++ must return the image as a SafeArray.
- Developers of Alpaca camera devices are strongly advised to implement the ImageBytes mechanic, which is specified in the Alpaca API Reference, to ensure fast image transfer to the client.

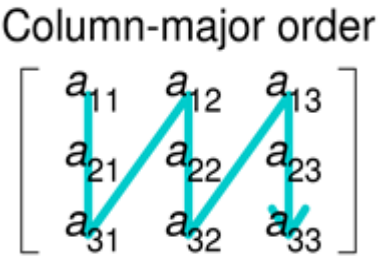
Array shaping and ordering - Monochrome and Bayer

The returned two dimensional array that supports monochrome and Bayer matrix colour sensors is specified with width as its first dimension and height as its second, rightmost, dimension. From an **infrastructure** perspective, the .NET CLR and C like languages store arrays in memory using *row major order*, which means that the rightmost array index changes most rapidly.

For an array `Array[X, Y]` it is the Y index that changes most rapidly, leading to a memory layout that looks like this:



On the other hand, the `ImageArray` property is specified to return `Array[NumX, NumY]` where `X` represents width (horizontal lines) and `Y` represents height (vertical columns). For the `ImageArray` array, the rightmost dimension is defined as the image height, hence, when stored in memory, the height index will change most rapidly. This means that, from an **application perspective**, values are held in memory in *column major* order



despite being stored in row major order from an infrastructure perspective. We consider the application view to have primacy and thus consider the returned array to be column major in structure, regardless of the form in which it is stored in memory.

Bayer Matrix Color Images

In the following definitions, `R` = red, `G` = green, `B` = blue, `C` = cyan, `M` = magenta, `Y` = yellow. The Bayer matrix is defined with `X` increasing from left to right, and `Y` increasing from top to bottom. The pattern repeats every `N` x `M` pixels for the entire pixel array, where `N` is the height of the Bayer matrix, and `M` is the width.

`SensorType` `RGGB` indicates the following matrix

	X = 0	**X = 1**
Y = 0	R	G
Y = 1	G	B

`SensorType` `CMYG` indicates the following matrix

	X = 0	**X = 1**
Y = 0	Y	C
Y = 1	G	M

`SensorType` `CMYG2` indicates the following matrix

	X = 0	X = 1
Y = 0	C	Y
Y = 1	M	G
Y = 2	C	Y
Y = 3	G	M

`SensorType` LRGB indicates the following matrix (Kodak TRUESENSE)

	X = 0	X = 1	X = 2	X = 3
Y = 0	L	R	L	G
Y = 1	R	L	G	L
Y = 2	L	G	L	B
Y = 3	G	L	B	L

The alignment of the array may be modified by `BayerOffsetX` and `BayerOffsetY`. The offset is measured from the 0,0 position in the sensor array to the upper left corner of the Bayer matrix table. **Please note that the Bayer offset values are not affected by subframe settings.**

For example, the CMYG2 sensor has a Bayer matrix offset as shown below, `BayerOffsetX` is 0 and `BayerOffsetY` is 1

	X = 0	X = 1
Y = 0	G	M
Y = 1	C	Y
Y = 2	M	G
Y = 3	C	Y

Array Shaping and Ordering - Multi-Plane Color Images (non-Bayer)

Full color images contain multiple “planes”, each of which specifies the pixel data for a single color. In `ImageArray` the third or rightmost index selects the plane. For an RGB image the planes (in order) are `R=0`, `G=1`, and `B=2`. For an LRGB image the planes are `L=0`, `R=1`, `G=2`, and `B=3`.

property `Camera.ImageArrayVariant`: Array[COM-Variant]

This property is used only in Windows ASCOM/COM drivers. Alpaca drivers and stand-alone (Alpaca) cameras must raise `PropertyNotImplementedException`.

Returns Array containing the exposure pixel values in ADU.

Return type Windows COM Variant ADU values.

Raises

- **`PropertyNotImplementedException`** – Unless the driver is for ASCOM/COM on Windows OS
- **`InvalidOperationException`** – If no image data is available (`ImageReady` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property is included here for completeness, however it is unique and useful only within Windows COM (C#, Visual Basic).

Alpaca drivers and stand-alone Alpaca cameras must raise a PropertyNotImplementedException.

For more information see the documentation for ASCOM libraries in the Windows Platform.

Note

- This is a synchronous call and clients should be prepared for it to take a long time to complete when large images are being transferred.
- Drivers written in C++ must return the image as a SafeArray.
- See the description of `Camera.ImageArray` above for details. `ImageArrayVariant` differs only in the binary form of the individual array elements (COM-Variant instead of Integer).

property `Camera.ImageReady`: Boolean

Returns True if the requested exposure has completed and `Camera.ImageArray` contains the image ready to be read via that property.

Return type Boolean

Raises

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

detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

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 retrieval via `Camera.ImageArray`.

property `Camera.InterfaceVersion`: Short

Added in version 2: Member added.

Returns ASCOM Device *interface definition* version that this device supports. Should return 4 for this interface version.

Return type Short

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For ICameraV4, this must be 4. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- Clients can detect legacy V1 drivers by trying to read this property. If the driver raises an error, it is a V1 driver. V1 did not specify this property. A driver may also return a value of 1. In other words, a raised error or a return value of 1 indicates that the driver is a V1 driver.

property `Camera.IsPulseGuiding`: Boolean

Returns Indicates that the camera is currently executing a `PulseGuide()` operation. This duplicates the function of `Telescope.IsPulseGuiding`. See Section 4.15

Return type Boolean

Raises

- **`PropertyNotImplementedException`** – If the camera does

not support pulse guiding (`CanPulseGuide` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

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.
- This duplicates the function of `Telescope.IsPulseGuiding` except using hardware "guider cables". See Section 4.15

property `Camera.`**`LastExposureDuration`**: Float

Returns The *actual* duration of the last exposure in seconds

Return type Float

Raises

- **`PropertyNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

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

property `Camera.`**`LastExposureStartTime`**: String

Changed in version 2: (Platform 6.3) Clarified that the value must be in UTC.

Returns UTC start time of the last exposure in FITS standard format ISO-8601,

UTC.

Return type String

Raises

- **`PropertyNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Reports the actual exposure **UTC** start date/time in the FITS-standard / ISO-8601 `CCYY-MM-DDThh:mm:ss[.sss…]` format.
- This must be the actual time at which the camera started to collect photons. This may differ from the time at which `StartExposure()` was called, due to shutter latency, camera timing precision, etc.

property `Camera.MaxADU`: **Integer**

Returns The maximum possible ADU value that the camera can produce *in its current mode*. See Section 4.4.

Return type Integer

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- Must be implemented; must not throw a `PropertyNotImplementedException`.
- See Section 4.4.

property `Camera.MaxBinX`: **Integer**

Returns The maximum supported X binning value of the camera *in its current mode*.

Return type Integer

Raises

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

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

If CanAsymmetricBin = False, returns the maximum allowed binning factor. If CanAsymmetricBin = True, returns the maximum allowed binning factor for the X axis.

property Camera.**MaxBinY: Integer**

Returns The maximum supported Y binning value of the camera *in its current mode*.

Return type Integer

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

If CanAsymmetricBin = False, returns the same value as MaxBinX. If CanAsymmetricBin = True, returns the maximum allowed binning factor for the Y axis.

property Camera.**Name: String**

Added in version 2: Member added.

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

Return type string

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `Camera.`**NumX: Integer**

Read/Write Set or return the current (sub)frame width. Combined with `StartX` > 0 to specify a subframe in the X dimension.

Returns The current X size of the image in binned pixels

Return type Integer

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- If binning is active, value is in binned pixels.
- Immediately after completing an exposure, `NumX` will be the X dimension of `ImageArray`.
- Must default to `CameraXSize` with `StartX` = 0 and `BinX` = 1 (full frame unbinned) on initial camera startup.

Warning

No error check is performed for incompatibility with binning and subframe settings and capabilities when this property is changed. If these values are incompatible, the driver must throw an **InvalidValueException** from a subsequent call to `StartExposure()`.

property `Camera.`**NumY: Integer**

Read/Write Set or return the current (sub)frame width. Combined with `StartY` > 0 to specify a subframe in the Y dimension.

Returns The current Y size of the image in binned pixels

Return type Integer

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- If binning is active, value is in binned pixels.
- Immediately after completing an exposure, "NumY" will be the Y dimension of `ImageArray`.
- Must default to `CameraYSize` with `StartY` = 0 and `BinY` = 1 (full frame unbinned) on initial camera startup.

Warning

No error check is performed for incompatibility with binning and subframe settings and capabilities. If these values are incompatible, the driver must throw an **InvalidValueException** from a subsequent call to `StartExposure()`.

property `Camera.Offset`**: Integer**

Added in version 3: Member added.

Read/Write Gets or sets the current offset value or offsets index per its current offset-setting operating mode (**see description below**)

Returns The current offset value or Offsets index

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `Offset` is not supported at all (neither **Offsets Index** nor **Offset Value** mode is supported.)
- **`InvalidValueException`** – If the supplied 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

If the camera can support several different operating modes where Offset and other parameters are different, it is better to use `ReadoutMode` and `ReadoutModes` to change all parameters in one go. If this approach is used, Offset and any other affected parameters must be kept in sync with operational values established by the readout mode.

Offset Modes Description

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 names.
  - The `Offsets` property returns a *0-based* array of string offset names which should describe available offset settings.
  - `OffsetMin` and `OffsetMax` will throw a `PropertyNotImplementedException`.
- **Offset Value:** The `Offset` property is a direct numeric representation of the camera's offset.
  - `OffsetMin` and `OffsetMax` properties must return integers specifying the valid range for Offset value.
  - The `Offsets` array property will throw a `PropertyNotImplementedException`.

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 `PropertyNotImplementedException` then the mode is not active.

Note

- If a driver supports both modes the astronomer must be able to select the required mode through the driver Setup dialogue.
- During driver initialisation the driver must set Offset to a valid value.

property `Camera.OffsetMax`: **Integer**

Added in version 3: Member added.

Returns The maximum offset value that this camera supports in **Offset Value** mode (see notes and `Offset`)

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `Offset` is not supported at all or if the camera is operating in the **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. Include sufficient

detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

For Offsets Value mode

When `Offset` is operating in Offsets-Value mode:

- `OffsetMax` must return the camera's highest valid `Offset` setting
- `OffsetMax` must be equal to or greater than `OffsetMin`
- The `Offsets` property must throw `PropertyNotImplementedException`

Note

- `OffsetMin` and `OffsetMax` act together and that either both must return values, or both must throw `PropertyNotImplementedException`.
- 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.

property `Camera.OffsetMin`**: Integer**

Added in version 3: Member added.

Returns Minimum offset value that this camera supports in **Offset Value** mode (see notes and `Offset`)

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `Offset` is not supported at all or if the camera is operating in the **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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

For Offset Value mode

When `Offset` is operating in Offset Value mode:

- `OffsetMin` must return the camera's lowest valid `Offset` setting
- `OffsetMin` must be equal to or less than `OffsetMax`
- The `Offsets` property must throw `PropertyNotImplementedException`

Note

- `OffsetMin` and `OffsetMax` act together and that either both must return values, or both must throw `PropertyNotImplementedException`.
- 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.

property `Camera.`**`Offsets`**: Array[String]

Added in version 3: Member added.

Returns Array (0-based) of Offset *names* supported by the camera when in **Offset Index** mode (see Notes and `Offset`)

Return type Array of strings

Raises

- **`PropertyNotImplementedException`** – If `Offset` is not supported at all or if the camera is operating in the **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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

For Offsets-Index mode

When `Offset` is operating in the Offsets Index mode:

- The `Offsets` property returns a 0-based array of available offset setting *names*.
- The `OffsetMax` and `OffsetMin` properties will throw `PropertyNotImplementedException`.

Note

- Typically the application software will display the returned offset names in a drop list, from which the astronomer can select the desired offset's name. The application can then configure the selected offset by setting the camera's `Offset` property to the *array index* of the selected offset name.
- 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.

property `Camera.PercentCompleted: Integer`

Added in version 2: Member added.

Returns The percentage completeness (0% - 100%) of the current exposure in progress

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `PercentCompleted` is not supported
- **`InvalidOperationException`** – Thrown when `CameraState` is inappropriate for reading `PercentCompleted` (see Note below)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

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 must be raised.
- Typically the application user interface will show a progress bar based on the `PercentCompleted` value.

property `Camera.PixelSizeX: Float`

Returns The physical width (microns) of the camera sensor elements.

Return type Float

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- Value must be greater than 0 microns
- 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.

property `Camera.PixelSizeY`: Float

Returns The physical height (microns) of the camera sensor elements.

Return type Float

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Value must be greater than 0 microns
- 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.

property `Camera.ReadoutMode`: Integer

Added in version 2: Member added.

ReadoutModes are provided so the camera can offer a simple way to select combinations of gain, offset, analog-to-digital converter bit depth, and other operational parameters. Often the specific combinations of these parameters is chosen by the camera maker to support a specific usage of the camera. Each combination is given a name for `ReadoutModes`, for example “High Gain”, “Low Gain”, “Fast”, etc. The index in this array is written to `ReadoutMode` to select the mode to use. The name `ReadoutMode` is historical and should be construed to mean simply “mode”. See the Notes below.

Read/Write Gets or sets the *index* of the current camera readout mode in `ReadoutModes` array.

Returns The index in `ReadoutModes` array of the currently active camera readout mode.

Return type Integer

Raises

- **`PropertyNotImplementedException`** – If `CanFastReadout` is True.
- **`InvalidValueException`** – If the supplied value is not valid (index out of range)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented if `CanFastReadout` is false, must throw a `PropertyNotImplementedException` if `CanFastReadout` is True. Please note that the `FastReadout` mechanic is now deprecated and should not be used in any new cameras.

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.
- The name `ReadoutMode` is historical, meaning a mode of reading the pixel data from the sensor. We can't change the name, but with the advent of newer CMOS cameras, multiple combinations of gain, offset, analog-to-digital converter bit depth, and other operational parameters exist. Thus this property has been generalized to cover these combinations as modes.

Important

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

property `Camera.`**`ReadoutModes`**`: Array[String]`

Added in version 2: Member added.

ReadoutModes are provided so the camera can offer a simple way to select combinations of gain, offset, analog-to-digital converter bit depth, and other operational parameters. Often the specific combinations of these parameters is chosen by the camera maker to support a specific usage of the camera. Each combination is given a name for `ReadoutModes`, for example "High Gain", "Low Gain", "Fast", etc. This array contains these names.

Returns Array (0-based) of ReadoutMode *names* supported by the camera (see notes and `ReadoutMode`)

Return type Array of Strings (0-based)

Raises
- **`PropertyNotImplementedException`** – If `CanFastReadout` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented if `CanFastReadout` is False, must throw a `PropertyNotImplementedException` if `CanFastReadout` is True.

Note

- Readout modes may be available from the camera, and if so then `CanFastReadout` must be False. The two camera mode selection schemes are mutually exclusive.
- This property provides an array of strings, each of which is the name of 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 mode names. The choice of available modes made available is entirely at the discretion of the camera.
- The default readout mode (on startup) must be `ReadoutModes[0]`. The device should reserve this for "standard" imaging operations since it is the power-up default.
- To select a mode, set `ReadoutMode` to the index of the desired mode in `ReadoutModes`. The index is zero-based.
- Applications should only read this property while a connection to the camera is actually established. Drivers often support multiple cameras with different capabilities, which are not known until the connection is made. If the available readout modes are not known because no connection has been established, this property shall throw a `NotConnectedException`.

property `Camera.SensorName`: String

Added in version 2: Member added.

Returns The name of the sensor used within the camera. **See Notes**

Return type String

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

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Mandatory, must return an empty string if the sensor's name is not known.

Note

- Returns the name (data sheet part number) of the sensor, e.g. ICX285. The format is to be exactly as shown on manufacturer data sheet, subject to the following rules:
  - All letters shall be upper-case.
  - Spaces shall not be included.
  - Any extra suffixes that define region codes, package types, temperature range, coatings, grading, colour/monochrome, etc. shall not be included.
  - For colour sensors, if a suffix differentiates different Bayer matrix encodings, it shall be included.
  - The property must 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.

property `Camera.SensorType: enum SensorType`

Added in version 2: Member added.

Returns The type of sensor within the camera

Return type `SensorType`

Raises
- **PropertyNotImplementedException** – If the sensor type is unknown or the device just doesn't support this.
- **NotConnectedException** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- See ImageArray for details on array shaping for various sensor types.
- Please note that additional type values may be defined in future updates of the standard, as new Bayer matrices may be created by sensor manufacturers in the future. If this occurs, then a new enumeration value shall be defined. The pre-existing enumeration values shall not change.
- 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.

property Camera.**SetCCDTemperature: Float**

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

Returns The camera's cooler temperature setpoint (degrees Celsius)

Return type Float

Raises

- **InvalidValueException** – If an attempt is made to set a value that is outside the camera's valid temperature setpoint range.
- **PropertyNotImplementedException** – If CanSetCCDTemperature 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Setting this property must be short-lived because it is only expected to establish the new setpoint and *must not block until the setpoint has been reached or otherwise wait for any actual temperature changes*.

Note

- This method should be short-lived because it is only expected to 'set' the new set point and must not block until the set point has been reached.
- The driver must throw an `InvalidValueException` if an attempt is made to set `SetCCDTemperatureoutside` the valid range for the camera. As an assistance to driver authors, to protect equipment and prevent harm to individuals, Conform will report an issue if it is possible to set `SetCCDTemperature` below -280C or above +100C.
- Camera hardware and/or driver should perform cooler ramping to prevent thermal shock and potential damage to the CCD array or cooler stack, however a property-write to `SetCCDTemperature` *must not wait for any temperature changes*.

property `Camera.`**`StartX`**: **Integer**

Read/Write Set or return the current X-axis start position in binned pixels (zero based).

Subframe Feature: To set the camera to use a subframe on the sensor, first set the binning, then set `StartX`, `NumX`, `StartY`, and `NumY` to establish the binned area on the sensor to use for subsequent exposures. Each time an exposure completes, `NumX` and `NumY` will describe the dimensions of `ImageArray`.

Returns The current X-axis start position on the sensor in binned pixels.

Return type Integer

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- Must be implemented, must not throw a `PropertyNotImplementedException`
- No error check is performed for incompatibility with binning values and restrictions. If these values are incompatible, you will receive an `InvalidValueException` from a subsequent call to `StartExposure()`.

Note

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

property Camera.StartY: Integer

Read/Write Set or return the current Y-axis start position in binned pixels (zero based).

Subframe Feature: To set the camera to use a subframe on the sensor, first set the binning, then set StartX, NumX, StartY, and NumY to establish the binned area on the sensor to use for subsequent exposures. Each time an exposure completes, NumX and NumY will describe the dimensions of ImageArray.

Returns The current Y-axis start position on the sensor in binned pixels.

Return type Integer

Raises
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- Must be implemented, must not throw a PropertyNotImplementedException
- No error check is performed for incompatibility with binning values and restrictions If these values are incompatible, you will receive an InvalidValueException from a subsequent call to StartExposure().

Note

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

property Camera.SubExposureDuration: Float

Added in version 3: Member added.

(**Read/Write**) Sets the camera's sub-exposure interval (seconds)

On-Board Sub-exposure Feature: This feature was added to Camera V3 to support CMOS cameras which can provide high dynamic range images via stacking of short duration images. It allows applications to avoid downloading and storing large image frames only to stack them on the client system resulting in a single image. Instead the camera may stack the short-duration images on-board and return the stacked result in ImageArray, saving both storage on the client system and transfer time of the subframes.

Returns The camera's sub-exposure interval (seconds)

Return type Float

Raises
- **PropertyNotImplementedException** – If the camera does

not support on-board stacking with user-supplied sub-exposure interval.

- **InvalidValueException** – The supplied duration is not valid (negative or zero). See notes for other conditions.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This is an optional property and can throw a `PropertyNotImplementedException`.

Note

- This property provides for a camera to divide an exposure interval (as given to `StartExposure()`) into separate sub-exposures, then stack them internally, returning the final exposure to the client. This is often required with CMOS sensors to get the best dynamic range.
- The `SubExposureDuration` must be checked against the `Duration` parameter at the time of a call to `StartExposure()`. If the value makes no sense (longer than `Duration` or possibly an odd fraction of `Duration`), then `StartExposure()` must throw `InvalidValueException`. See notes for `StartExposure()`.

property `Camera.SupportedActions`: COM: ArrayList of String elements, Alpaca: Array of String

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.
Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

5.1.3 Enumerated Constants

`Camera.CameraStates`: Integer

Current condition of the Camera. See Section 4.5. Valid values are as follows:

Symbol	Val	Description
`cameraIdle`	0	At idle state, available to start exposure
`cameraWaiting`	1	Exposure started but waiting (for shutter, trigger, filter wheel, etc.)
`cameraExposing`	2	Exposure currently in progress
`cameraReading`	3	Sensor array is being read out (digitized)
`cameraDownloading`	4	Downloading data to host
`cameraError`	5	Camera error condition serious enough to prevent further operations

`Camera.GuideDirections`: Integer

The direction in which the guide-rate motion is to be made. See Section 4.15

Symbol	Val	Description
`guideNorth`	0	North (+ declination)
`guideSouth`	1	South (- declination)
`guideEast`	2	East (+ right ascension)
`guideWest`	3	West (- right ascension)

`Camera.SensorType`: Integer

Type of sensor in the Camera.

Symbol	Val	Description
`Monochrome`	0	Single-plane monochrome
`Color`	1	Multiple-plane Color
`RGGB`	2	Single-plane Bayer matrix RGGB

CMYG	3	Single-plane Bayer matrix CMYG
CMYG2	4	Single-plane Bayer matrix CMYG2
LRGB	5	Single-plane Bayer matrix LRGB



5.2 ICoverCalibratorV2 Interface

class CoverCalibrator

Interface: ASCOM.DeviceInterface.ICoverCalibratorV2

ASCOM Standard ICoverCalibratorV2 Interface

5.2.1 Methods

CoverCalibrator.**Action** (*ActionName: str*, *ActionParameters*)

Invoke the specified device-specific custom action

Parameters

- **ActionName** (*str*) – A name from SupportedActions that represents the action to be carried out.
- **ActionParameters** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises

- **MethodNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

async `CoverCalibrator.`**`CalibratorOff`**`( )`

Turns the calibrator off if the device has calibration capability.

Non-blocking: Test `CalibratorChanging` to detect completion of the changes. See Notes.

Raises

- **`MethodNotImplementedException`** – When `CalibratorState` returns `NotPresent`. See `CalibratorStatus`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory method for a calibrator device.

Note

- **Asynchronous** (non-blocking): If the calibrator requires time to safely shut down after use, `CalibratorState` must return `NotReady` and set `CalibratorChanging` to `True`, then when the shut down is complete `CalibratorState` must change to `Off` and `CalibratorChanging` must change to `False`.
- For devices with both cover and calibrator capabilities, this method must return the `CoverState` to its status prior to calling `CalibratorOff()`.

async CoverCalibrator.**CalibratorOn** (*Brightness: int*)

Turns the calibrator on or changes its brightness, if the device has calibration capability.

Non-Blocking test CalibratorChanging to detect completion of the changes. See Notes.

Parameters **Brightness** (*integer*) – The calibrator illumination brightness to be set

Raises

- **InvalidValueException** – When BrightnessVal is outside the range 0 to MaxBrightness.
- **MethodNotImplementedException** – When CalibratorState returns NotPresent. See CalibratorStatus
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory method for a calibrator device.

Note

- **Asynchronous** (non-blocking): If the calibrator requires time to safely stabilise after use, CalibratorState must return NotReady and set CalibratorChanging to True, then when brightness change is complete CalibratorState must change to Ready and CalibratorChanging must change to False.
- When the calibrator is ready for use, CalibratorState must return Ready.
- For devices with both cover and calibrator capabilities, this method may change the CoverState if necessary.
- If an error condition arises while turning on the calibrator, CalibratorState must be set to Error rather than Unknown.

async CoverCalibrator.**CloseCover** ( )

Initiates cover closing if a cover is present.

Non-blocking Use CoverMoving to detect completion. See Notes

Raises

- **MethodNotImplementedException** – When CoverState returns NotPresent. See CoverStatus
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one

of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- **Asynchronous** (non-blocking) Upon return from `CloseCover()`, while the cover is closing, `CoverMoving` must return `True`, and `CoverState` must return `Moving`. See `CoverStatus` enum.
- When the cover is closed, `CoverMoving` must return `False`, and `CoverState` must return `Closed`, indicating *successful* completion.
- If an error condition arises while moving between states, `CoverState` must be set to `Error` rather than `Unknown`.

`CoverCalibrator.`**`CommandBlind`** (*Command: str, Raw: bool*)

Deprecated since version 2: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If `True`, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

`CoverCalibrator.`**`CommandBool`** (*Command: str, Raw: bool*)

Deprecated since version 2: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

`CoverCalibrator.`**`CommandString`** (*Command: str, Raw: bool*)

Deprecated since version 2: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

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

Returns String response from the command

Return type string

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

async `CoverCalibrator.`**`Connect`**`( )`

Added in version 2: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

async `CoverCalibrator.`**`Disconnect`**`( )`

Added in version 2: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

`CoverCalibrator.`**`HaltCover`**`( )`

Stops any cover movement that may be in progress if a cover is present and cover movement can be interrupted.

Raises

- **`MethodNotImplementedException`** – When `CoverState` returns `NotPresent`, or if cover movement cannot be interrupted. See `CoverStatus`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

This must be a short-lived method.

Note

- This must stop any cover movement *as soon as possible*, set `CoverMoving` to `False`, and set `CoverState` of `Open`, `Closed` or `Unknown` as appropriate.
- If cover movement cannot be interrupted, a `MethodNotImplementedException` must be thrown.

async `CoverCalibrator.`**`OpenCover`**`( )`

Initiates cover opening if a cover is present.

Non-blocking Use `CoverMoving` to detect completion. See Notes

Raises

- **`MethodNotImplementedException`** – When `CoverState` returns `NotPresent`. See `CoverStatus`
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by

someone other than yourself. Includes hardware or communication errors.

Note

- **Asynchronous** (non-blocking) Upon return from `OpenCover()`, while the cover is opening, `CoverMoving` must return True, and `CoverState` must return `Moving`. See `CoverStatus` enum.
- When the cover is open, `CoverMoving` must return `False`, and `CoverState` must return `Open`, indicating *successful* completion.
- If an error condition arises while moving between states, `CoverState` must be set to `Error` rather than `Unknown`.

`CoverCalibrator.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

5.2.2 Properties

property `CoverCalibrator.`**`Brightness`**: integer

Added in version 2: Member added.

Returns The current calibrator brightness in the range 0 (completely off) to `MaxBrightness` (fully on)

Return type integer

Raises

- **`PropertyNotImplementedException`** – When `CalibratorState` returns `NotPresent`. See `CalibratorStatus`
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory property that must always return a value for a calibrator device

Note

- The brightness value must be 0 when `CalibratorState` is `Off`

property `CoverCalibrator.`**`CalibratorChanging`**: boolean

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

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory property that must always return a value for a calibrator device

Note

- Use this property to determine when an (async) `CalibratorOn()` or `CalibratorOff()` has completed, at which time it must transition from `True` to `False`.
- The brightness value must be 0 when `CalibratorState` is `Off`

property `CoverCalibrator.`**`CalibratorState`**: enum CalibratorStatus

Returns The state of the calibration device, if present, otherwise returns `NotPresent`

Return type `CalibratorStatus`

Raises

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

detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory property that must always return a value for a calibrator device.

Note

- Whenever the calibrator is changing, both `CalibratorChanging` must be `True`, and `CalibratorState` must be `Changing`.
- If no calibrator is present, the state must be `NotPresent`. Must not throw a `PropertyNotImplementedException`.
- The `Brightness` value must be 0 when `CalibratorState` is `Off`.
- The `Unknown` CoverState must 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. Clients do not need to take special action if this state is returned, as they must carry on as usual, calling `CalibratorOn()` and `CalibratorOf()` 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`.
- This property is intended to be available under all but the most disastrous driver conditions. If something has gone wrong, the `CalibratorState` must be `Error` rather than throwing an exception.

property `CoverCalibrator`**.Connected: boolean**

Changed in version 2: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of CoverCalibrator V2. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `CoverCalibrator.`**`Connecting`**: **boolean**

Added in version 2: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.
- New in ICoverCalibratorV2

property `CoverCalibrator.`**`CoverMoving`**: boolean

Added in version 2: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the cover is moving. Used to determine completion of a non-blocking `OpenCover()` or `CloseCover()` operation

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This is the correct property to use when determining completion of a non-blocking `OpenCover()` or `CloseCover()` operation

property `CoverCalibrator.`**`CoverState`**: enum CoverStatus

Returns The state of the device cover.

Return type `CoverStatus`

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory property that must always return a value for a calibrator device.

Note

- Whenever the cover is opening of closing, both `CoverMoving` must be `True`, and `CoverState` must be `Moving`.
- If no cover is present, the `CoverState` must be `NotPresent`. You must not throw a `PropertyNotImplementedException`.
- `CoverState` = `Unknown` must only be returned if the device is unaware of the cover's state e.g. if the hardware does not report the open/closed state and the cover has just been powered on. Clients do not need to take special action if this state is returned, as they must carry on as usual, calling `OpenCover()` and `CloseCover()` 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`.
- This property is intended to be available under all but the most disastrous driver conditions. If something has gone wrong, the `CoverState` must be `Error` rather than throwing an exception.

property `CoverCalibrator.`**`Description`**: String

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `CoverCalibrator.`**`DeviceState`**: List[StateValue]

Added in version 2: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

This device must return the following operational properties if they are known:

- `Brightness`
- `CalibratorChanging`
- `CalibratorState`
- `CoverMoving`
- `CoverState`
- Section 4.3

Note

- For more info see Section 4.2.

property `CoverCalibrator.`**`DriverInfo`**: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long.It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself.To get the driver version in a parse-able string, use the `DriverVersion` property.

property `CoverCalibrator.`**`DriverVersion`**: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in

the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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.
- On systems with a comma as the decimal point you may need to make accommodations to parse the value.

property `CoverCalibrator.`**`InterfaceVersion`**: Integer

Returns ASCOM Device *interface definition* version that this device supports. Should return 2 for this interface version.

Return type integer

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV4, this will be 4. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.

property `CoverCalibrator.`**`MaxBrightness`**: Integer

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

Return type integer

Raises

- **`MethodNotImplementedException`** – When `CalibratorState` returns `NotPresent`.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes hardware or communication errors.

Important

This is a mandatory property *for a calibrator device* that must always return a value within the integer range 1 to 2,147,483,647 (32-bit integer)

Note

The value will always be a positive integer, indicating the available maximum value. 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”.

property `CoverCalibrator.Name: String`

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `CoverCalibrator.SupportedActions`**: COM: ArrayList of String elements, Alpaca: Array of String**

Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

5.2.3 Enumerated Constants

`CalibratorStatus`: integer

Describes the state of a calibration device.

Symbol	Val	Description
`NotPresent`	0	This device does not have a calibration capability
`Off`	1	The calibrator is off
`NotReady`	2	The calibrator is stabilising or is not yet in the commanded state
`Ready`	3	The calibrator is ready for use
`Unknown`	4	The calibrator state is unknown
`Error`	5	The calibrator encountered an error when changing state

`CoverStatus`: integer

Describes the state of a telescope cover

Symbol	Val	Description
`NotPresent`	0	This device does not have a cover that can be closed independently
`Closed`	1	The cover is closed
`Moving`	2	The cover is moving to a new position
`Open`	3	The cover is open
`Unknown`	4	The state of the cover is unknown
`Error`	5	The device encountered an error when changing state



5.3 IDomeV3 Interface

class Dome

Interface: `ASCOM.DeviceInterface.IDomeV3`

ASCOM Standard IDomeV3 Interface

> **Important**
>
> This interface has some subtleties. Please see Section 4.6

5.3.1 Methods

async `Dome.`**`AbortSlew`**`( )`

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

Non-blocking: Returns immediately with `Slewing = True` until the dome movement has been stopped, at which time `Slewing` becomes `= False`. See Notes, and Section 4.1.

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

> **Attention!**
>
> Must be implemented, must not throw a `MethodNotImplementedException`.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor stopping of the movement. When the dome has *successfully* stopped the movement, `Slewing` becomes `False`. See Section 4.1
- When motion stops, `Slewing` must become False, and slaving must have stopped as indicated 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()` must initiate stoppage of alt/az movement of the opening as well as stoppage of opening or closing.

`Dome.`**`Action`** (*ActionName: str*, *ActionParameters*)

Added in version 2: To replace deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` with a more flexible extension mechanic.
Invoke the specified device-specific custom action

Parameters
- **`ActionName`** (*str*) – A name from `SupportedActions` that represents the action to be carried out.
- **`ActionParameters`** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises
- **`MethodNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

async `Dome.CloseShutter` ( )

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.

Raises
- **`MethodNotImplementedException`** – If the dome does not have a controllable shutter/roof. In this case `CanSetShutter` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

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 = True`, it means the shutter was already closed, another success.
- If another app calls `CloseShutter()` while the shutter is already closing, the request must be accepted without error.

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.

`Dome.CommandBlind` (*Command: str, Raw: bool*)

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters
- **`Command`** (*`string`*) – The literal command string to be transmit-

ted.The literal command string to be transmitted.

- **Raw** (*bool*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises
- **MethodNotImplementedException** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Dome.`**`CommandBool`** (*Command: str, Raw: bool*)

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

Parameters
- **Command** (*string*) – The literal command string to be transmitted.
- **Raw** (*bool*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns True/False response from the command

Return type boolean

Raises

- **MethodNotImplementedException** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

`Dome.`**`CommandString`** (*Command: str, Raw: bool*)

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

Parameters

- **Command** (*string*) – The literal command string to be transmitted.
- **Raw** (*bool*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns String response from the command

Return type string

Raises

- **MethodNotImplementedException** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions.The driver is always aware of what is happening and can adapt more effectively to client needs.

async `Dome.`**`Connect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.

Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

async `Dome.`**`Disconnect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.

Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, Connecting must be True unless already disconnected. Disconnect has *successfully* completed when Connecting becomes (or is) False.
- This is a mandatory method and must not throw a MethodNotImplementedException.
- Use this to disconnect from a device rather than setting Connected to False.

async Dome.**FindHome** ( )

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

Non-blocking: Returns immediately with Slewing = True if the homing operation has *successfully* been started, or Slewing = False which means the home position has already been found (and of course AtHome must already be True). See Notes.

Raises

- **MethodNotImplementedException** – If the dome does not support homing.
- **SlavedException** – If Slaved 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- **Asynchronous** (non-blocking): Use the Slewing property to monitor the operation. When the home position has been *successfully* reached, Azimuth is synchronized to the appropriate value, Slewing becomes False and AtHome becomes True.
- Do not use AtHome to indicate completion, as it *may* become True during homing operations even if homing has not completed.
- Calling FindHome if the Dome is at home must be harmless.

async Dome.**OpenShutter** ( )

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.

Raises

- **MethodNotImplementedException** – If the dome does not have a controllable shutter/roof. In this case `CanSetShutter` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

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.
- If another app calls `OpenShutter()` while the shutter is already opening, the request must be accepted without error.

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.

`Dome.`**`Park`**`( )`

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` must already be True). See Notes.

Raises

- **MethodNotImplementedException** – If the dome does not support parking. In this case `CanPark` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When the park position has been *successfully* reached, `Slewing` becomes False and `AtPark` becomes True.
- Do not use `AtPark` to indicate completion, as it *may* become True during parking operations even if parking has not completed.
- An app should check `AtPark` before calling `Park()`.

`Dome.`**`SetPark`**`( )`

Set current azimuth position of dome to be the park position

Raises

- **`MethodNotImplementedException`** – If the dome does not support the setting of the park position. In this case `CanSetPark` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

`Dome.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

`Dome.`**`SlewToAltitude`**`(` *Altitude: float* `)`

Start slewing so that requested viewing altitude (degrees) is available for observ-

ing. See Section 4.6.

Non-blocking: Returns immediately with `Slewing` = True if the slewing operation has *successfully* been started. See Notes.

Parameters **`Altitude`** (*`float`*) – The desired viewing altitude (degrees, horizon zero and increasing positive to 90 degrees at the zenith)

Raises

- **`MethodNotImplementedException`** – If the dome opening does not support vertical (altitude) control. In this case `CanSetAltitude` must be False.
- **`InvalidValueException`** – If the supplied `Altitude` is out of range
- **`SlavedException`** – If `Slaved` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When 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
- The specified altitude (*referenced to the dome center/equator*) is of the position of the opening.
- The specified altitude should be interpreted by the device as the position on the sky that the observer wishes to observe. The device has detailed knowledge of the physical structure and must coordinate shutters, roofs or clamshell segments to open an aperture on the sky that satisfies the observer's request.

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` must be used to position the opening to the sky.

Changes

- The Attention block above
- Added SlavedException

`Dome.SlewToAzimuth` (*Azimuth: float*)

Start slewing so that requested viewing azimuth (degrees) is available for observing. See Section 4.6.

Non-blocking: Returns immediately with `Slewing` = True if the slewing operation has *successfully* been started. See Notes.

Parameters **`Azimuth`** (*`float`*) – Desired viewing azimuth (degrees, North zero and increasing clockwise, i.e. 90 East, 180 South, 270 West)

Raises

- **`MethodNotImplementedException`** – If the shutter does not support azimuth synchronization. In this case `CanSyncAzimuth` must be False.
- **`InvalidValueException`** – If the supplied `Azimuth` is out of range
- **`SlavedException`** – If `Slaved` 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When 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
- 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.
- The specified azimuth should be interpreted by the device as the position on the sky that the observer wishes to observe. The device has detailed knowledge of the physical structure and must coordinate shutters, roofs or clamshell segments to open an aperture on the sky that satisfies the observer's request.

Attention!

If the shutter is closed, this method must 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` must be used to re-position the opening to the sky. This may extend the time needed to complete the `OpenShutter()` operation.

Changes

- The Attention block above
- Added SlavedException

Dome.**SyncToAzimuth** (*Azimuth: float*)

Synchronize the current azimuth of the dome (degrees) to the given azimuth. See Section 4.6

Parameters **Azimuth** (*float*) – Target azimuth (degrees, North zero and increasing clockwise, i.e., 90 East, 180 South, 270 West)

Raises

- **MethodNotImplementedException** – If the shutter does not support azimuth synchronization. In this case CanSyncAzimuth must be False.
- **InvalidValueException** – If the supplied Azimuth is out of range
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Changes

- Added SlavedException

5.3.2 Properties

property Dome.**Altitude: float**

Returns The altitude (degrees, horizon zero and increasing positive to 90 zenith) of the part of the sky that the observer wishes to observe. See Section 4.6

Return type float

Raises

- **PropertyNotImplementedException** – If the dome does not support vertical (altitude) control / placement of its observing opening (such as a roll-off roof). In this case CanSetAltitude must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- The specified altitude (*referenced to the dome center/equator*) is of the opening to the sky through which the optics receive light, see Section 4.6.
- See `SlewToAltitude()` for operational details.
- 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.

property `Dome`**.AtHome: bool**

Returns The dome is in its home position. See `FindHome()` and Section 4.6

Return type bool

Raises

- **`PropertyNotImplementedException`** – If the dome does not support finding home. In this case `CanFindHome` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- This should not be used to determine completion of an (async) `FindHome()` operation. Use `Slewing()` for this. The `AtHome` value is reset with any azimuth slew operation that moves the dome away from the home position. `AtHome` *may* also become True during 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.
- The home position is normally defined by a hardware sensor positioned around the dome circumference and represents a fixed, known azimuth reference.
- Applications should not rely on the reported azimuth position being identical each time `AtHome` becomes True. For some devices, the home position may encompass a small range of azimuth values, rather than a discrete value, since dome inertia, the resolution of the home position sensor and/or the azimuth encoder may be insufficient to return the exact same azimuth value on each occasion. On the other hand some dome controllers always force the azimuth reading to a fixed value whenever the home position sensor is active.

property `Dome`**.AtPark: bool**

Returns True if the dome is in the programmed park position. See Section 4.6

Return type bool

Raises

- **`PropertyNotImplementedException`** – If the dome does not support parking. In this case:attr:*~Dome.CanPark* must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- This should not be used to determine completion of an (async) `Park()` operation. Use `Slewing()` for this. The `AtPark` value is reset with any azimuth slew operation that moves the dome away from the park position. `AtPark` *may* also become True during normal slew operations, if the dome passes through the park 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 park position.
- Set following a `Park()` operation and reset with any slew operation.
- Applications should not rely on the reported azimuth position being identical each time `AtPark` becomes true. For some devices, the park position may encompass a small range of azimuth values, rather than a discrete value, since dome inertia, the resolution of the park position sensor and/or the azimuth encoder may be insufficient to return the exact same azimuth value on each occasion. On the other hand some dome controllers always force the azimuth reading to a fixed value whenever the park position sensor is active.

property `Dome`**.Azimuth: float**

Returns Dome azimuth (degrees) of the opening to the sky

Return type float

This does not include the geometric transformations needed for mount and optics configurations. See Section 4.6

Raises

- **`PropertyNotImplementedException`** – If the dome does not support directional (azimuth) control / placement of its observing opening (such as a roll-off roof). In this case `CanSetAzimuth` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by

someone other than yourself. Includes hardware or communication errors.

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. See Section 4.6.
- You can detect a roll-off roof by seeing that `CanSetAzimuth` is False.
- See *~Dome.SlewToAzimuth* for operational details.
- 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.

property `Dome`**.CanFindHome: bool**

Returns True if the dome can find its home position via `FindHome()`

Return type bool

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome`**.CanPark: bool**

Returns True if the dome can be programmatically parked via `Park()`

Return type bool

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome`**.CanSetAltitude: bool**

Returns True if the opening's altitude can be set via `SlewToAltitude()`.

Return type bool

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome`**.CanSetAzimuth: bool**

Returns True if the opening's azimuth can be set via `SlewToAzimuth()`.

Return type bool

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome`**.CanSetPark: bool**

Returns True if dome park position can be set via `SetPark()`.

Return type bool

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome`**.CanSetShutter: bool**

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

Return type bool

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome`**.CanSlave: bool**

Returns The opening can be slaved to the telescope/optics via `Slaved`. Only for integrated control systems, see Section 4.6

Return type bool

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- If this is True, then the exposed Dome interface is part of an integrated mount/dome control system that offers automatic slaving. See Section 4.6.

property `Dome`**.CanSyncAzimuth: bool**

Returns True if the opening's azimuth position can be synchronised via

SyncToAzimuth().

Return type bool

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

property Dome.**Connected: boolean**

Changed in version 3: Writing to change connection state superseded by asynchronous Connect(), Disconnect(), and Connecting.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer Connect() and Disconnect() methods, and the newer Connecting property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of Dome V3. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking Connect() and Disconnect() methods, with the new Connecting property serving as the completion property.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `Dome`**.Connecting: Boolean**

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.

property `Dome`**.Description: String**

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `Dome`**.DeviceState: List[**StateValue**]**

Added in version 3: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

This device must return the following operational properties if they are known:

- `Altitude`
- `AtHome`
- `AtPark`
- `Azimuth`
- `ShutterStatus`
- `Slewing`
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the Dome Interface Version 3 and later.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

property `Dome.DriverInfo`: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parse-able string, use the `DriverVersion` property.

property `Dome.DriverVersion`: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property Dome.**InterfaceVersion: Short**

Returns ASCOM Device *interface definition* version that this device supports. Should return 3 for this interface version.

Return type short

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For IDomeV3, this must be 3. It should not to be confused with the DriverVersion property, which is the major.minor version of the driver for this device.

property Dome.**Name: String**

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

Return type string

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

The Description property is used to return info about the *device* rather than the *driver*.

property Dome.**ShutterStatus: enum ShutterState**

Returns Status of the dome shutter or roll-off roof: ShutterState. See Section 4.6

Return type ShutterState

Raises

- **PropertyNotImplementedException** – If the dome does not have a controllable shutter/roof. In this case CanSetShutter must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

> **Note**
>
> This property is the correct way to monitor an in-progress shutter movement. It must be `ShutterOpening` immediately after returning from an `OpenShutter()` call, and `ShutterClosing` immediately after returning from a `CloseShutter()` call.

property `Dome`**.Slaved: bool**

(Read/Write) Indicate or set whether the dome is slaved to the telescope. Only for integrated telescope/dome systems. See Section 4.6

Returns True if the dome is slaved to the telescope/optics

Return type bool

Raises

- **PropertyNotImplementedException** – If the dome controller is not part of an integrated dome/telescope control system which offers controllable dome slaving. In this case `CanSlave` must 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

property `Dome`**.Slewing: bool**

Returns True if any part of the dome is moving, opening, or closing. See Notes and Section 4.6

Return type bool

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

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

property `Dome.SupportedActions`: COM: ArrayList of String elements, Alpaca: Array of String

Returns the list of custom action names supported by this driver, to be used with `Action()`.

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

5.3.3 Enumerated Constants

`ShutterState`: integer

Indicates the current state of the shutter or roof. Valid values are as follows:

Symbol	Val	Description
shutterOpen	0	The shutter or roof is open
shutterClosed	1	The shutter or roof is closed
shutterOpening	2	The shutter or roof is opening
shutterClosing	3	The shutter or roof is closing
shutterError	4	The shutter or roof has encountered a problem



5.4 IFilterWheelV3 Interface

class FilterWheel

Bases: ASCOM.DeviceInterface

ASCOM Standard IFilterWheel V3 Interface

5.4.1 Methods

FilterWheel.**Action** (*ActionName: str, ActionParameters*)

Added in version 2: Recommended over (now) deprecated CommandBlind(), CommandBool(), and CommandString() as a more flexible extension mechanic.

Invoke the specified device-specific custom action

Parameters

- **ActionName** (*str*) – A name from SupportedActions that represents the action to be carried out.
- **ActionParameters** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises

- **MethodNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`FilterWheel.`**`CommandBlind`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`FilterWheel.`**`CommandBool`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`FilterWheel.`**`CommandString`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns String response from the command

Return type string

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `FilterWheel.`**`Connect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

async `FilterWheel.`**`Disconnect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

`FilterWheel.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

5.4.2 Properties

property `FilterWheel.`**`Connected`**: boolean

Changed in version 3: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.

(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.

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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of FilterWheel V3. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `FilterWheel.Connecting`: **Boolean**

Added in version 3: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.
- New in IFilterWheelV3

property `FilterWheel.`**`Description`**: String

Added in version 2: Member added

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `FilterWheel.`**`DeviceState`**: List[StateValue]

Added in version 3: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

This device must return the following operational properties if they are known:

- `Position`
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the FilterWheel Interface Version 3 and later.

property `FilterWheel.`**`DriverInfo`**: String

Added in version 2: Member added

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parseable string, use the `DriverVersion` property.

property `FilterWheel.`**`DriverVersion`**: String

Added in version 2: Member added

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property `FilterWheel.`**`FocusOffsets`**: Array of integers

Focus offset of each filter in the wheel

Returns Integer focus offset values

Return type Array[integer]

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- For each valid slot number (from 0 to N-1), reports the focus offset for the given filter position. These values are focuser and filter dependent, and would usually be set up by the user via the SetupDialog.
- At least one filter must have an offset of zero so it may be used as the the reference for the offsets of the others.
- The number of slots N can be determined from the length of the array.
- If focuser offsets are not available, then `FocusOffsets` should report zero for all filters.

property `FilterWheel.`**`InterfaceVersion`**: Short

Added in version 2: Member added

Returns ASCOM Device *interface definition* version that this device supports. Should return 3 for this interface version.

Return type short

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by

someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For IFilterWheelV3, this must be 3. It should not be confused with the `DriverVersion` property, which is the major.minor version of the driver for this FilterWheel.
- Clients can detect legacy V1 drivers by trying to read this property. If the driver raises an error, it is a V1 driver. V1 did not specify this property. A driver may also return a value of 1. In other words, a raised error or a return value of 1 indicates that the driver is a V1 driver.

property `FilterWheel.Name`: String

Added in version 2: Member added

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `FilterWheel.Names`: Array of strings

Returns Array of the names of each filter in the wheel

Return type Array[string]

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- For each valid slot number (from 0 to N-1), reports the name of the given filter.
- The number of slots N can be determined from the length of the array.
- If names are not available, the list should contain "Filter 1", "Filter 2", ... "Filter N".

property `FilterWheel.`**`Position`**: **integer**

(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.

Returns Current filter wheel position (zero-based) if the wheel is stationary, else -1 if the wheel is moving.

Return type integer

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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- Returning a position of -1 is mandatory while the filter wheel is in motion; *valid slot numbers must not be reported back while the filter wheel is rotating past filter positions.*
- Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- **Asynchronous** (non-blocking): Writing to `Position` returns as soon as the filter change operation has been *successfully* started. `Position` must return -1 while the change is in progress. After the requested position has been *successfully* reached and motion stops, `Position` must return the requested new filter number.
- Write a position number between 0 and N-1, where N is the number of filter slots (see `Names`). Starts filter wheel rotation immediately when written. Reading the property must return current slot number (if wheel stationary) or -1 if wheel is moving.
- **Exception** Some filter wheels are built into the camera (one driver, two interfaces). Some cameras may not actually rotate the wheel until the exposure is triggered. In this case, the written value must be available immediately as the read value, and -1 is never returned.

property `FilterWheel.SupportedActions`: COM: ArrayList of String elements, Alpaca: Array of String

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.

Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.



5.5 IFocuserV4 Interface

class Focuser

Bases: `ASCOM.DeviceInterface`

ASCOM Standard IFocuser V4 Interface

5.5.1 Methods

`Focuser.`**`Action`** (*ActionName: str, ActionParameters*)

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.

Invoke the specified device-specific custom action

Parameters

- **`ActionName`** (*str*) – A name from `SupportedActions` that represents the action to be carried out.
- **`ActionParameters`** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises

- **`MethodNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`Focuser.`**`CommandBlind`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Focuser.`**`CommandBool`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Focuser.`**`CommandString`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

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

Returns String response from the command

Return type string

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `Focuser.`**`Connect`**`( )`

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Link` to True.

async `Focuser.`**`Disconnect`**`( )`

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

`Focuser.`**`Halt`**`( )`

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

Raises

- **`MethodNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- This method must be short-lived; it is defined as synchronous in this specification.
- Some focusers may not support this function, in which case a `MethodNotImplementedException` exception must be raised.
- **Recommendation:** Host software should call this method upon initialization and, if it fails, disable the Halt button in the user interface.

async `Focuser.`**`Move`**`(` *Position: int* `)`

Changed in version 3: Exception conditions - See Historical Notes below

Starts moving the focuser by the specified amount or to the specified position depending on the value of the `Absolute` property.

Non-blocking: Returns immediately after *successfully* starting the focus change with `IsMoving` = True. See Notes.

See Notes for details on absolute versus relative focusers

Parameters **`Position`** (*int*) – Step distance or absolute position, depending on the value of the `Absolute` property. See notes.

Raises

- **`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). See the historical info below.

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

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.
- If the `Absolute` property is True, then this is an absolute positioning focuser. The `Move()` method must cause the focuser to move to an exact step position, and the Position parameter of the `Move()` method must be an integer between 0 and `MaxStep`.
- If the `Absolute` property is False, then this is a relative positioning focuser. The `Move()` method must cause 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`.

Recommendation

There is no common agreement among optical engineers as to the meaning of positive versus negative motion of a focuser. It is recommended, therefore, that a focuser have reversal setting that may be used to accommodate applications that are hard-wired to assume one or the other direction. In addition, it is recommended that auto-focus applications provide a reversal option as well.

Historical Notes

Prior to Platform 6.4, the interface specification mandated that drivers must throw an `InvalidOperationException` if a move was attempted when TempComp was True, even if the focuser was able to execute the move safely without disrupting temperature compensation.

Following discussion on ASCOM-Talk in January 2018, the Focuser interface specification has been revised to IFocuserV3, removing the requirement to throw an `InvalidOperationException`. IFocuserV3 compliant drivers are expected to execute Move requests when temperature compensation is active and to hide any specific actions required by the hardware from the client. For example this could be achieved by disabling temperature compensation, moving the focuser and re-enabling temperature compensation or simply by moving the focuser with compensation enabled if the hardware supports this.

Conform will continue to pass **IFocuserV2** drivers that throw `InvalidOperationException`. However, Conform will now fail **IFocuserV3** and later drivers that throw `InvalidOperationException`, in line with this revised specification.

`Focuser.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

5.5.2 Properties

property `Focuser.`**`Absolute`**: **boolean**

Changed in version 4: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.

Returns The focuser does absolute positioning. See the details in the docu-

mentation for the `Move()` method.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

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. See the details in the documentation for the `Move()` method.

property `Focuser.`**`Connected`**: **boolean**

Added in version 2: Supersedes `Link()`. See Historical Note below

Changed in version 4: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

Property-write deprecated as of FocuserV4. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer `Connect()` and `Disconnect()` methods.

Attention!

Must be implemented

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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 must not cause an error.

Historical Note

In previous versions of this specification the property `Link` was used to both set and retrieve the connected state, similar to the Connected property in other devices. This `Connected` property is not implemented in Version 1 drivers; these use the `Link` property and will raise a `PropertyNotImplementedException` exception for this property. Version 2 drivers must implement both `Connected` and `Link`. Applications should check that InterfaceVersion returns 2 or more before using Connected.

property `Focuser`**.Connecting: Boolean**

Added in version 4: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.

property Focuser.Description: String

Added in version 2: Member added.

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

- This describes the *device*, not the driver. See the DriverInfo property for information on the ASCOM driver.
- The description length must 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.

property Focuser.DeviceState: List[StateValue]

Added in version 4: To allow reduction of status polling

Returns List of StateValue objects representing the operational properties of this device. See Section 4.2.

Return type List[StateValue]

This device must return the following operational properties if they are known:

- IsMoving
- Position
- Temperature
- Section 4.3

Note

- For more info see Section 4.2.

property Focuser.DriverInfo: String

Added in version 2: Member added.

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

Return type string

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the Description property for information on the device itself. To get the driver version in a parse-able string, use the DriverVersion property.

property Focuser.**DriverVersion: String**

Added in version 2: Member added.

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

Return type string

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

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.
- On systems with a comma as the decimal point you may need to make accommodations to parse the value.

property Focuser.**InterfaceVersion: Short**

Added in version 2: Member added.

Returns ASCOM Device *interface definition* version that this device supports. Should return 4 for this interface version.

Return type short

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For IFocuserV4, this must be 4. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- Clients can detect legacy V1 drivers by trying to read this property. If the driver raises an error, it is a V1 driver. V1 did not specify this property. A driver may also return a value of 1.In other words, a raised error or a return value of 1 indicates that the driver is a V1 driver.

property `Focuser.IsMoving`**: boolean**

Returns The focuser is currently moving to a new position

Return type boolean

Raises

- `NotConnectedException` – If the device is not connected
- `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

Note

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

property `Focuser.Link` (*COM only*) **: boolean**

Deprecated since version 2: Deprecated long ago, may not be present in Alpaca device.

Returns True if connected to the hardware, else false

Return type boolean

Deprecation Notice

Use of this property has been deprecated since Focuser V2. To see if the device is connected to the hardware, use the `Connected` property. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

COM drivers: Link Get must be implemented and must not throw a `PropertyNotImplementedException`.

Attention!

There is no Link endpoint in the Alpaca interface. Use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property.

Note

- Must behave like `Connected`, including property-write being deprecated.

property `Focuser.`**`MaxIncrement`**: integer

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

Return type integer

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`.

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.

property `Focuser.`**`MaxStep`**: integer

Returns Maximum step position permitted.

Return type integer

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

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.

property `Focuser.`**`Name`**: String

Added in version 2: Member added.

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `Focuser.`**`Position`**: integer

Returns Current focuser position, in steps.

Return type integer

Raises

- **PropertyNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Applications must 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.

Note

- Valid only for absolute positioning focusers (see the Absolute property).

property Focuser.StepSize: float

Returns Step size (microns) for the focuser.

Return type float

Raises

- **PropertyNotImplementedException** – If the focuser 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

property Focuser.SupportedActions: COM: ArrayList of String elements, Alpaca: Array of String

Added in version 2: Recommended over (now) deprecated CommandBlind(), CommandBool(), and CommandString() as a more flexible extension mechanic.
Returns the list of custom action names supported by this driver, to be used with Action(),

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

property `Focuser`**.TempComp: boolean**

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

Returns True if temperature compensation is enabled, else False

Return type boolean

Raises

- **`PropertyNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Note

- TempComp Read must be implemented and must not throw a `PropertyNotImplementedException`.
- 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 must always return False.

Historical Note

- Prior to Platform 6.4, the interface specification mandated that drivers must throw an `InvalidOperationException` if a move was attempted when `TempComp` was `True`, even if the focuser was able to execute the move safely without disrupting temperature compensation.
- Following discussion on ASCOM-Talk in January 2018, the Focuser interface specification has been revised to `IFocuserV3`, removing the requirement to throw the `InvalidOperationException` exception. `IFocuserV3` compliant drivers are expected to execute Move requests when temperature compensation is active and to hide any specific actions required by the hardware from the client. For example this could be achieved by disabling temperature compensation, moving the focuser and re-enabling temperature compensation or simply by moving the focuser with compensation enabled if the hardware supports this.
- Conform will continue to pass `IFocuserV2` drivers that throw `InvalidOperationException` exceptions. However, Conform will now fail `IFocuserV3` drivers that throw `InvalidOperationException` exceptions, in line with this revised specification.

property `Focuser.`**`TempCompAvailable`**: **boolean**

Returns True if the focuser has temperature compensation available.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

Will be true only if the focuser's temperature compensation can be turned on and off via the `TempComp` property.

property `Focuser.`**`Temperature`**: **float**

Returns Current **ambient** temperature (degrees Celsius) as measured by the focuser.

Return type float

Raises
- **`PropertyNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Historical 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.



5.6 IObservingConditionsV2 Interface

class ObservingConditions

Bases: ASCOM.DeviceInterface

ASCOM Standard IObservingConditions V2 Interface

5.6.1 Methods

ObservingConditions.**Action** (*ActionName: str, ActionParameters*)

Invoke the specified device-specific custom action

Parameters

- **ActionName** (*str*) – A name from SupportedActions that represents the action to be carried out.
- **ActionParameters** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises

- **MethodNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`ObservingConditions.`**`CommandBlind`** (*Command: str, Raw: bool*)

Deprecated since version 2: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`ObservingConditions.`**`CommandBool`** (*Command: str, Raw: bool*)

Deprecated since version 2: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`ObservingConditions.`**`CommandString`** (*Command: str, Raw: bool*)

Deprecated since version 2: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

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

Returns String response from the command

Return type string

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `ObservingConditions.`**`Connect`**`( )`

Added in version 2: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. The connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

Note

New in ObservingConditions V2

async `ObservingConditions.`**`Disconnect`**`( )`

Added in version 2: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of

the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

Note

New in ObservingConditions V2

`ObservingConditions.`**`Refresh`**( )

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

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If refreshing 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This must be a short-lived synchronous call that triggers a refresh. It must not wait for long running processes to complete. It is the client's responsibility to poll `TimeSinceLastUpdate` to determine whether / when the data has been refreshed.

`ObservingConditions.`**`SensorDescription`**(*PropertyName: str*)

Description of the sensor providing the requested property

Parameters **`PropertyName`** (*`str`*) – The *caseless* name of the `ObservingConditions` meteorological property for which the sensor *description* is desired. For example "WindSpeed" (for `WindSpeed`) shall return a description of the sensor used to measure the wind speed.

Returns Description of the sensor used to measure the specified property.

Return type string

Raises

- **MethodNotImplementedException** – If the requested property/sensor is not implemented at all. See Attention section below for conditions.
- **InvalidValueException** – If PropertyName is not the name of one of the properties of ObservingConditions.
- **NotConnectedException** – If the device is not connected, and a connection is needed to get the descriptive name.
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must **not** throw a MethodNotImplementedException when the specified sensor is implemented but not returning data. **Must** throw a MethodNotImplementedException when the specified sensor is not implemented at all.

Note

- PropertyName must be the *caseless* name of one of the sensor properties specified in the ObservingConditions interface. If the caller supplies some other value, throw an InvalidValueException.
- If the sensor is implemented, this must return a valid string, even if the driver is not connected, so that applications can use this to determine what sensors are available.
- If the sensor is not implemented at all, this must throw a MethodNotImplementedException.

ObservingConditions.**SetupDialog**( )

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

`ObservingConditions.`**`TimeSinceLastUpdate`** (*PropertyName: str*)

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

Parameters **`PropertyName`** (*str*) – The name (in any casing) of the ObservingConditions meteorological property for which time since last update is desired. For example "WindSpeed" (for `WindSpeed`) shall return the number of seconds since wind speed was last updated.

Returns Elapsed time (seconds) since the requested property was last updated

Return type float

Raises

- **`MethodNotImplementedException`** – If the requested property/sensor is not implemented
- **`InvalidValueException`** – If `PropertyName` is not the name of one of the properties of `ObservingConditions`.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must **not** throw a `MethodNotImplementedException` when the specified sensor is implemented but **must** throw a `MethodNotImplementedException` when the specified sensor is not implemented.

Note

- `PropertyName` must be the *caseless* name of one of the sensor properties specified in the IObservingConditions interface. If the caller supplies some other value, throw an `InvalidValueException`.
- Return a negative value to indicate that no valid value has ever been received from the hardware.
- If an empty string is supplied as the `PropertyName`, the driver must return the time since the most recent update of *any sensor*. A `MethodNotImplementedException` must **not** be thrown.

5.6.2 Properties

property `ObservingConditions`**.AveragePeriod: float**

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

Returns The time period (hours) over which sensor readings will be averaged.

Return type float

Raises

- **`InvalidValueException`** – If the value set is not available for this driver. All drivers must accept 0.0 to specify that an instantaneous value 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Mandatory property, must be implemented, can **not** throw a `PropertyNotImplementedException`

Note

- AveragePeriod must return the time period (hours) over which sensor readings will be averaged. If the device is delivering instantaneous sensor readings this property must return a value of 0.0.
- Please resist the temptation to throw exceptions when clients query sensor properties when insufficient time has passed to get a true average reading. A best estimate of the average sensor value should be returned in these situations.

property `ObservingConditions.CloudCover: float`

Returns Amount of sky obscured by cloud in percent (0.0 - 100.0)

Return type float

Raises
- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Optional property, can throw a `PropertyNotImplementedException`

Note

This property should return a value between 0.0 and 100.0 where 0.0 = clear sky and 100.0 = 100% cloud coverage

property `ObservingConditions.Connected: boolean`

Changed in version 2: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of ObservingConditions V2. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `ObservingConditions.`**`Connecting`**: **Boolean**

Added in version 2: Preferred asynchronous connection mechanic. See Important section below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.
- New in IObservingConditionsV2

property `ObservingConditions.`**`Description`**: **String**

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

- This describes the *device*, not the driver. See the DriverInfo property for information on the ASCOM driver.
- The description length must 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.

property ObservingConditions.**DeviceState: List[StateValue]**

Added in version 2: To allow reduction of status polling

Returns List of StateValue objects representing the operational properties of this device. See Section 4.2.

Return type List[StateValue]

This device must return the following operational properties if they are known:

- CloudCover
- DewPoint
- Humidity
- Pressure
- RainRate
- SkyBrightness
- SkyQuality
- SkyTemperature
- StarFWHM
- Temperature
- WindDirection
- WindGust
- WindSpeed
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the ObservingConditions Interface Version 2 and later.

property `ObservingConditions.DewPoint`: **float**

Returns Atmospheric dew point temperature (degrees Celsius) at the observatory

Return type float

Raises

- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- Optional property, can throw a `PropertyNotImplementedException` when the `Humidity` property also throws a `PropertyNotImplementedException`.
- Mandatory property, must *not* throw a `PropertyNotImplementedException` when the `Humidity` property *is* implemented.
- The ASCOM specification requires that DewPoint and Humidity are either both implemented or both throw `PropertyNotImplementedException`. It is not allowed for one to be implemented and the other to throw a `PropertyNotImplementedException`.

Note

- The units of this property are degrees Celsius.

property `ObservingConditions.DriverInfo`: **String**

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parse-able string, use the `DriverVersion` property.

property `ObservingConditions`.**`DriverVersion`: String**

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property `ObservingConditions`.**`Humidity`: float**

Returns Atmospheric relative humidity (0.0 - 100.0 percent) at the observatory

Return type float

Raises

- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- Optional property, can throw a `PropertyNotImplementedException` when the `DewPoint` property also throws a `PropertyNotImplementedException`.
- Mandatory property, must *not* throw a `PropertyNotImplementedException` when the `DewPoint` property *is* implemented.
- The ASCOM specification requires that DewPoint and Humidity are either both implemented or both throw `PropertyNotImplementedException`. It is not allowed for one to be implemented and the other to throw a `PropertyNotImplementedException`.

property `ObservingConditions.`**`InterfaceVersion`**: Short

Returns ASCOM Device *interface definition* version that this device supports. Should return 2 for this interface version.

Return type short

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For IObservingConditionsV2, this will be 2. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this ObservingConditions.

property `ObservingConditions.`**`Name`**: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `ObservingConditions.`**`Pressure`**: **float**

Returns Atmospheric pressure (hPa) *at the observatory altitude*

Return type float

Raises

- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This must be the pressure *at the observatory altitude* and not the adjusted pressure at sea level. Please check whether your pressure sensor delivers local observatory pressure or sea level pressure. If it returns sea level pressure, your device must convert this to actual pressure at the observatory's altitude before returning a value to the client.

property `ObservingConditions.`**`RainRate`**: **float**

Returns Rain rate (mm/hr) at the observatory

Return type float

Raises

- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- The units of this property are millimetres per hour.
- This property can be interpreted as 0.0 = Dry and any positive nonzero value = wet. Rainfall intensity is classified according to the rate of precipitation:
- Light rain — when the precipitation rate is less than 2.5 mm (0.098 in) per hour
- Moderate rain — when the precipitation rate is between 2.5 mm (0.098 in) and 10 mm (0.39 in) per hour
- Heavy rain — when the precipitation rate is between 10 mm (0.39 in) and 50 mm (2.0 in) per hour
- Violent rain — when the precipitation rate is > 50 mm (2.0 in) per hour

property `ObservingConditions.`**`SkyBrightness`**: **float**

Returns Sky brightness (Lux) at the observatory

Return type float

Raises

- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

This property returns the sky brightness measured in Lux.

- 0.0001 lux Moonless, overcast night sky (starlight)
- 0.002 lux Moonless clear night sky with airglow
- 0.27–1.0 lux Full moon on a clear night
- 3.4 lux Dark limit of civil twilight under a clear sky
- 50 lux Family living room lights (Australia, 1998)
- 80 lux Office building hallway/toilet lighting
- 100 lux Very dark overcast day
- 320–500 lux Office lighting
- 400 lux Sunrise or sunset on a clear day.
- 1000 lux Overcast day; typical TV studio lighting
- 10000–25000 lux Full daylight (not direct sun)
- 32000–100000 lux Direct sunlight

property `ObservingConditions.`**`SkyQuality`**: **float**

Returns Sky quality (mag per sq-arcsec) at the observatory

Return type float

Raises

- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

Examples of typical sky quality values were published by Sky and Telescope (http://www.skyandtelescope.com/astronomy-resources/rate-your-skyglow/) and, in slightly adapted form, are reproduced below:

Sky Quality (mag/arcsec^2)	Description
22.0	By convention, this is often assumed to be the average brightness of a moonless night sky that's completely free of artificial light pollution.
21.0	This is typical for a rural area with a medium-sized city not far away. It's comparable to the glow of the brightest section of the northern Milky Way, from Cygnus through Perseus.
20.0	This is typical for the outer suburbs of a major metropolis. The summer Milky Way is readily visible but severely washed out.
19.0	Typical for a suburb with widely spaced single-family homes. It's a little brighter than a remote rural site at the end of nautical twilight, when the Sun is 12° below the horizon.
18.0	Bright suburb or dark urban neighborhood. It's also a typical zenith skyglow at a rural site when the Moon is full. The Milky Way is invisible, or nearly so.
17.0	Typical near the center of a major city.
13.0	The zenith skyglow at the end of civil twilight, roughly a half hour after sunset, when the Sun is 6° below the horizon. Venus and Jupiter are easy to see, but bright stars are just beginning to appear.
7.0	The zenith skyglow at sunrise or sunset

property `ObservingConditions.SkyTemperature`: **float**

Returns Sky temperature (degrees Celsius) at the observatory

Return type float

Raises
- `PropertyNotImplementedException` – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- The units of this property are degrees Celsius. Driver and application authors can use the ConvertUnits(Double, Units, Units) method to convert these units to and from degrees Fahrenheit.
- This is expected to be returned by an infra-red sensor looking at the sky. The lower the temperature the more the sky is likely to be clear.

property `ObservingConditions.`**`StarFWHM`**: **float**

Returns Seeing (FWHM in arc-sec) at the observatory

Return type float

Raises

- **PropertyNotImplementedException** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `ObservingConditions.`**SupportedActions**: **COM: ArrayList of String elements, Alpaca: Array of String**

Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

property `ObservingConditions.`**`Temperature`**: float

Returns Atmospheric temperature (degrees Celsius) at the observatory

Return type float

Raises
- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `ObservingConditions.`**`WindDirection`**: float

Returns Direction (deg) from which the wind is blowing at the observatory

Return type float

Raises
- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- **Meteorological 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 must be reported as 0.

property `ObservingConditions.`**`WindGust`**: **float**

Returns Peak 3 second wind gust (m/s) at the observatory over the last 2 minutes

Return type float

Raises
- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `ObservingConditions.`**`WindSpeed`**: **float**

Returns Wind speed (m/s) at the observatory

Return type float

Raises
- **`PropertyNotImplementedException`** – The device does not implement 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.



5.7 IRate Interface

class `Rate`

Describes a range of rates supported by the `Telescope.MoveAxis()` method (degrees/per second). These are contained within an the `Telescope.AxisRates`

collection and serve to describe one or more supported ranges of rates of motion about a mechanical axis.It is possible that the Maximum and Minimum properties will be equal. In this case, the `Rate` object expresses a single discrete rate. Both the `Minimum` and `Maximum` properties are always expressed in units of degrees per second.

5.7.1 Properties

`Rate.Maximum:`

Returns The maximum rate (degrees per second)

Return type float

Raises
- **`InvalidValueException`** – If an invalid Axis 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

`Rate.Minimum:`

Returns The minimum rate (degrees per second)

Return type float

Raises
- **`InvalidValueException`** – If an invalid Axis 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.



5.8 IRotatorV4 Interface

class `Rotator`

Bases: `ASCOM.DeviceInterface`

ASCOM Standard IRotator V4 Interface

The Rotator V4 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 directly in PA for you as well as other apps that might

be using the rotator. See Section 4.7.

Important

It is *vital* that an instrument rotator prevent wrapping of cables going to the imager. This must be transparent to the rotation commands coming from the application, allowing the application to freely move the rotator between any two angles without limits or "dead zones".

5.8.1 Methods

`Rotator.Action` (*ActionName: str, ActionParameters*)

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.
Invoke the specified device-specific custom action

Parameters
- **`ActionName`** (*str*) – A name from `SupportedActions` that represents the action to be carried out.
- **`ActionParameters`** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises
- **`MethodNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`Rotator.`**`CommandBlind`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Rotator.`**`CommandBool`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Rotator.`**`CommandString`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns String response from the command

Return type string

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `Rotator.`**`Connect`**`( )`

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

Note

New in Rotator V4

async `Rotator.`**`Disconnect`**`( )`

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of

the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

Note

New in Rotator V4

`Rotator.`**`Halt`**`( )`

Immediately stop any rotator motion due to a previous `Move()` or `MoveAbsolute()` call.

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be short-lived synchronous. The client may still check `IsMoving` to see when it actually stops if this takes some time.

Note

- Some Rotators may not support this function, in which case `MethodNotImplementedException` exception must be raised.
- **Recommendation:** Host software should call this method upon initialization and, if it fails, disable the Halt button in the user interface.

`Rotator.`**`Move`**`(` *Position: float* `)`

Changed in version 3: Clarified that this method must be implemented
Starts rotation relative to the current position (degrees). See Section 4.7.
Non-blocking: Must return immediately with `IsMoving` = True if the operation has

successfully been started (unless it is already at the requested position). After the requested angle is successfully reached and motion stops, the `IsMoving` property must become False.

Important

It is *vital* that an instrument rotator prevent wrapping of cables going to the imager. This must be transparent to the rotation commands coming from the application, allowing the application to freely move the rotator between any two angles without limits or "dead zones".

Parameters **`Position`** (*`float`*) – Relative position to move in degrees from current `Position`. See Section 4.7.

Raises

- **`InvalidValueException`** – If `Position` is invalid.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

Note

- **Asynchronous**: The method returns as soon as the rotation operation has been successfully started, with the `IsMoving` property True (unless already at the requested position). After the requested angle is successfully reached and motion stops, the `IsMoving` property must become False.
- Calling `Move()` must cause 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.
- See Section 4.7.

`Rotator.`**`MoveAbsolute`** (*Position: float*)

Changed in version 3: Clarified that this method must be implemented
Starts rotation to the given `Position` (degrees). See Section 4.7.
Non-blocking: Must return immediately with `IsMoving` = True if the operation has *successfully* been started (unless it is already at the requested position). After the requested angle is successfully reached and motion stops, the `IsMoving` property must become False.

Important

It is *vital* that an instrument rotator prevent wrapping of cables going to the imager. This must be transparent to the rotation commands coming from the application, allowing the application to freely move the rotator between any two angles without limits or “dead zones”.

Parameters **`Position`** (*`float`*) – New position in degrees. See Section 4.7.

Raises

- **`InvalidValueException`** – If `Position` is invalid.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

Note

- **Asynchronous**: The method returns as soon as the rotation operation has been successfully started, with the `IsMoving` property True (unless already at the requested position). After the requested angle is successfully reached and motion stops, the `IsMoving` property becomes False.
- Calling `MoveAbsolute()` must cause the `TargetPosition` property to change to the `Position` parameter then starts rotation to `TargetPosition`.
- See Section 4.7.

`Rotator.`**`MoveMechanical`** (*Position: float*)

Added in version 3: Member added
Starts rotation to the given mechanical `Position` (degrees). See Section 4.7.
Non-blocking: Must return immediately with `IsMoving` = True if the operation has *successfully* been started (unless it is already at the requested position). After the requested angle is successfully reached and motion stops, the `IsMoving` property must become False.

Important

It is *vital* that an instrument rotator prevent wrapping of cables going to the imager. This must be transparent to the rotation commands coming from the application, allowing the application to freely move the rotator between any two angles without limits or “dead zones”.

Parameters **`Position`** (*`float`*) – New mechanical position in degrees. See Section 4.7.

Raises

- **InvalidValueException** – If `Position` is invalid.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

Note

- **Asynchronous**: The method returns as soon as the rotation operation has been successfully started, with the `IsMoving` property True (unless already at the requested position). After the requested angle is successfully reached and motion stops, the `IsMoving` property becomes False.
- See Section 4.7.
- This method is to address requirements that need a physical rotation angle such as taking sky flats.

`Rotator.`**Sync** (*Position: float*)

Added in version 3: Member added

Syncs the rotator to the specified position angle without moving it. See Section 4.7.

Parameters **Position** (*float*) – Synchronised rotator position angle. See Section 4.7.

Returns Nothing

Raises

- **InvalidValueException** – If `Position` is invalid.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented

Note

- Must be short-lived and synchronous.
- Once this method has been called and the sync offset determined, both the `MoveAbsolute()` method and the `Position` property must function in synced coordinates rather than mechanical coordinates. The sync offset must persist across driver starts and device reboots.
- The offset should be persistent across device startup and shutdown. It represents a mechanical offset that should not change.
- See Section 4.7.

`Rotator.SetupDialog( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.
Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

5.8.2 Properties

property `Rotator.CanReverse`**: Boolean**

Added in version 2: Member added

Returns The direction of rotation can be set via the `Reverse` property

Return type boolean

Raises
- `NotConnectedException` – If the device is not connected
- `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Attention!

Must be implemented, must always return True.

Note

- See Section 4.7

property `Rotator.Connected`**: boolean**

Changed in version 4: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of Rotator V4. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `Rotator.`**`Connecting`**: Boolean

Added in version 4: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.

property `Rotator.`**`Description`**: String

Added in version 2: Member added

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `Rotator.DeviceState`: List[StateValue]

Added in version 4: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

This device must return the following operational properties if they are known:

- `IsMoving`
- `MechanicalPosition`
- `Position`
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the Rotator Interface Version 4 and later.

property `Rotator.DriverInfo`: String

Added in version 2: Member added

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

Return type string

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parseable string, use the `DriverVersion` property.

property `Rotator.`**`DriverVersion`**: String

Added in version 2: Member added

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property `Rotator.`**`InterfaceVersion`**: Short

Added in version 2: Member added

Returns ASCOM Device *interface definition* version that this device supports. Should return 4 for this interface version.

Return type short

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For example, for IRotatorV4, this will be 4. It should not to be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.
- Clients can detect legacy V1 drivers by trying to read this property. If the driver raises an error, it is a V1 driver. V1 did not specify this property. A driver may also return a value of 1. In other words, a raised error or a return value of 1 indicates that the driver is a V1 driver.

property `Rotator.IsMoving`**: boolean**

Changed in version 3: Clarified that this method must be implemented

Returns The rotator is currently moving to a new position

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented.

Note

- This is the correct property to use to determine *successful* completion of a (non-blocking) `Move()`, `MoveAbsolute()`, or `MoveMechanical()` request. `IsMoving` must be True immediately upon returning from any of these three movement calls (unless already at the requested position), and must remain True until *successful* completion, at which time `IsMoving` must become False.

property `Rotator.MechanicalPosition`**: float**

Added in version 3: Member added

Returns The raw mechanical position of the rotator in degrees, *relative to the optics*. See `Sync()` and Section 4.7.

Return type float

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

- Must be implemented
- Applications must not use this as a way to determine if a (non-blocking) `MoveMechanical()` has completed. The `MechanicalPosition` may transit through the requested position before finally settling. Use the `IsMoving` property.

Note

- Note the "relative to the optics" in the definition of this property. See Section 4.7.1.

property `Rotator.Name`: String

Added in version 2: Member added

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

Return type string

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `Rotator.Position`: float

Changed in version 3: Clarified that this method must be implemented

Returns Current instantaneous Rotator position, allowing for any sync offset, in degrees. See Section 4.7.

Return type float

Raises

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

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

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

Note

- 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` must be equal to `MechanicalPosition`. Once called, however, the offset must remain across driver starts and device reboots.
- For more info see Section 4.7.

property `Rotator`**.Reverse: boolean**

Changed in version 3: Clarified that this method must be implemented
(Read/Write) Set or indicate rotation direction reversal. See Section 4.7.

Returns True if rotation direction is reversed

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented

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. See Section 4.7

property `Rotator`.**`StepSize`**: float

Returns The minimum rotation step size (degrees)

Return type float

Raises
- **`PropertyNotImplementedException`** – If the rotator does not know its step size.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

property `Rotator`.**`SupportedActions`**: COM: ArrayList of String elements, Alpaca: Array of String

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.
Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

property Rotator.**TargetPosition: float**

Changed in version 3: Clarified that this method must be implemented

Returns The destination position angle for Move() and MoveAbsolute().

Return type float

Note

This will contain the new Position, including any Sync() offset, immediately upon return from a call to Move(), MoveAbsolute()



5.9 ISafetyMonitorV3 Interface

class SafetyMonitor

Bases: ASCOM.DeviceInterface

ASCOM Standard ISafetyMonitor V3 Interface

5.9.1 Methods

SafetyMonitor.**Action** (*ActionName: str, ActionParameters*)

Invoke the specified device-specific custom action

Parameters

- **ActionName** (*str*) – A name from SupportedActions that represents the action to be carried out.
- **ActionParameters** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises

- **MethodNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`SafetyMonitor.`**`CommandBlind`** (*Command: str, Raw: bool*)

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

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

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`SafetyMonitor.`**`CommandBool`** (*Command: str, Raw: bool*)

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`SafetyMonitor.`**`CommandString`** (*Command: str, Raw: bool*)

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

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

Returns String response from the command

Return type string

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `SafetyMonitor.`**`Connect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

async `SafetyMonitor.`**`Disconnect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to False.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to False.

`SafetyMonitor.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

5.9.2 Properties

property `SafetyMonitor.`**`Connected`**: boolean

Changed in version 3: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.

(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.

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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of SafetyMonitor V3. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `SafetyMonitor.Connecting`: **Boolean**

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.

property `SafetyMonitor.`**`Description`**: String

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `SafetyMonitor.`**`DeviceState`**: List[StateValue]

Added in version 3: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

This device must return the following operational properties if they are known:

- `IsSafe`
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the SafetyMonitor Interface Version 3 and later.

property `SafetyMonitor.DriverInfo`: String

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

Return type string

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parse-able string, use the `DriverVersion` property.

property `SafetyMonitor.DriverVersion`: String

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

Return type string

Raises `DriverException` – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property `SafetyMonitor.InterfaceVersion: Short`

Returns ASCOM Device *interface definition* version that this device supports. Should return 3 for this interface version.

Return type short

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For example, for ISafetyMonitorV3, this will be 3. It should not be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.

property `SafetyMonitor.IsSafe: Boolean`

Returns True when the monitored state is safe for use.

Return type boolean

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes hardware or communication errors.

Attention!

Must be implemented and must not throw a PropertyNotImplementedException

property `SafetyMonitor.Name: String`

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

The `Description` property is used to return info about the *device* rather than the *driver*.

property `SafetyMonitor.SupportedActions`: COM: ArrayList of String elements, Alpaca: Array of String

Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.



5.10 IStateValue Interface

class `StateValue`

Each device type has a property `DeviceState` which returns a set of aggregated operational properties of the device. Each of these operational properties are returned as an object with `Name` and `Value` properties.

5.10.1 Properties

property `Name`: string

The name of an operational property. The name is case sensitive and must match the property name's spelling and casing in the relevant ASCOM interface specification.

property `Value`: object

The corresponding value of the named operational property. The `StateValue.Value` property has the object type so that it can accept any type including the types commonly used in ASCOM interfaces such as int16, int32, double, string and enum. This approach avoids localisation complexities when transferring numeric and bool types

Important

- `Integer` values should be transferred as little endian byte sequences.
- `Floating point` values should be transferred using the 32-bit single-precision and 64-bit double-precision `IEC 60559` formats for `Single` and `Double` values respectively.
- `Boolean` values should be transferred using integer `0` for `FALSE` and a non-zero value to represent `TRUE`.
- `String` values should be `UTF16` encoded without a null terminator.

These requirements are met by the Microsoft C++ and .NET languages, for other languages, please consult your language implementation documentation.



5.11 ISwitchV3 Interface

class `Switch`

Bases: `ASCOM.DeviceInterface`

ASCOM Standard ISwitch V3 Interface

Important

This interface has some potentially confusing aspects. Please see Section 4.16

5.11.1 Methods

`Switch.`**`Action`** (*ActionName: str, ActionParameters*)

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.

Invoke the specified device-specific custom action

Parameters

- **`ActionName`** (*`str`*) – A name from `SupportedActions` that represents the action to be carried out.
- **`ActionParameters`** (*`str`*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises

- **`MethodNotImplementedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`Switch.CanAsync` (*Id: int*)

Added in version 3: Member added
Flag indicating whether this switch can operate asynchronously. See `SetAsync` for details of asynchronous switch operations.

Parameters **`Id`** (*`int`*) – the specified switch number (0 to `MaxSwitch` - 1)

Returns The specified switch device can operate asynchronously.

Return type boolean

Raises
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This is a mandatory method and must not throw a `MethodNotImplementedException`.

`Switch.CancelAsync` (*Id: int*)

Added in version 3: Member added
Cancels an in-progress asynchronous state change operation. See `SetAsync` for details of asynchronous switch operations.

Parameters **`Id`** (*`int`*) – the specified switch number (0 to `MaxSwitch` - 1)

Returns Nothing

Raises
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Attention!

This is a mandatory method and must not throw a `MethodNotImplementedException`.

Note

- On return, calls to `StateChangeComplete` for this switch will raise an `OperationCancelledException`, which will continue to be the case until a new asynchronous operation is started and `StateChangeComplete` returns either True or False as appropriate.
- See `SetAsync` for details of asynchronous switch operations.

`Switch.CanWrite` (*Id: int*)

Added in version 2: Member added

Parameters **`Id`** (*`int`*) – the specified switch number (0 to `MaxSwitch` - 1)

Returns True if the specified switch can be written to.

Return type boolean

Raises

- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Examples of switches that cannot be written to include a limit switch or a sensor.

`Switch.CommandBlind` (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and does not wait for a response.

Parameters

- **`Command`** (*`string`*) – The literal command string to be transmitted.
- **`Raw`** (*`bool`*) – If True, command is transmitted ‘as-is’. If False, then protocol framing characters may be added prior to transmis-

sion.If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Switch.`**`CommandBool`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

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

Returns True/False response from the command

Return type boolean

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Switch.`**`CommandString`** (*Command: str, Raw: bool*)

Added in version 2: Member added as part of common interface elements.

Deprecated since version 3: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

Parameters

- **Command** (*string*) – The literal command string to be transmitted.
- **Raw** (*bool*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns String response from the command

Return type string

Raises

- **MethodNotImplementedException** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `Switch.`**`Connect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.

Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to `True`.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be `True` unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) `False`.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to `True`.

Note

New in Switch V3

async `Switch.`**`Disconnect`**`( )`

Added in version 3: Preferred asynchronous connection mechanic. See Important section below.

Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting `Connected` to `False`.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be `True` unless already disconnected. Disconnect has *successfully* completed when `Connecting` becomes (or is) `False`.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to disconnect from a device rather than setting `Connected` to `False`.

Note

New in Switch V3

`Switch.`**`GetSwitch`**`(` *Id: int* `)`

Return the state of switch `Id` as a boolean. See Section 4.16

Parameters **`Id`** (*`int`*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns The state of the switch `Id`.

Return type boolean

Raises

- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`InvalidOperationException`** – If there is a temporary condition that prevents the switch's value being returned, including after power-up if the switch's state is unknown (not yet set).
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Do not use this as a way to tell if a `SetAsync()` has completed. It is possible that `GetSwitch(Id)` may be `True` or `False` while the async process is still underway. Use `StateChangeComplete()` to determine if a call to `SetAsync()` has completed

Note

- **For a variable output device `GetSwitch()` must return `False` if it is at its**
  minimum value, else `True`.
- Some switches do not support reading their state although they do allow state to be set. In these cases, on startup, the driver can not know the hardware state and it is recommended that the driver either:
  - Sets the switch to a known state on connection
  - Throws an `InvalidOperationException` until the client software has set the switch state for the first time

  In both cases the driver should save a local copy of the switch state which it last set and return this through `GetSwitch()` and `GetSwitchValue()`. See Section 4.16

`Switch.`**`GetSwitchDescription`** (*Id: int*)

Added in version 2: Member added.
Gets the description of the specified switch. This is to allow a fuller description of the switch to be returned, for example for a tool tip.

Parameters **`Id`** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns String giving the switch description.

Return type string

Raises
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

`Switch.`**`GetSwitchName`** (*Id: int*)

Gets the “short” name of the specified switch.

Parameters **Id** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns String giving the switch name.

Return type string

Raises

- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

`Switch.`**`GetSwitchValue`** (*Id: int*)

Added in version 2: Member added.
Return the value of switch `Id` as a float. See Section 4.16

Parameters **Id** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns The value of the switch `Id`. The value is expected to be between `MinSwitchValue()` and `MaxSwitchValue()` in steps of `SwitchStep`.

Return type float

Raises

- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`InvalidOperationException`** – If there is a temporary condition that prevents the device value being returned, including after power-up if the switch's state is unknown (not yet set).
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Do not use this as a way to tell if a `SetAsyncValue()` has completed. The value may be anything while the async process is still underway, *including the requested value*. Use `StateChangeComplete()` to determine if a call to `SetAsyncValue()` has completed

Note

- For a boolean on/off switch, `GetSwitchValue()` must return `MinSwitchValue` if the switch is off, and `MaxSwitchValue` if the switch is on.
- Some switches do not support reading their state although they do allow state to be set. In these cases, on startup, the driver can not know the hardware state and it is recommended that the driver either:
  - Sets the switch to a known state on connection
  - Throws an `InvalidOperationException` until the client software has set the switch state for the first time

In both cases the driver should save a local copy of the state which it last set and return this through `GetSwitchValue()` and `GetSwitch()`. See Section 4.16

`Switch.MaxSwitchValue` (*Id: int*)

Added in version 2: Member added.
Returns the maximum value for switch `Id`

Parameters **`Id`** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns The maximum value to which this switch can be set or which a read only sensor will return.

Return type float

Raises
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This is a mandatory method and must not throw a `MethodNotImplementedException`.

Note

- This must be greater than `MinSwitchValue()` for switch `Id`..
- For an on/off switch, `MaxSwitchValue` must return the value 1.0.

`Switch.MinSwitchValue` (*Id: int*)

Added in version 2: Member added.

Returns the minimum value for switch `Id`

Parameters **`Id`** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns The minimum value to which this switch can be set or which a read only sensor will return.

Return type float

Raises

- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This is a mandatory method and must not throw a `MethodNotImplementedException`.

Note

- This must be less than `MaxSwitchValue()` for switch `Id`.
- For an on/off switch, `MinSwitchValue` must return the value 0.0.

`Switch.`**`SetAsync`** (*Id: int, State: bool*)

Added in version 3: Member added.

Asynchronously Set a switch to the specified boolean on/off state.

Non-blocking: Returns immediately after *successfully* starting the state change with `StateChangeComplete()` for the given switch `Id` = `False` See Notes.

Parameters

- **`Id`** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)
- **`State`** (*bool*) – The required control state (on or off)

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If `CanAsync()` is False for switch `Id`
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- **Asynchronous** (non-blocking): The method returns as soon as the state change operation has been *successfully* started, with `StateChangeComplete()` for the given switch `Id` = `False`. After the state change has completed `StateChangeComplete()` becomes `True`.
- Switch devices are numbered from 0 to `MaxSwitch` - 1.
- `GetSwitchValue()` for switch `Id` must return `MaxSwitchValue()` for that switch if the set state is `True`, and `MinSwitchValue()` for that switch if the set state is `False`.

`Switch.SetAsyncValue` (*Id: int, Value: float*)

Added in version 3: Member added.
Asynchronously set a switch to the specified float value.

Parameters
- **`Id`** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)
- **`Value`** (*float*) – The value to be set, between `MinSwitchValue()` and `MaxSwitchValue()` for switch `Id`

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If either `CanWrite()` or `CanAsync()` is False for switch `Id`
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1), or if `Value` is not between `MinSwitchValue()` and `MaxSwitchValue()` for 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

`Switch.SetSwitch` (*Id: int, State: bool*)

Set a switch to the specified boolean on/off state

Parameters
- **`Id`** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)
- **`State`** (*bool*) – The required control state (on or off)

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If `CanWrite()` is False for switch `Id`
- **`InvalidValueException`** – If `Id` is out of range (0 to `MaxSwitch` - 1)
- **`NotConnectedException`** – If the device is not connected

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- GetSwitchValue() for switch Id must return MaxSwitchValue() for that switch if the set state is True, and MinSwitchValue() for that switch if the set state is False.

Switch.**SetSwitchName** (*Id: int, Name: str*)

Added in version 2: Member added.
Set a switch's name to the specified value.

Parameters
- **Id** (*int*) – The specified switch number (0 to MaxSwitch - 1)
- **Name** (*str*) – The name of the switch

Returns Nothing

Raises
- **MethodNotImplementedException** – If the switch name cannot be set by the client
- **InvalidValueException** – If Id is out of range (0 to MaxSwitch - 1)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Switch.**SetSwitchValue** (*Id: int, Value: float*)

Added in version 2: Member added.
Set a switch's value to the specified float value.

Parameters
- **Id** (*int*) – The specified switch number (0 to MaxSwitch - 1)
- **Value** (*float*) – The value to be set, between MinSwitchValue() and MaxSwitchValue() for switch Id

Returns Nothing

Raises
- **MethodNotImplementedException** – If CanWrite() is False for switch Id
- **InvalidValueException** – If Id is out of range (0 to MaxSwitch - 1), or if Value is not between MinSwitchValue() and MaxSwitchValue() for 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Switch.**SetupDialog**( )

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a MethodNotImplementedException

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

Switch.**StateChangeComplete**(*Id: int*)

Added in version 3: Member added.

Parameters **Id** (*int*) – The specified switch number (0 to MaxSwitch - 1)

Returns True if the last SetAsync() or SetAsyncValue() has completed and the switch is in the requested state.

Return type boolean

Raises

- **MethodNotImplementedException** – 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** – If Id is out of range (0 to MaxSwitch - 1), or if Value is not between MinSwitchValue() and MaxSwitchValue() for 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

`Switch.SwitchStep ( Id: int )`

Added in version 2: Member added.
The step size that this switch supports (the difference between successive values of the switch).

Parameters **Id** (*int*) – The specified switch number (0 to `MaxSwitch` - 1)

Returns The step size for this switch.

Return type float

Raises

- **InvalidValueException** – If `Id` is out of range (0 to `MaxSwitch` - 1), or if `Value` is not between `MinSwitchValue()` and `MaxSwitchValue()` for 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Important

`SwitchStep` must be greater than zero and the number of steps can be calculated as:

`` ((MaxSwitchValue - MinSwitchValue) / SwitchStep) + 1.``

5.11.2 Properties

property `Switch.Connected`**: boolean**

Changed in version 3: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of Switch V3. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `Switch.`**Connecting: Boolean**

Added in version 3: Preferred asynchronous connection mechanic. See Notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes `False` after calling either of these methods.
- New in ISwitchV3

property `Switch.Description`: String

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `Switch.DeviceState`: List[StateValue]

Added in version 3: To allow reduction of status polling

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

Since a single Switch device can be configured to provide many controllable switches, a standard is required to enable multiple controllable switch states to be returned by the Switch device. The standard is:

`The controllable switch number must be appended to the name of the property when constructing the DeviceState property name.`

For the `GetSwitch` property, DeviceState property names would start at GetSwitch0 and progress through GetSwitch1, GetSwitch2 etc. until reaching

the last controllable switch number: MaxSwitch - 1. A similar model applies to the `GetSwitchValue` and `StateChangeComplete` properties.

E.g. For a Switch device with four controllable switches, the DeviceState property names would be:

- GetSwitch0, GetSwitch1, GetSwitch2, GetSwitch3
- GetSwitchValue0, GetSwitchValue1, GetSwitchValue2, GetSwitchValue3
- StateChangeComplete0, StateChangeComplete1, StateChangeComplete2, StateChangeComplete3
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the Switch Interface Version 3 and later.

property `Switch.DriverInfo`**: String**

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parse-able string, use the `DriverVersion` property.

property `Switch.DriverVersion`**: String**

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property `Switch.`**`InterfaceVersion`**: Short

Returns ASCOM Device *interface definition* version that this device supports. Should return 3 for this interface version.

Return type short

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For example, for ISwitchV3, this will be 3. It should not be confused with the `DriverVersion` property, which is the major.minor version of the driver for this device.

property `Switch.`**`MaxSwitch`**: integer

Count of switches managed by this driver.

Returns Number of switches managed by this driver

Return type integer

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

> **Note**
>
> Switches are numbered from 0 to `MaxSwitch` - 1.

property `Switch`.**`Name`: String**

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

> **Attention!**
>
> Must be implemented, must not throw a `PropertyNotImplementedException`

> **Note**
>
> The `Description` property is used to return info about the *device* rather than the *driver*.

property `Switch`.**`SupportedActions`: COM: ArrayList of String elements, Alpaca: Array of String**

Added in version 2: Recommended over (now) deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` as a more flexible extension mechanic.
Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

> **Attention!**
>
> Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.



5.12 ITelescope V4 Interface

class `Telescope`

Bases: `ASCOM.DeviceInterface`

ASCOM Standard ITelescope V4 Interface

Important

All mounts that can be slewed must implement asynchronous slewing. ASCOM COM drivers must also implement synchronous slewing for backward compatibility. See Section 4.8. Clients must use asynchronous slewing methods if at all possible, use of synchronous slewing methods is deprecated in ITelescope V4.

Revision History

Members added, deprecated, or changed are indicated by a legend at the top of that member indicating the `InterfaceVersion` at which the change occurred. For this interface, only changes after `InterfaceVersion = 2` (2004) are indicated. Telescope V1.x existed over 20 years ago during the period where ASCOM was experimental and subject to developmental changes.

5.12.1 Methods

async `Telescope.`**`AbortSlew`**`( )`

Stops any motion in progress: slewing, parking, find-home, and move-axis.

Non-blocking: Returns immediately with `Slewing = True` until the motion has been stopped, at which time `Slewing` becomes `= False`. See Notes, and Section 4.1.

Returns Nothing

Raises
- **MethodNotImplementedException** – If not supported.
- **ParkedException** – If the mount is parked (ITelescopeV4 and later).
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- **Asynchronous** (non-blocking): Use the `Slewing` property to monitor stopping of motion. When the mount has *successfully* stopped the motion, `Slewing` becomes `False`. See Section 4.1
- Does nothing if no slew/motion is in progress
- In the case of `MoveAxis()` or `SlewToAltAzAsync()`, `Tracking` must be returned to the state before the slew stopped. Note that equatorial slews by definition always start and finish with `Tracking` on.
- **New in ITelescopeV4** - This method must throw a `ParkedException` when parked rather than throwing a more generic `DriverException`.

`Telescope.`**`Action`** (*ActionName: str, ActionParameters*)

Added in version 3: To replace deprecated `CommandBlind()`, `CommandBool()`, and `CommandString()` with a more flexible extension mechanic.
Invoke the specified device-specific custom action

Parameters
- **ActionName** (*str*) – A name from `SupportedActions` that represents the action to be carried out.
- **ActionParameters** (*str*) – List of required arguments or empty string if none are required.

Returns Action response. The meaning of returned strings is set by the driver author. See notes below.

Return type string

Raises
- **MethodNotImplementedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Important

- **Must be implemented** but may throw `MethodNotImplementedException` if no custom actions are supported.
- This method, combined with `SupportedActions`, is the supported mechanic for adding non-standard functionality.

Note

- Action names must be case insensitive, so for example SelectWheel, selectwheel and SELECTWHEEL all refer to the same action.
- An example of a string response: Suppose filter wheels start to appear with automatic wheel changers; new actions could be QueryWheels and SelectWheel. The former returning a formatted list of wheel names and the second taking a wheel name and making the change, returning appropriate values to indicate success or failure.

`Telescope.`**`AxisRates`** (*Axis: TelescopeAxes*)

Determine the rates at which the telescope may be moved about the specified axis by the `MoveAxis()` method. See `MoveAxis()` and Section 4.11 for details.

Parameters **`Axis`** (`TelescopeAxes`) – The *mechanical* axis about which rate information is desired

Returns A list or collection of `Rate` objects, each of which specifies a minimum and a maximum angular rate (degrees/second) at which the given axis of the mount may be moved.

Return type A list or collection of `Rate` objects

Raises

- **`InvalidValueException`** – If an invalid Axis 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`;

Note

- See `MoveAxis()` and Section 4.11 for details.
- A mount may specify one or more ranges of rates on each of its axes.
- An empty list must be returned if `MoveAxis()` is not supported.
- Returned rates must always be positive. It is up to the client to choose the rotation direction as a positive or negative rate for the call to `MoveAxis()`.
- `MoveAxis()` is a complex feature, see `TelescopeAxes` and Section 4.11.

`Telescope.`**`CanMoveAxis`** (*Axis: TelescopeAxes*)

The mount can be moved about the given *mechanical* axis. See `MoveAxis()` for details.

Parameters **`Axis`** (`TelescopeAxes`) – The axis about which this information is required

Returns True if the mount can move about the requested axis, else False

Return type boolean

Raises
- **`InvalidValueException`** – If an invalid Axis 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

See `MoveAxis()` for details.

`Telescope.`**`CommandBlind`** (*Command: str, Raw: boolean*)

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.

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

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

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Telescope.`**`CommandBool`** (*Command: str, Raw: boolean*)

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a boolean response.

Parameters
- **Command** (*string*) – The literal command string to be transmitted.
- **Raw** (*boolean*) – If True, command is transmitted 'as-is'. If False, then protocol framing characters may be added prior to transmission.

Returns True/False response from the command

Return type boolean

Raises
- **MethodNotImplementedException** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

`Telescope.`**`CommandString`** (*Command: str, Raw: boolean*)

Deprecated since version 4: Use the more flexible `Action()` and `SupportedActions` mechanic. See Notes below.
Transmit an arbitrary string to the device and wait for a string response.

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

Returns String response from the command

Return type string

Raises
- **`MethodNotImplementedException`** – If the method 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Deprecated, may result in `MethodNotImplementedException`

Note

The CommandXXX methods are a historic mechanic that provides clients with direct and unimpeded access to change device hardware configuration. While highly enabling for clients, this mechanic is inherently risky because clients can fundamentally change hardware operation without the driver being aware that a change is taking / has taken place.

The newer `Action` and `SupportedActions` mechanic provides discrete, named, functions that can deliver any functionality required. They do need driver authors to make provision for them within the driver, but this approach is much lower risk than using the CommandXXX methods because it enables the driver to resolve conflicts between standard device interface commands and extended commands provided as Actions. The driver is always aware of what is happening and can adapt more effectively to client needs.

async `Telescope.`**Connect** ( )

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Connect to the device asynchronously. Use this to connect to a device rather than setting `Connected` to True.

Returns Nothing

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, `Connecting` must be True unless already connected. Connection has *successfully* completed when `Connecting` becomes (or is) False.
- This is a mandatory method and must not throw a `MethodNotImplementedException`.
- Use this to connect to a device rather than setting `Connected` to True.

`Telescope.`**DestinationSideOfPier** (*RightAscension: float, Declination: float*)

Parameters
- **`RightAscension`** (*`float`*) – The destination right ascension (hours)
- **`Declination`** (*`float`*) – The destination declination (degrees, positive North)

Returns The `PierSide` indicating the pointing state in which the mount will be if slewed to the given coordinates at this instant of time.

Return type `PierSide`

Provided so apps can manage GEM flipping during an image sequence. See

SideOfPier, Section 4.13 and Section 4.10

Raises
- **MethodNotImplementedException** – If the method is not implemented
- **InvalidValueException** – If an invalid RightAscension or Declination 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

async Telescope.**Disconnect**( )

Added in version 4: Preferred asynchronous connection mechanic. See Important section below.
Disconnect from the device asynchronously. Use this to disconnect from a device rather than setting Connected to False.

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Important

- **Non-Blocking**. On return, Connecting must be True unless already disconnected. Disconnect has *successfully* completed when Connecting becomes (or is) False.
- This is a mandatory method and must not throw a MethodNotImplementedException.
- Use this to disconnect from a device rather than setting Connected to False.

async Telescope.**FindHome**( )

Changed in version 4: Formally defined as asynchronous
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 Section 4.1.

Returns Nothing

Raises
- **MethodNotImplementedException** – If the method is not implemented (and CanFindHome = False)
- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

> **Note**
>
> **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation's progress. When the mount has *successfully* reached its home position, `Slewing` becomes `False` and `AtHome` becomes `True`. See Section 4.1

async `Telescope.`**MoveAxis** (*Axis: TelescopeAxes*, *Rate: float*)

Changed in version 4: Formally defined as asynchronous

Start motion of the mount about the given *mechanical* axis at the given non-zero angular rate or, for a zero angular rate, stop `MoveAxis()` movement about the axis and resume any configured tracking movement. See `TelescopeAxes` and Section 4.11.

Non-blocking: Must return immediately with `Slewing` = `True` for `Rate` > 0 after *successfully* starting the axis rotation operation. `Rate` = 0 stops motion immediately.

Parameters

- **Axis** (`TelescopeAxes`) – The *mechanical* axis about which motion is desired
- **Rate** (*float*) – The rate of rotation desired (degrees/second)

Returns Nothing

Raises

- **MethodNotImplementedException** – If the method is not implemented
- **ParkedException** – If the mount is parked (`AtPark` = `True`)
- **InvalidValueException** – If an invalid axis or rate value is given
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

> **Attention!**
>
> A call with `Rate` = 0 is required to stop motion and *return to the previous tracking state of that axis*. See Section 4.11

Note

- **Asynchronous** (non-blocking): Clients must 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.
- When the motion is stopped the scope will be set to the previous `TrackingRate` or to no movement, depending on the previous state of the `Tracking` property.
- A mount may support multiple rate ranges (via `AxisRates()`) on each of its multiple `TelescopeAxes`
- This is a complex feature and relates to the mount's *mechanical* axes. See `TelescopeAxes` and Section 4.11.
- The meaning of positive vs negative values as applies to rotation directions about the axes is purposely left undefined. App developers need to provide adaptation to various mount geometries and control systems and their rotation directions. See Section 4.11.

Important

- A `MoveAxis()` operation on one axis must not affect any of the other axes. See Section 4.11
- Do not use this method to effect guiding. Use `PulseGuide()`. See Section 4.15.
- Use `RightAscensionRate` and `DeclinationRate` to effect the needed slight adjustments to normal sidereal tracking for solar system objects (major and minor planets, comets). See Section 4.14
- This is provided for applications such as software hand-boxes and tracking satellites in coordinate systems other than equatorial.

async `Telescope.`**Park** ( )

Changed in version 4: Formally defined as asynchronous
Move the telescope to its park position, stop all motion (or restrict to a small safe range), and set AtPark to `True`.
Non-blocking: Returns immediately with:

- `Slewing` = `False` and `AtPark` = `True` if mount is already in its parked state.
- `Slewing` = `True` and `AtPark` = `False` if the park operation has *successfully* started but has not yet completed.

While the park operation is underway, `Slewing` must return `True` until the driver determines that the park is complete, at which point the driver must set `AtPark` to `True` before setting `Slewing` to `False`.
See Notes, and Section 4.1.

Returns Nothing

Raises

- **MethodNotImplementedException** – If the method is not implemented (and CanPark = False)
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- When a mount supports parking, one or more Parked states may be offered by the manufacturer / driver. It is the astronomer's responsibility to select an appropriate state for their needs by configuring the mount and driver. The Park command will deliver whatever state has been selected by the astronomer.
- Parking should put the telescope into a state where its pointing accuracy must not be lost if it is power-cycled (without moving it).
- It is permissible for the mount to require power cycling before being unparked (with Unpark()) and used.
- Parking a mount that is already parked (AtPark = True) is harmless and must always be successful.

async Telescope.**PulseGuide** (*Direction: GuideDirections*, *Duration: int*)

Changed in version 4: Formally defined as asynchronous
Moves the mount in the specified angular direction for the specified time (ms). The directions are in the **Equatorial coordinate system** only, regardless of the mount's AlignmentMode. The distance moved depends on the GuideRateDeclination and GuideRateRightAscension, as well as Duration. See Section 4.15.
Non-blocking: See Notes, and Section 4.1

Parameters

- **Direction** (GuideDirections) – Equatorial axis and direction of guide motion
- **Duration** (*int*) – The duration of the guide-rate motion (milliseconds)

Raises

- **MethodNotImplementedException** – If the method is not implemented (CanPulseGuide is False)
- **InvalidValueException** – If an invalid Direction or Duration is given
- **InvalidOperationException** – If the pulse guide cannot be effected e.g. if the telescope is slewing (Slewing is True), or is not tracking (Tracking is False). This exception may also be thrown by a mount that is incapable of performing pulse guides in both RA and Dec simultaneously. *Please make this limitation*

clear in your error message! See Section 4.15.

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- **Asynchronous**: The method returns as soon as 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 Section 4.1
- If the device cannot have simultaneous `PulseGuide` operations in both RightAscension and Declination, it must throw `InvalidOperationException` when the overlapping operation is attempted.
- See Section 4.15

`Telescope.`**`SetPark`**`( )`

Set the mount's Parked state to be at its current position.

Returns Nothing

Raises

- **MethodNotImplementedException** – If the method is not implemented (`CanPark` is False)
- **InvalidOperationException** – If the Park position cannot be set e.g. because the mount is slewing (`Slewing` is True), or `CanSetPark` is False, or the mount is in some other state that prevents setting the park position.
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

`Telescope.`**`SetupDialog`**`( )`

Launches a configuration dialogue box for the driver. The call will not return until the user clicks OK or cancels manually.

Please note that this method is only valid for COM drivers. Alpaca devices should provide configuration through the Alpaca HTML endpoints and should not implement a SetupDialog endpoint.

Returns Nothing

Raises **DriverException** – An error occurred that is not described by one of

the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `MethodNotImplementedException`

Note

- **Blocking** It is permissible that the configuration dialog is *modal*, and for the driver not to respond to other calls while this dialog is open.

`Telescope.`**`SlewToAltAz`** (*Azimuth: float*, *Altitude: float*)

Deprecated since version 4: Use `SlewToAltAzAsync()`. See Deprecation Notice below.
Move the mount synchronously to the given local horizontal coordinates, return only when slew is complete. See Section 4.8.

Deprecation Notice

This method is deprecated for clients as of ITelescope V4. Use `SlewToAltAzAsync()`. ASCOM COM mounts that can slew to local horizontal coordinates at all must implement this, however, to prevent a breaking change. ASCOM Alpaca mounts should not implement this and should return False for `CanSlewAltAz`, and `MethodNotImplementedException` if called by the client. See Section 4.8.

Parameters

- **`Azimuth`** (*`float`*) – Destination azimuth coordinate (degrees, North-referenced, positive East/clockwise)
- **`Altitude`** (*`float`*) – Destination altitude coordinate (degrees, positive up).

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method is not implemented (`CanSlewAltAz` is False)
- **`InvalidOperationException`** – If `Tracking = True`, or if `AtPark = True`, or if the requested slew would fail due to hardware limit(s).
- **`InvalidValueException`** – If an invalid `Azimuth` or `Altitude` is given
- **`ParkedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Note

- It is impossible for the mount to be slewed to an Alt/Az coordinate if `Tracking = True` since the mount would instantly move away from the requested Alt/Az coordinates due to the equatorial tracking motion. Hence the requirement for raising `InvalidOperationException` if `Tracking = True`

async `Telescope.`**`SlewToAltAzAsync`** (*Azimuth: float*, *Altitude: float*)

Changed in version 4: If the mount can slew to local horizontal coordinates, it must implement this method.
Start a slew to the given local horizontal coordinates.
Non-blocking: Returns immediately. See Notes, and Section 4.1

Parameters
- **`Azimuth`** (*`float`*) – Destination azimuth coordinate (degrees, North-referenced, positive East/clockwise)
- **`Altitude`** (*`float`*) – Destination altitude coordinate (degrees, positive up).

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method is not implemented (`CanSlewAltAzAsync` is False)
- **`InvalidOperationException`** – If `Tracking = True`, or if `AtPark = True`, or if the requested slew would fail due to hardware limit(s).
- **`InvalidValueException`** – If an invalid `Azimuth` or `Altitude` is given
- **`ParkedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

Asynchronous (non-blocking): Use the `Slewing` property to monitor the operation. When 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 Section 4.1.

- It is impossible for the mount to be slewed to an Alt/Az coordinate if `Tracking = True` since the mount would instantly move away from the requested Alt/Az coordinates due to the equatorial tracking motion. Hence the requirement for raising `InvalidOperationException` if `Tracking = True`

Important

As of ITelescope V4 if the mount can slew to local horizontal coordinates, it must implement this method.

`Telescope.`**`SlewToCoordinates`** (*RightAscension: float*, *Declination: float*)

Deprecated since version 4: Use `SlewToCoordinatesAsync()`. See Deprecation Notice below.
Move the mount to the given equatorial coordinates per `EquatorialSystem`, return only when slew is complete. See Section 4.8.

Deprecation Notice

This method is deprecated for clients as of ITelescope V4. Use `SlewToCoordinatesAsync()`. ASCOM COM Mounts that can slew to equatorial coordinates at all must continue to implement this, however, to prevent a breaking change. ASCOM Alpaca mounts should not implement this and should return False for `CanSlew`, and `MethodNotImplementedException` if called by the client. See Section 4.8.

Parameters

- **`RightAscension`** (*`float`*) – Destination right ascension coordinate (hours, per `EquatorialSystem`). Copied to `TargetRightAscension`
- **`Declination`** (*`float`*) – Destination declination coordinate (degrees, per `EquatorialSystem`). Copied to `TargetDeclination`

Returns Nothing

Raises

- **`MethodNotImplementedException`** – If the method is not implemented and (`CanSlewAsync` is False)
- **`InvalidValueException`** – If an invalid `RightAscension` or `Declination` is given
- **`InvalidOperationException`** – If `Tracking` is False or if the requested slew would fail due to hardware limit(s).

- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

async `Telescope.`**SlewToCoordinatesAsync** (*RightAscension: float*, *Declination: float*)

Changed in version 4: If the mount can slew to equatorial coordinates, it must implement this method.
Start a slew to the given equatorial coordinates per `EquatorialSystem`.
Non-blocking: Returns immediately. See Notes, and Section 4.1

Parameters
- **RightAscension** (*float*) – Destination right ascension coordinate (hours, per `EquatorialSystem`). Copied to `TargetRightAscension`
- **Declination** (*float*) – Destination declination coordinate (degrees, per `EquatorialSystem`). Copied to `TargetDeclination`

Returns Nothing

Raises
- **MethodNotImplementedException** – If the method is not implemented and (`CanSlewAsync` is False)
- **InvalidValueException** – If an invalid `RightAscension` or `Declination` is given
- **InvalidOperationException** – If `Tracking` is False or if the requested slew would fail due to hardware limit(s).
- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

Asynchronous (non-blocking): Use the `Slewing` property to monitor the operation. When 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 Section 4.1.

Telescope.SlewToTarget ( )

Deprecated since version 4: Use SlewToTargetAsync(). See Deprecation Notice below.

Move the mount to the TargetRightAscension and TargetDeclination coordinates per EquatorialSystem, return only when slew is complete. See Section 4.8.

Returns Nothing

Deprecation Notice

This method is deprecated for clients as of ITelescope V4. Use SlewToTargetAsync(). ASCOM COM Mounts that can slew to equatorial coordinates at all must continue to implement this, however, to prevent a breaking change. ASCOM Alpaca mounts should not implement this and should return False for CanSlew, and MethodNotImplementedException if called by the client. See Section 4.8.

Raises

- **MethodNotImplementedException** – If the method is not implemented (CanSlew is False).
- **InvalidOperationException** – If Tracking is False or if the requested slew would fail due to hardware limit(s).
- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

async Telescope.SlewToTargetAsync ( )

Changed in version 4: If the mount can slew to equatorial coordinates, it must implement this method.

Start an asynchronous slew to the coordinates in TargetRightAscension and TargetDeclination per EquatorialSystem

Non-blocking: Returns immediately with Slewing = True if the slewing operation has *successfully* been started. See Notes, and Section 4.1

Returns Nothing

Raises

- **MethodNotImplementedException** – If the method is not implemented (CanSlewAsync is False)
- **InvalidOperationException** – If Tracking is False or if the requested slew would fail due to hardware limit(s).
- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by

someone other than yourself. Includes communication errors.

> **Note**
>
> **Asynchronous** (non-blocking): Use the `Slewing` property to monitor the operation. When the requested coordinates have been successfully* reached, `Slewing` becomes False. If `SlewToTargetAsync()` returns with `Slewing` = False then the mount was already at the requested coordinates, which is also a success. See Section 4.1.

`Telescope.`**`SyncToAltAz`** (*Azimuth: float*, *Altitude: float*)

Match the mount's local horizontal coordinates to the given local horizontal coordinates.

Parameters
- **`Azimuth`** (*`float`*) – Destination azimuth coordinate (degrees, North-referenced, positive East/clockwise)
- **`Altitude`** (*`float`*) – Destination altitude coordinate (degrees, positive up).

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method is not implemented (`CanSyncAltAz` is False)
- **`InvalidValueException`** – If an invalid `Azimuth` or `Altitude` is given
- **`ParkedException`** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

> **Note**
>
> May throw `DriverException` if `Tracking` is True.

`Telescope.`**`SyncToCoordinates`** (*RightAscension: float*, *Declination: float*)

Match the mount's equatorial coordinates with the given equatorial coordinates

Parameters
- **`RightAscension`** (*`float`*) – Destination right ascension coordinate (hours, per `EquatorialSystem`). Copied to `TargetRightAscension`
- **`Declination`** (*`float`*) – Destination altitude coordinate (degrees, per `EquatorialSystem`). Copied to `TargetDeclination`

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method is not

implemented (CanSync is False)

- **InvalidValueException** – If an invalid RightAscension or Declination is given
- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

May throw DriverException if Tracking is False.

Telescope.**SyncToTarget**( )

Match the mount's equatorial coordinates with the coordinates in TargetRightAscension and TargetDeclination.

Returns Nothing

Raises

- **MethodNotImplementedException** – If the method is not implemented (CanSync is False)
- **InvalidValueException** – If an invalid RightAscension or Declination is given
- **ParkedException** – 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

May throw DriverException if Tracking is False.

async Telescope.**Unpark**( )

Changed in version 4: Formally defined as asynchronous
Takes telescope out of the Parked state. The state of Tracking after unparking is undetermined. Valid only after Park().
Non-blocking: Returns immediately with:

- Slewing = False and AtPark = False if the mount is already in its unparked state.
- Slewing = True and AtPark = True if the unpark operation has *successfully* started but has not yet completed.

While the unpark operation is underway, Slewing must return True until the driver

determines that the unpark is complete, at which point the driver must set `AtPark` to `False` before setting `Slewing` to `False`.
See Notes, and Section 4.1.

Returns Nothing

Raises
- **`MethodNotImplementedException`** – If the method is not implemented (and `CanPark` = False)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Unparking a mount that is not parked (`AtPark` = False) is harmless and must always be successful.

5.12.2 Properties

property `Telescope`.`AlignmentMode`: enum Telescope.AlignmentModes

Returns The *mechanical construction* of the mount and the *orientation of its axes* (Alt/Az, Polar, German Polar), etc.

Return type Telescope.AlignmentModes

Raises
- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

This property is read-only; it cannot be changed at run-time because it reflects the *design* of the mount. Regardless of their `AlignmentMode` all mounts may operate in equatorial (RA/Dec) and/or local horizontal (Alt/Az) coordinate systems. `AlignmentMode` is unrelated to the coordinate systems in use by the mount.

Note

This property describes the *mechanical construction* of the mount, not the coordinate system being used operationally. Thus `AlignmentMode` cannot be changed or set by client applications. Astronomical mounts all have the ability to to read back and slew using equatorial (RA and Dec) coordinates, and most may also read back and slew using local horizontal (Alt and Az) coordinates, independent of their mechanical construction.

The `AlignmentMode` does not restrict a mount's ability to be pointed/slewed via equatorial and/or local horizontal coordinates and/or to read back RA/Dec and/or Alt/Az (e.g. a push-to Dobsonian may display both RA/Dec and Alt/Az).

property `Telescope.`**`Altitude`**: **float**

Returns The Altitude (degrees) above the horizon at which the mount is currently pointing (local horizontal coordinates).

Return type float

Raises

- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Telescope.`**`ApertureArea`**: **float**

Returns The telescope's effective aperture area (square meters).

Return type float

Raises

- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- The area takes into account any obstructions; it is the actual light-gathering area.

property `Telescope.ApertureDiameter`: **float**

Returns The telescope's effective aperture (meters).

Return type float

Raises
- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Telescope.AtHome`: **boolean**

Returns The mount is at the home position.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- `Slewing` is the correct property to use to determine *successful* completion of the (non-blocking) `FindHome()` operation. Using `AtHome` runs the risk that the mount will pass through the home position before finally completing the homing operation. See Section 4.1
- True if the mount is stopped in the Home position. Can be True only following a `FindHome()` operation.
- `AtHome` must become False immediately upon any slewing operation
- Must always be False if the mount does not support homing. Use `CanFindHome` to determine if the mount supports homing.

property `Telescope.AtPark`: **boolean**

Returns `True` if the telescope has been put into the parked state by the `Park()` method. Set `False` by calling the `Unpark()` method.

Return type boolean

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

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- ASCOM clients should note that `Slewing` is the most reliable way determine *successful* completion of the (non-blocking) `Park()` and `Unpark()` operations. Some drivers tie the state of AtPark to specific mount coordinates and using `AtPark` runs the risk that the mount will pass through and overshoot the configured *Park* position before returning to the expected park position. See Section 4.1
- Clients should be prepared for the possibility that the mount may report `AtPark` as `True` on initial connection.
- When parked, the mount must be stationary or restricted to a small safe range of movement. `Tracking` must be False.
- You must take the mount out of park by calling `Unpark()`.
- Attempts to slew, move or enable tracking while parked must raise a `ParkedException`.
- Must always be False if the mount does not support parking. Use `CanPark` to determine if the mount supports parking.

property `Telescope.`**Azimuth: float**

Returns The azimuth (degrees) at which the mount is currently pointing (local horizontal coordinates).

Return type float

Raises

- **PropertyNotImplementedException** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Azimuth is per the usual alt/az coordinate convention: degrees North-referenced, positive East/clockwise.

property `Telescope.`**`CanFindHome`**: **boolean**

Returns The mount can find its home position.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `FindHome()`.

property `Telescope.`**`CanPark`**: **boolean**

Returns The mount can be parked.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- See `Park()`.

property `Telescope.`**`CanPulseGuide`**: **boolean**

Returns The mount can be pulse guided.

Return type boolean

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

detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- See `PulseGuide()`.

property `Telescope.`**`CanSetDeclinationRate`**: **boolean**

Returns The Declination tracking rate may be offset.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- See `DeclinationRate` and Section 4.14

property `Telescope.`**`CanSetGuideRates`**: **boolean**

Returns The mount can adjust pulse guide rates.

Return type boolean

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `PulseGuide`.

property `Telescope.`**`CanSetPark`**: **boolean**

Returns The position of the mount's Parked state can be set.

Return type boolean

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

- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `SetPark()`

property `Telescope.`**CanSetPierSide: boolean**

Returns The mount can be force-flipped via setting `SideOfPier`

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `SideOfPier`.
- This applies to both German and simple/fork mounts. See Section 4.10. On a fork mount, if the imaging payload cannot pass through the fork, this must return False. Circumpolar locations "under" the pole require either the fork to roll over *more* than 180 degrees, or the imaging payload to pass through the fork. This is the equivalent of flipping on a German equatorial mount.

property `Telescope.`**CanSetRightAscensionRate: boolean**

Returns The Right Ascension tracking rate may be offset.

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately

diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `RightAscensionRate` and Section 4.14.

property `Telescope.`**`CanSetTracking`**: **boolean**

Returns The mount's sidereal tracking may be turned on and off

Return type boolean

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `Tracking`
- While tracking at sidereal rate, a mount is holding its `Telescope.RightAscension` and `Telescope.Declination` *constant*. The mount allows the eye camera to stare at a fixed point on the celestial sphere.

property `Telescope.`**`CanSlew`**: **boolean**

Returns The mount can synchronously slew to equatorial coordinates.

Return type boolean

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

See `SlewToCoordinates()`, `SlewToCoordinatesAsync()`, `SlewToTarget()`, and `SlewToTargetAsync()`

Attention!

Synchronous methods are deprecated in this version (V4) of ITelescope and Clients should not use them. ASCOM COM Driver authors however must implement synchronous methods, if the mount can slew, to ensure backward compatibility. See Section 4.8.

property `Telescope.`**`CanSlewAltAz`**: boolean

Returns The mount can synchronously slew to alt/az coordinates.

Return type boolean

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

See `SlewToAltAz()` and `SlewToAltAzAsync()`

Attention!

Synchronous methods are deprecated in this version (V4) of ITelescope and Clients should not use them. ASCOM COM Driver authors however must implement synchronous methods, if the mount can slew, to ensure backward compatibility. See Section 4.8.

property `Telescope.`**`CanSlewAltAzAsync`**: boolean

Returns The mount can asynchronously slew to alt/az coordinates.

Return type boolean

Raises

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

detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `SlewToAltAzAsync()`
- Clients should always use asynchronous slewing if available (`CanSlewAltAzAsync = True`) because synchronous methods are deprecated in this version (V4) of ITelescope, and will not be implemented at all by Alpaca devices.
- If the mount can slew, driver authors must implement asynchronous slewing.

property `Telescope.`**`CanSlewAsync`**: boolean

Returns The mount can asynchronously slew to equatorial coordinates.

Return type boolean

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `SlewToCoordinatesAsync()` and `SlewToTargetAsync()`
- Clients should always use asynchronous slewing if available (`CanSlewAltAzAsync = True`) because synchronous methods are deprecated in this version (V4) of ITelescope, and will not be implemented at all by Alpaca devices.
- If the mount can slew, driver authors must implement asynchronous slewing.

property `Telescope.`**`CanSync`**: boolean

Returns The mount can be synchronized to equatorial coordinates.

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `SyncToCoordinates`.

property `Telescope.`**`CanSyncAltAz`**: **boolean**

Returns The mount can be synchronized to alt/az coordinates.

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `SyncToAltAz()`.

property `Telescope.`**`CanUnpark`**: **boolean**

Returns The mount can be unparked

Return type boolean

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `Unpark()` and `Park()`.

property `Telescope`**.Connected: boolean**

Changed in version 4: Writing to change connection state superseded by asynchronous `Connect()`, `Disconnect()`, and `Connecting`.
(Read/Write) Retrieve or set the connected state of the device. **Writing is deprecated**, use the newer `Connect()` and `Disconnect()` methods, and the newer `Connecting` property. See remarks below.
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.

Returns True if connected to the hardware, else false.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Deprecation Notice

Property-write is deprecated as of Telescope V4. Starting with Platform 7 and the interface revisions contained therein, writing to Connected is discouraged. To connect and disconnect, use the newer non-blocking `Connect()` and `Disconnect()` methods, with the new `Connecting` property serving as the completion property.

Attention!

Must be implemented

Note

- Do not use a `NotConnectedException` here, that exception is for use in other methods that require a connection in order to succeed.
- 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.

property `Telescope.Connecting`: Boolean

Added in version 4: Preferred asynchronous connection mechanic. See notes below.

Returns True while the device is undertaking an asynchronous connect or disconnect operation.

Return type boolean

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This is the correct property for determining when the non-blocking methods `Connect()` or `Disconnect()` have completed. Completion is when `Connecting` becomes False after calling either of these methods.
- New in ITelescope V4

property `Telescope.Declination`: float

Returns The mount's current Declination (degrees, see Notes)

Return type float

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Attention!

This property must be implemented.

Note

- Declination must be in the equinox given by the current value of `EquatorialSystem`. See Section 4.12

property `Telescope.DeclinationRate: float`

Changed in version 4: Formalized to clarify that these rates apply only when the mount is tracking at sidereal rate.
(Read/Write) Read or set a secular rate of change to the mount's `Declination` in arc seconds per SI second. See Notes and Section 4.14

Returns The current declination rate offset (arc seconds per SI second).

Return type float

Raises

- **`PropertyNotImplementedException`** – If `CanSetDeclinationRate` is False yet an attempt is made to write to this property.
- **`InvalidOperationException`** – If `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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

`DeclinationRate` *read* must be implemented, *write* must be implemented when `CanSetDeclinationRate` is True.

Note

- DeclinationRate is an offset from 0 (no change in declination), given in arc seconds per UTC (clock) 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.
- If offset tracking is in effect (non-zero), and a slew is initiated, the mount must continue to update the slew destination coordinates at the given offset rate.
- Reading this property must return a value of zero if `TrackingRate` is not `driveSidereal`.
- See Section 4.14

property `Telescope.`**`Description`**: String

Returns Description of the **device** such as manufacturer and model number. Any ASCII characters may be used.

Return type string

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

- This describes the *device*, not the driver. See the `DriverInfo` property for information on the ASCOM driver.
- The description length must 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.

property `Telescope.`**`DeviceState`**: List[StateValue]

Returns List of `StateValue` objects representing the operational properties of this device. See Section 4.2.

Return type List[`StateValue`]

This device must return the following operational properties if they are known:

- `Altitude`
- `AtHome`
- `AtPark`
- `Azimuth`
- `Declination`
- `IsPulseGuiding`
- `RightAscension`
- `SideOfPier`
- `SiderealTime`
- `Slewing`
- `Tracking`
- `UTCDate`
- Section 4.3

Note

- For more info see Section 4.2.
- Available only for the Telescope Interface Version 4 and later.

property `Telescope.`**`DoesRefraction`**: **boolean**

(Read/Write) The mount applies atmospheric refraction corrections

Returns True if the mount applies atmospheric refraction corrections internally, else False.

Return type boolean

Raises

- **`PropertyNotImplementedException`** – If 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

`DoesRefraction` *read* must be implemented.

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) must 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 False, and the mount (already) does not do refraction, no exception must be raised.

property `Telescope.DriverInfo`: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

This string may contain line endings and may be hundreds to thousands of characters long. It is intended to display detailed information on the ASCOM driver, including version and copyright data. See the `Description` property for information on the device itself. To get the driver version in a parse-able string, use the `DriverVersion` property.

property `Telescope.DriverVersion`: String

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

Return type string

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

Note

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

property `Telescope.`**`EquatorialSystem`**: **enum Telescope.EquatorialCoordinateType**

Returns The current equatorial coordinate system used by the mount

Return type `EquatorialCoordinateType`

Raises
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

Note

- See `EquatorialCoordinateType`
- Most mounts use topocentric coordinates. Some high-end research mounts use J2000 coordinates.

property `Telescope.`**`FocalLength`**: **float**

Returns The telescope's focal length in meters.

Return type float

Raises
- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Telescope.`**`GuideRateDeclination`**: **float**

(Read/Write) The current rate of change of Declination (deg/sec) for guiding, typi-

cally via `PulseGuide`. See Notes.

Returns The current declination guide rate (deg/sec).

Return type float

Raises

- **`InvalidValueException`** – If an invalid guide rate is set
- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

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 must set the other to the same value.
- This value must be set to a default upon startup.

property `Telescope.`**`GuideRateRightAscension`**: float

(Read/Write) The current rate of change of Right Ascension (**deg/sec**) for guiding, typically via `PulseGuide`. See Notes.

Raises

- **`InvalidValueException`** – If an invalid guide rate is set
- **`PropertyNotImplementedException`** – If the 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

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 must set the other to the same value.
- This value is in degrees per second, not in hours per second.
- This value must be set to a default upon startup.

property `Telescope.`**`InterfaceVersion`**: Short

Returns The ASCOM Device *interface definition* version that this device

supports. Should return 4 for this interface version.

Return type short

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

- This is a single "short" integer indicating the version of this specific ASCOM universal interface definition. For example, for ICameraV4, this must be 4. It should not be confused with the DriverVersion property, which is the major.minor version of the driver for this device.
- Clients can detect legacy V1 drivers by trying to read this property. If the driver raises an error, it is a V1 driver. V1 did not specify this property. A driver may also return a value of 1. In other words, a raised error or a return value of 1 indicates that the driver is a V1 driver.

property Telescope.**IsPulseGuiding: boolean**

Returns True if the mount is currently executing a PulseGuide() command.

Return type boolean

Raises

- **PropertyNotImplementedException** – If the property is not implemented (CanPulseGuide 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- Use this property to determine when a (non-blocking) pulse guide command has completed. See Notes and Section 4.1.
- 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 Section 4.1

property Telescope.**Name: String**

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

Return type string

Raises **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a PropertyNotImplementedException

Note

The Description property is used to return info about the *device* rather than the *driver*.

property Telescope.RightAscension: float

Returns The mount's current right ascension (hours) in the current EquatorialSystem. See Section 4.12.

Return type float

Raises

- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented.

property Telescope.RightAscensionRate: float

Changed in version 4: Formalized to clarify that these rates apply only when the mount is tracking at sidereal rate.
(Read/Write) Read or set a secular rate of change to the mount's RightAscension (seconds of RA per *sidereal* second). See Notes and Section 4.14

Returns The current right ascension rate offset (seconds of RA per sidereal second).

Return type float

Raises

- **PropertyNotImplementedException** – If CanSetRightAscensionRate is False yet an attempt is made to write to this property.
- **InvalidOperationException** – If 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

`RightAscensionRate` *read* must be implemented.

Note

- To convert a given rate in (the more common) 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 property. See Section 4.14
- 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 must continue to update the slew destination coordinates at the given offset rate.
- Reading this property must return a value of zero if `TrackingRate` is not `driveSidereal`.
- Use the `Tracking` property to stop and start sidereal tracking.

property `Telescope.SideOfPier`: enum PierSide

(Read/Write) Start a change of, or return, the mount's pointing state. Returns a value of type:attr:*Telescope.PierSide*. See Section 4.10

Non-blocking write: 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 Section 4.1

Returns The current mount pointing state.

Return type `Telescope.PierSide`

Raises

- **PropertyNotImplementedException** – 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).
- **InvalidValueException** – If an invalid `Telescope.PierSide` value is set
- **NotConnectedException** – If the device is not connected
- **DriverException** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Note

- **Asynchronous** (non-blocking) if writing `SideOfPier` to force a pointing state change (e.g. forced 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 Section 4.1

Note

- Please note that "SideofPier" is a misnomer and that this method actually refers to the mount's pointing state. For German Equatorial mounts there is a complex relationship between pointing state and the physical side of the pier on which the mount resides.
- **Example:** Suppose the mount is tracking on the east side of the pier, counterweights down, observing a target on the celestial equator at hour angle +3.0.Now suppose that the observer wishes to observe a new target at hour angle -9.0. All the mount needs to do is to rotate the declination axis, through the celestial pole where the hour angle will change from +3.0 to -9.0, and keep going until it gets to the required declination at hour angle -9.0. Other than tracking, the RA axis has not moved.
  In this example the mount is still physically on the east side of the pier but the pointing state will have changed when the declination axis moved through the celestial pole.

Note

Please note that the PierSide enum is named PointingState in the ASCOM Library. See the Library Documentation

Important

Please see Section 4.10.

property `Telescope.SiderealTime`: **float**

Returns Local apparent sidereal time (hours)

Return type float

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

This property must be implemented

Note

- Local Apparent Sidereal Time is the sidereal time used for pointing mounts, and thus must be calculated from the Greenwich Mean Sidereal time, longitude, nutation in longitude and True ecliptic obliquity.
- It is required for a mount to calculate this from its source of `UTCDate` and `SiteLongitude`.

Important

See `UTCDate` for vital information on mount time sources and operating modes.

property `Telescope.`**`SiteElevation`**: float

(Read/Write) The observing site's elevation (meters) above mean sea level. See Section 4.9

Returns The elevation of the observing site (meters)

Return type float

Raises

- **`PropertyNotImplementedException`** – If the property is not implemented at all, or if *writing* to the property is not implemented. See Notes.
- **`InvalidValueException`** – If the given value is outside the range -300 through 10000 meters.
- **`InvalidOperationException`** – When `SiteElevation` is read and the mount cannot provide this property itself and a value has not yet been established by writing to the property 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

See Section 4.9

property `Telescope.`**`SiteLatitude`**: float

(Read/Write) The latitude (degrees) of the observing site. See Section 4.9

Returns The latitude of the observing site (degrees)

Return type float

Raises

- **PropertyNotImplementedException** – If the property is not implemented at all, or if *writing* to the property is not implemented. See Notes.
- **InvalidValueException** – If the given value is outside the range -90 through +90 degrees latitude.
- **InvalidOperationException** – When `SiteLatitude` is read and the mount cannot provide this property itself and a value has not yet been established by writing to the property 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

See Section 4.9

property `Telescope.`**SiteLongitude: float**

(Read/Write) The longitude (degrees, positive east) of the observing site. See Section 4.9

Returns The longitude of the observing site (degrees)

Return type float

Raises

- **PropertyNotImplementedException** – If the property is not implemented at all, or if *writing* to the property is not implemented. See Notes.
- **InvalidValueException** – If the given value is outside the range -90 through +90 degrees latitude.
- **InvalidOperationException** – When `SiteLongitude` is read and the mount cannot provide this property itself and a value has not yet been established by writing to the property 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

See Section 4.9

Attention!

West longitude is negative.

property `Telescope.SlewSettleTime`**: int**

(Read/Write) The post-slew settling time (seconds).
Artificially lengthens all slewing operations, delaying setting `Slewing` to False even though the slew actually completes. Useful for mounts or buildings that require additional mechanical settling time after a slew to stabilize (pier wobble, etc.).

Returns The post-slew settling time (seconds)

Return type int

Raises

- **`PropertyNotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is negative or preposterously high (driver dependent).
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

property `Telescope.Slewing`**: boolean**

Returns True if the mount is in motion resulting from a slew, parking, find-home, or a move-axis. See Section 4.1

Return type boolean

Raises

- **`PropertyNotImplementedException`** – If the property is not implemented (the mount has no slewing capability)
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- This is the correct property to use to determine *successful* completion of a (non-blocking) `SlewToCoordinatesAsync()`, `SlewToTargetAsync()`. `SlewToCoordinatesAsync()`, `Park()`, `FindHome()`, or by writing to `SideOfPier` to force a flip. See Section 4.1
- `Slewing` must be `True` immediately upon returning from any of these calls, and must remain `True` until *successful* completion, at which time Slewing must become `False`.
- `Slewing` must also be true during any `MoveAxis()` operations.
- `Slewing` must not be True during `PulseGuide()` operations or application of `RightAscensionRate` or `DeclinationRate` offsets.
- You might see `Slewing = False` on returning from a slew or `MoveAxis()` 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*.

property `Telescope.SupportedActions`: COM: ArrayList of String elements, Alpaca: Array of String

Added in version 3: To replace deprecated `CommandBlind`, `CommandBool`, and `CommandString` with a more flexible extension mechanic. See Notes below.
Returns the list of custom action names supported by this driver, to be used with `Action()`,

Returns The list of custom action names supported by this driver

Return type COM: ArrayList of String elements, Alpaca: Array of String

Raises **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Must be implemented, must not throw a `PropertyNotImplementedException`

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 custom action names exactly, without additional descriptive text. However, returned names may use any casing because the `ActionName` parameter of `Action()` is case insensitive.

property `Telescope.`**`TargetDeclination`**: **float**

(Read/Write) Set or return the declination (degrees, positive North) for the target of an equatorial slew or sync operation. See Notes and Section 4.12.

Returns The target declination (degrees)

Return type float

Raises

- **`PropertyNotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is outside the range -90 through 90 degrees.
- **`InvalidOperationException`** – If the value is read before being set for the first time. See Notes.
- **`ParkedException`** – If the mount is parked
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- This is a pre-set target coordinate for `SlewToTargetAsync()` and `SyncToTarget()`.
- This is set by a call to `SlewToCoordinatesAsync()` from the `Declination` parameter of the call.
- Target coordinates are for the current `EquatorialSystem`. See Section 4.12.

property `Telescope.TargetRightAscension: float`

(Read/Write) Set or return the right ascension (hours, positive North) for the target of an equatorial slew or sync operation. See Notes and Section 4.12.

Returns The target right ascension (hours)

Return type float

Raises

- **`PropertyNotImplementedException`** – If the property is not implemented
- **`InvalidValueException`** – If the given value is outside the range 0 to 24 hours.
- **`InvalidOperationException`** – If the value is read before being set for the first time. See Notes.
- **`ParkedException`** – If the mount is parked
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Note

- This is a pre-set target coordinate for `SlewToTargetAsync()` and `SyncToTarget()`.
- This is set by a call to `SlewToCoordinatesAsync()` from the `RightAscension` parameter of the call.
- Target coordinates are for the current `EquatorialSystem`. See Section 4.12.

property `Telescope.Tracking: boolean`

(Read/Write) The on/off state of the mount's sidereal tracking drive. See Notes.

Returns True if tracking is enabled, else False

Return type boolean

Raises

- **`PropertyNotImplementedException`** – If *writing* to the property (tracking control) is not implemented (if `CanSetTracking` is False)
- **`NotConnectedException`** – If the device is not connected
- **`ParkedException`** – When Tracking is set True and the telescope is parked (`AtPark` is True). Introduced in ITelescopeV4
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Attention!

Reading must be implemented and must not throw a `PropertyNotImplementedException`.

Note

- When tracking is turned on, the mount must use the last selected `TrackingRate`.
- While tracking at sidereal rate, a mount is holding its `Telescope.RightAscension` and `Telescope.Declination` *constant* allows the eye or camera to stare at a fixed point on the celestial sphere.

property `Telescope.`**`TrackingRate`**: enum DriveRates

(Read/Write) The current (sidereal) tracking rate of the mount, from `Telescope.DriveRates`. See Notes.

Returns The current tracking rate

Return type `DriveRates`

Raises

- **`InvalidValueException`** – If the value being written is not one of the `DriveRates` or if the requested rate is not supported by the mount (not all are).
- **`PropertyNotImplementedException`** – If *writing* to the property (changing tracking rate) is not implemented at all.
- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Reading the tracking rate must be supported.

property `Telescope.`**`TrackingRates`**: List[DriveRates]

Returns A list of supported drive rates.

Return type List[`DriveRates`]

Raises

- **`NotConnectedException`** – If the device is not connected
- **`DriverException`** – An error occurred that is not described by one of the more specific ASCOM exceptions. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communica-

tion errors.

Note

- At a minimum, this list must contain an item for `driveSidereal`

property `Telescope`.**`UTCDate`: DateTime**

(Read/Write) The UTC date/time of the mount's time source. See Section 4.9

Returns The current UTC date/time of the mount's time source.

Return type DateTime

Raises

- **`InvalidValueException`** – If an illegal DateTime value is written to the property.
- **`InvalidOperationException`** – When UTCDate is read and the mount cannot provide this property itself and a value has not yet been established by writing to the property.
- **`PropertyNotImplementedException`** – If *writing* to the property (changing the mount's UTC date-time) 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. Include sufficient detail in the message text to enable the issue to be accurately diagnosed by someone other than yourself. Includes communication errors.

Attention!

Reading this property must be implemented.

Note

- See Section 4.9
- *Alpaca Note:* The DateTime value must be in ISO 8601 format with 'Z' suffix to indicate UTC time.

5.12.3 Enumerated Constants

`Telescope`.**`AlignmentModes`: Integer**

The alignment mode (geometry) of the mount. See `AlignmentMode`.

Symbol	Val	Description
`algAltAz`	0	Altitude-Azimuth type mount
`algPolar`	1	Polar (equatorial) mount other than German equatorial
`algGermanPolar`	2	German equatorial type mount

`Telescope.DriveRates`: Integer

Well-known mount tracking rates. See `TrackingRate`.

Symbol	Val	Description
`driveSidereal`	0	Sidereal tracking rate (15.041 arcseconds per second)
`driveLunar`	1	Lunar tracking rate (14.685 arcseconds per second)
`driveSolar`	2	Solar tracking rate (15.0 arcseconds per second)
`driveKing`	3	King tracking rate (15.0369 arcseconds per second)

`Telescope.EquatorialCoordinateType`: Integer

Equatorial coordinate systems used by mounts. See Section 4.12

Symbol	Val	Description
`equOther`	0	Custom or unknown equinox and/or reference frame.
`equTopocentric`	1	Topocentric coordinates.
`equJ2000`	2	J2000 equator/equinox
`equJ2050`	3	J2050 equator/equinox
`equB1950`	4	B1950 equinox, FK4 reference frame

`Telescope.GuideDirections`: Integer

The direction in which a `PulseGuide()` guide-rate motion is to be made. These are *not* mechanical axes, these are directions in the equatorial coordinate system. See the Note below and the Section 4.15

Symbol	Val	Description
`guideNorth`	0	North (+ declination)
`guideSouth`	1	South (- declination)
`guideEast`	2	East (+ right ascension)
`guideWest`	3	West (- right ascension)

Note

The North/South directions are references to *equatorial* coordinates and must be independent of the pointing state (flip state) of the mount. See Section 4.15

`Telescope.PierSide`: Integer

The pointing states of mounts. See Section 4.10

Symbol	Val	Description
`pierEast`	0	Normal pointing state
`pierWest`	1	Through the pole pointing state
`pierUnknown`	-1	Unknown or indeterminate

Note

Please note that the PierSide enum is named PointingState in the ASCOM Library. See the Library Documentation for further information.

`Telescope.`**`TelescopeAxes`**`: Integer`

These are the *mechanical* axes of the mount. See `MoveAxis()` and Section 4.11. The direction of rotation (plus or minus) is left undefined and dependent on the mount's mechanical construction. See Notes.

Symbol	Val	Description
`axisPrimary`	0	Primary mechanical axis
`axisSecondary`	1	Secondary mechanical axis
`axisTertiary`	2	Tertiary mechanical axis

Note

- The meaning of primary, secondary, and tertiary axis varies with the mount mechanical geometry. These are not equatorial coordinate axes, they are *mechanical* axes.
- Typically, Primary may be the right ascension or the azimuth mechanical axis
- Typically Secondary may be the declination or the altitude mechanical axis
- Tertiary may refer to rotation about the optical axis or to some other axis for special geometries.
- The meaning of positive vs negative values as applies to rotation directions about the axes is purposely left undefined. App developers need to provide adaptation to various mount geometries and control systems and their rotation directions. See Section 4.11.



5.13 Exception Classes

The ASCOM interfaces share a set of standard exception classes, including one, `DriverException`, which may be used flexibly by a driver to indicate various problems, each with a custom driver error code and message.

Note

The exception codes shown here are generic values used by Alpaca or any other environment. For classic ASCOM COM, in the Windows environment, these (16-bit) codes are logically 'or'ed with 0x80040000 (Windows generic user exception). So, for example, the value 0x40B would appear in the COM context as 0x8004040B.

Bases: `ASCOM.DeviceInterface`

exception `ActionNotImplementedException` (*message: str*)

Exception thrown by a driver when it receives an unknown command through the Action method.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

Numeric value: 0x040C (1036)

exception `DriverException` (*number: int*, *message: str*)

Generic driver exception. See note below.

Parameters

- **`number`** (*`int`*) – The error number 0x500-0xFFF. See the info at the top of this page for specifics in the Windows COM environment. The driver may choose this number for its own purposes.
- **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

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 exceptions are already defined.

exception `InvalidOperationException` (*message: str*)

This exception should be thrown by the driver to reject a command from the client.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

Numeric value: 0x40B (1035)

exception `InvalidValueException` (*message: str*)

Exception to report an invalid value supplied to a driver.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction such as the legal range for the value.

Numeric value: 0x401 (1025)

exception `MethodNotImplementedException` (*message: str*)

All methods defined by the relevant ASCOM standard interface must exist in each driver. However, those methods do not all have to be *implemented*. The minimum requirement for each defined method is to throw `MethodNotImplementedException`.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction. **At a minimum it must contain the method name.**

Numeric value: 0x400 (1024)

> **Note**
>
> For historical reasons, this exception shares the same numeric code as `PropertyNotImplementedException`.

exception `NotConnectedException` (*message: str*)

This exception should be thrown when an operation is attempted that requires communication with the device, but the device is disconnected.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

Numeric value: 0x407 (1031)

This refers to the driver not being connected to the device. It is not for network outages or bad URLs.

exception `OperationCancelledException` (*message: str*)

This exception should be thrown to indicate that an (asynchronous) in-progress operation has been cancelled.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

Numeric value: 0x40E (1038)

exception `ParkedException` (*message: str*)

This exception should be thrown to indicate that movement (or other invalid operation) was attempted while the device was in a parked state.

Numeric value: 0x408 (1032)

exception `PropertyNotImplementedException` (*message: str*)

All properties defined by the relevant ASCOM standard interface must exist in each driver. However, those properties do not all have to be *implemented*. The minimum requirement for each defined property is to throw `PropertyNotImplementedException`.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction. **At a minimum it must**

contain the property name.

Numeric value: 0x400 (1024)

> **Note**
>
> For historical reasons, this exception shares the same numeric code as `MethodNotImplementedException`.

exception `SlavedException` (*message: str*)

This exception should be used to indicate that movement (or other invalid operation) was attempted while the device was in slaved mode. This applies primarily to Dome drivers.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

Numeric value: 0x409 (1033)

exception `ValueNotSetException` (*message: str*)

Exception to report that no value has yet been set for this property.

Parameters **`message`** (*`str`*) – The textual error message, which should always be informative and useful to help identify and solve the problem. It may even contain suggestions for correction.

Numeric value: 0x402 (1026)

A

AbortExposure() (Camera method), 29
AbortSlew() (Dome method), 99
AbortSlew() (Telescope method), 227
Absolute (Focuser property), 143
Action() (Camera method), 30
Action() (CoverCalibrator method), 81
Action() (Dome method), 100
Action() (FilterWheel method), 123
Action() (Focuser method), 136
Action() (ObservingConditions method), 154
Action() (Rotator method), 177
Action() (SafetyMonitor method), 195
Action() (Switch method), 207
Action() (Telescope method), 228
ActionNotImplementedException, 279
AlignmentMode (Telescope property), 246
AlignmentModes (Telescope attribute), 276
Altitude (Dome property), 110
Altitude (Telescope property), 247
ApertureArea (Telescope property), 247
ApertureDiameter (Telescope property), 248
AtHome (Dome property), 111
AtHome (Telescope property), 248
AtPark (Dome property), 112
AtPark (Telescope property), 248
AveragePeriod (ObservingConditions property), 162
AxisRates() (Telescope method), 229
Azimuth (Dome property), 112
Azimuth (Telescope property), 249

B

BayerOffsetX (Camera property), 38
BayerOffsetY (Camera property), 39
BinX (Camera property), 40
BinY (Camera property), 40
Brightness (CoverCalibrator property), 89

C

CalibratorChanging (CoverCalibrator property), 90
CalibratorOff() (CoverCalibrator method), 82
CalibratorOn() (CoverCalibrator method), 83
CalibratorState (CoverCalibrator property), 90
CalibratorStatus, 98
Camera (built-in class), 29
CameraState (Camera property), 41
CameraStates (Camera attribute), 80
CameraXSize (Camera property), 41
CameraYSize (Camera property), 41
CanAbortExposure (Camera property), 42
CanAsymmetricBin (Camera property), 42
CanAsync() (Switch method), 208
CancelAsync() (Switch method), 208
CanFastReadout (Camera property), 42
CanFindHome (Dome property), 113
CanFindHome (Telescope property), 250
CanGetCoolerPower (Camera property), 43
CanMoveAxis() (Telescope method), 230
CanPark (Dome property), 113
CanPark (Telescope property), 250
CanPulseGuide (Camera property), 43
CanPulseGuide (Telescope property), 250
CanReverse (Rotator property), 186
CanSetAltitude (Dome property), 114
CanSetAzimuth (Dome property), 114
CanSetCCDTemperature (Camera proper-

ty), 44
CanSetDeclinationRate (Telescope property), 251
CanSetGuideRates (Telescope property), 251
CanSetPark (Dome property), 114
CanSetPark (Telescope property), 251
CanSetPierSide (Telescope property), 252
CanSetRightAscensionRate (Telescope property), 252
CanSetShutter (Dome property), 115
CanSetTracking (Telescope property), 253
CanSlave (Dome property), 115
CanSlew (Telescope property), 253
CanSlewAltAz (Telescope property), 254
CanSlewAltAzAsync (Telescope property), 254
CanSlewAsync (Telescope property), 255
CanStopExposure (Camera property), 45
CanSync (Telescope property), 255
CanSyncAltAz (Telescope property), 256
CanSyncAzimuth (Dome property), 115
CanUnpark (Telescope property), 256
CanWrite() (Switch method), 209
CCDTemperature (Camera property), 45
CloseCover() (CoverCalibrator method), 83
CloseShutter() (Dome method), 101
CloudCover (ObservingConditions property), 163
CommandBlind() (Camera method), 31
CommandBlind() (CoverCalibrator method), 84
CommandBlind() (Dome method), 101
CommandBlind() (FilterWheel method), 124
CommandBlind() (Focuser method), 137
CommandBlind() (ObservingConditions method), 155
CommandBlind() (Rotator method), 178
CommandBlind() (SafetyMonitor method), 196
CommandBlind() (Switch method), 209
CommandBlind() (Telescope method), 230
CommandBool() (Camera method), 32
CommandBool() (CoverCalibrator method), 85
CommandBool() (Dome method), 102
CommandBool() (FilterWheel method), 125
CommandBool() (Focuser method), 138
CommandBool() (ObservingConditions method), 156
CommandBool() (Rotator method), 179
CommandBool() (SafetyMonitor method), 197
CommandBool() (Switch method), 210
CommandBool() (Telescope method), 231
CommandString() (Camera method), 33
CommandString() (CoverCalibrator method), 86
CommandString() (Dome method), 103
CommandString() (FilterWheel method), 126
CommandString() (Focuser method), 139
CommandString() (ObservingConditions method), 157
CommandString() (Rotator method), 180
CommandString() (SafetyMonitor method), 198
CommandString() (Switch method), 211
CommandString() (Telescope method), 232
Connect() (Camera method), 34
Connect() (CoverCalibrator method), 87
Connect() (Dome method), 104
Connect() (FilterWheel method), 127
Connect() (Focuser method), 140
Connect() (ObservingConditions method), 158
Connect() (Rotator method), 181
Connect() (SafetyMonitor method), 199
Connect() (Switch method), 212
Connect() (Telescope method), 233
Connected (Camera property), 45
Connected (CoverCalibrator property), 91
Connected (Dome property), 116
Connected (FilterWheel property), 128
Connected (Focuser property), 144
Connected (ObservingConditions property), 163
Connected (Rotator property), 187
Connected (SafetyMonitor property), 200
Connected (Switch property), 221
Connected (Telescope property), 257
Connecting (Camera property), 46
Connecting (CoverCalibrator property), 92
Connecting (Dome property), 117
Connecting (FilterWheel property), 129
Connecting (Focuser property), 145
Connecting (ObservingConditions property), 164
Connecting (Rotator property), 188

Connecting (SafetyMonitor property), 201
Connecting (Switch property), 222
Connecting (Telescope property), 258
CoolerOn (Camera property), 47
CoolerPower (Camera property), 47
CoverCalibrator (built-in class), 81
CoverMoving (CoverCalibrator property), 93
CoverState (CoverCalibrator property), 93
CoverStatus, 98

D

Declination (Telescope property), 258
DeclinationRate (Telescope property), 259
Description (Camera property), 48
Description (CoverCalibrator property), 94
Description (Dome property), 117
Description (FilterWheel property), 130
Description (Focuser property), 146
Description (ObservingConditions property), 164
Description (Rotator property), 188
Description (SafetyMonitor property), 202
Description (Switch property), 223
Description (Telescope property), 260
DestinationSideOfPier() (Telescope method), 233
DeviceState (Camera property), 48
DeviceState (CoverCalibrator property), 95
DeviceState (Dome property), 118
DeviceState (FilterWheel property), 130
DeviceState (Focuser property), 146
DeviceState (ObservingConditions property), 165
DeviceState (Rotator property), 189
DeviceState (SafetyMonitor property), 202
DeviceState (Switch property), 223
DeviceState (Telescope property), 260
DewPoint (ObservingConditions property), 166
Disconnect() (Camera method), 34
Disconnect() (CoverCalibrator method), 87
Disconnect() (Dome method), 104
Disconnect() (FilterWheel method), 127
Disconnect() (Focuser method), 140
Disconnect() (ObservingConditions method), 158
Disconnect() (Rotator method), 181
Disconnect() (SafetyMonitor method), 199
Disconnect() (Switch method), 213
Disconnect() (Telescope method), 234
DoesRefraction (Telescope property), 261
Dome (built-in class), 99
DriveRates (Telescope attribute), 277
DriverException, 279
DriverInfo (Camera property), 49
DriverInfo (CoverCalibrator property), 95
DriverInfo (Dome property), 119
DriverInfo (FilterWheel property), 131
DriverInfo (Focuser property), 146
DriverInfo (ObservingConditions property), 166
DriverInfo (Rotator property), 189
DriverInfo (SafetyMonitor property), 203
DriverInfo (Switch property), 224
DriverInfo (Telescope property), 262
DriverVersion (Camera property), 49
DriverVersion (CoverCalibrator property), 95
DriverVersion (Dome property), 119
DriverVersion (FilterWheel property), 131
DriverVersion (Focuser property), 147
DriverVersion (ObservingConditions property), 167
DriverVersion (Rotator property), 190
DriverVersion (SafetyMonitor property), 203
DriverVersion (Switch property), 224
DriverVersion (Telescope property), 262

E

ElectronsPerADU (Camera property), 50
EquatorialCoordinateType (Telescope attribute), 277
EquatorialSystem (Telescope property), 263
ExposureMax (Camera property), 50
ExposureMin (Camera property), 51
ExposureResolution (Camera property), 51

F

FastReadout (Camera property), 52
FilterWheel (built-in class), 123
FindHome() (Dome method), 105
FindHome() (Telescope method), 234
FocalLength (Telescope property), 263
Focuser (built-in class), 136
FocusOffsets (FilterWheel property), 132
FullWellCapacity (Camera property), 53

G

Gain (Camera property), 53
GainMax (Camera property), 55
GainMin (Camera property), 55
Gains (Camera property), 56
GetSwitch() (Switch method), 213
GetSwitchDescription() (Switch method), 214
GetSwitchName() (Switch method), 214
GetSwitchValue() (Switch method), 215
GuideDirections (Camera attribute), 80
GuideDirections (Telescope attribute), 277
GuideRateDeclination (Telescope property), 263
GuideRateRightAscension (Telescope property), 264

H

Halt() (Focuser method), 141
Halt() (Rotator method), 182
HaltCover() (CoverCalibrator method), 88
HasShutter (Camera property), 57
HeatSinkTemperature (Camera property), 57
Humidity (ObservingConditions property), 167

I

ImageArray (Camera property), 58
ImageArrayVariant (Camera property), 61
ImageReady (Camera property), 61
InterfaceVersion (Camera property), 62
InterfaceVersion (CoverCalibrator property), 96
InterfaceVersion (Dome property), 120
InterfaceVersion (FilterWheel property), 132
InterfaceVersion (Focuser property), 147
InterfaceVersion (ObservingConditions property), 168
InterfaceVersion (Rotator property), 190
InterfaceVersion (SafetyMonitor property), 204
InterfaceVersion (Switch property), 225
InterfaceVersion (Telescope property), 264
InvalidOperationException, 279
InvalidValueException, 279
IsMoving (Focuser property), 148
IsMoving (Rotator property), 191
IsPulseGuiding (Camera property), 62
IsPulseGuiding (Telescope property), 265
IsSafe (SafetyMonitor property), 204

L

LastExposureDuration (Camera property), 63
LastExposureStartTime (Camera property), 63
Link (Focuser property), 148

M

MaxADU (Camera property), 64
MaxBinX (Camera property), 64
MaxBinY (Camera property), 65
MaxBrightness (CoverCalibrator property), 96
MaxIncrement (Focuser property), 149
MaxStep (Focuser property), 150
MaxSwitch (Switch property), 225
MaxSwitchValue() (Switch method), 216
MechanicalPosition (Rotator property), 191
MethodNotImplementedException, 280
MinSwitchValue() (Switch method), 216
Move() (Focuser method), 141
Move() (Rotator method), 182
MoveAbsolute() (Rotator method), 183
MoveAxis() (Telescope method), 235
MoveMechanical() (Rotator method), 184

N

Name, 206
Name (Camera property), 65
Name (CoverCalibrator property), 97
Name (Dome property), 120
Name (FilterWheel property), 133
Name (Focuser property), 150
Name (ObservingConditions property), 168
Name (Rotator property), 192
Name (SafetyMonitor property), 204
Name (Switch property), 226
Name (Telescope property), 265
Names (FilterWheel property), 133
NotConnectedException, 280
NumX (Camera property), 66
NumY (Camera property), 66

O

ObservingConditions (built-in class), 154
Offset (Camera property), 67

OffsetMax (Camera property), 68
OffsetMin (Camera property), 69
Offsets (Camera property), 70
OpenCover() (CoverCalibrator method), 88
OpenShutter() (Dome method), 105
OperationCancelledException, 280

P

Park() (Dome method), 106
Park() (Telescope method), 236
ParkedException, 280
PercentCompleted (Camera property), 71
PierSide (Telescope attribute), 277
PixelSizeX (Camera property), 71
PixelSizeY (Camera property), 72
Position (FilterWheel property), 134
Position (Focuser property), 150
Position (Rotator property), 192
Pressure (ObservingConditions property), 169
PropertyNotImplementedException, 280
PulseGuide() (Camera method), 35
PulseGuide() (Telescope method), 237

R

RainRate (ObservingConditions property), 169
Rate (built-in class), 175
ReadoutMode (Camera property), 72
ReadoutModes (Camera property), 73
Refresh() (ObservingConditions method), 159
Reverse (Rotator property), 193
RightAscension (Telescope property), 266
RightAscensionRate (Telescope property), 266
Rotator (built-in class), 176

S

SafetyMonitor (built-in class), 195
SensorDescription() (ObservingConditions method), 159
SensorName (Camera property), 74
SensorType (Camera property), 75
SensorType (Camera attribute), 80
SetAsync() (Switch method), 217
SetAsyncValue() (Switch method), 218
SetCCDTemperature (Camera property), 76
SetPark() (Dome method), 107
SetPark() (Telescope method), 238
SetSwitch() (Switch method), 218
SetSwitchName() (Switch method), 219
SetSwitchValue() (Switch method), 219
SetupDialog() (Camera method), 36
SetupDialog() (CoverCalibrator method), 89
SetupDialog() (Dome method), 107
SetupDialog() (FilterWheel method), 128
SetupDialog() (Focuser method), 143
SetupDialog() (ObservingConditions method), 160
SetupDialog() (Rotator method), 186
SetupDialog() (SafetyMonitor method), 200
SetupDialog() (Switch method), 220
SetupDialog() (Telescope method), 238
ShutterState, 122
ShutterStatus (Dome property), 120
SideOfPier (Telescope property), 267
SiderealTime (Telescope property), 268
SiteElevation (Telescope property), 269
SiteLatitude (Telescope property), 269
SiteLongitude (Telescope property), 270
SkyBrightness (ObservingConditions property), 170
SkyQuality (ObservingConditions property), 171
SkyTemperature (ObservingConditions property), 172
Slaved (Dome property), 121
SlavedException, 281
Slewing (Dome property), 121
Slewing (Telescope property), 271
SlewSettleTime (Telescope property), 271
SlewToAltAz() (Telescope method), 239
SlewToAltAzAsync() (Telescope method), 240
SlewToAltitude() (Dome method), 107
SlewToAzimuth() (Dome method), 109
SlewToCoordinates() (Telescope method), 241
SlewToCoordinatesAsync() (Telescope method), 242
SlewToTarget() (Telescope method), 243
SlewToTargetAsync() (Telescope method), 243
StarFWHM (ObservingConditions property), 173
StartExposure() (Camera method), 36
StartX (Camera property), 77
StartY (Camera property), 78
StateChangeComplete() (Switch method),

220
StateValue (built-in class), 206
StepSize (Focuser property), 151
StepSize (Rotator property), 194
StopExposure() (Camera method), 38
SubExposureDuration (Camera property), 78
SupportedActions (Camera property), 79
SupportedActions (CoverCalibrator property), 97
SupportedActions (Dome property), 122
SupportedActions (FilterWheel property), 135
SupportedActions (Focuser property), 151
SupportedActions (ObservingConditions property), 173
SupportedActions (Rotator property), 194
SupportedActions (SafetyMonitor property), 205
SupportedActions (Switch property), 226
SupportedActions (Telescope property), 272
Switch (built-in class), 206
SwitchStep() (Switch method), 221
Sync() (Rotator method), 185
SyncToAltAz() (Telescope method), 244
SyncToAzimuth() (Dome method), 110
SyncToCoordinates() (Telescope method), 244
SyncToTarget() (Telescope method), 245

T

TargetDeclination (Telescope property), 273
TargetPosition (Rotator property), 195
TargetRightAscension (Telescope property), 274
Telescope (built-in class), 227
TelescopeAxes (Telescope attribute), 278
TempComp (Focuser property), 152
TempCompAvailable (Focuser property), 153
Temperature (Focuser property), 153
Temperature (ObservingConditions property), 174
TimeSinceLastUpdate() (ObservingConditions method), 161
Tracking (Telescope property), 274
TrackingRate (Telescope property), 275
TrackingRates (Telescope property), 275

U

Unpark() (Telescope method), 245
UTCDate (Telescope property), 276

V

Value, 206
ValueNotSetException, 281

W

WindDirection (ObservingConditions property), 174
WindGust (ObservingConditions property), 175
WindSpeed (ObservingConditions property), 175