ASCOM ASYNCHRONOUS INTERFACES

Proposal Version 1

Peter Simpson, Bob Denny, Daniel Van Noord

Contents

Interface Changes for Async Operation

1 Context

Growing use of the network-based Alpaca protocol has highlighted the synchronous behaviour of some interface members where it can take several seconds or minutes before a completion response is sent to the client.

This paper describes documentation and technical changes to the ASCOM interfaces to ensure that asynchronous operation is specified for all long-running operations. Wherever possible the behavioural requirements of existing interface members have been revised to minimise the number of new interface members required.

On slow or unreliable networks, Alpaca can be subject to latency and bandwidth issues when applications regularly poll multiple device operational state properties. To address these concerns a new DeviceState property will be added to every interface that returns all device operational properties in one call. Please see Appendix 1 for details of the device properties that will be returned by this call.

As a result of these changes all device interface version numbers will be increased by 1.

1.1 Definition of Long-running

For the purposes of this paper operations are considered to be long running if they are likely to take more than 1 second to complete at the device.

2 Methods common to all interfaces

2.1 Interface Changes

2.1.1 New Connect() and Disconnect() methods

The current Connected property is synchronous in operation and some hardware can take several seconds to initialise, leaving applications “hanging”, waiting for the property to return control to the application. Given the greater emphasis now placed on responsive UIs, we are introducing an asynchronous connection mechanic to every device interface.

The new mechanic is implemented with two new initiator methods that must return quickly having started the required operation:

```
public void Connect() { }
public void Disconnect() { }
```

In addition, we will add a new `Connecting` property that applications can poll to determine when the connection / disconnection operation has completed:

```
public bool Connecting { get; }
```

Introducing the `Connecting` property will ensure a clear separation between the roles of state value (`Connected`) and operation status (`Connecting`) and will ensure maximum backward compatibility with current applications.

The `Connected` property setter will be retained, and must be implemented, to ensure backward compatibility with current applications.

2.1.2 New DeviceState property

The `DeviceState` property will return all operational properties of the device in a single call to reduce latency and network traffic when clients poll for device state. Please see Appendix 1 for further rationale and implementation details.

```
public ArrayList DeviceState { get; }
```

The `ArrayList` type is used to ensure compatibility with COM clients and devices.

Each `DeviceState` element is a class that implements `IStateValue`, which defines two properties:

```
public string Name { get; }
public object Value { get; }
```

2.2 Documentation Changes

Action method: This will remain a synchronous method. However, documentation will be added to describe best practice for implementing long-running operations. i.e. to use a short lived initiator Action and a second Action that can be polled to determine the state of the long-running operation.

3 Camera Interface

3.1 Documentation Changes

StartExposure & StopExposure: These will be defined as asynchronous with `ImageReady` as the completion variable.

SetCCDTemperature: A note will be added that 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.

PulseGuide: This will be clearly defined as asynchronous using the existing `IsPulseGuiding` property as the completion variable.

ImageArray: This will remain synchronous but with an additional note saying that applications should be prepared for `ImageArray` to take some time to return when large images are being retrieved.

In addition, the word safearray will be deleted from the `ImageArray` title and a remark will be added that C++ driver developers should return a safearray.

A further remark will be added noting the improvement in Alpaca image retrieval times when using the ImageBytes mechanic.

ImageArrayVariant: The `ImageArrayVariant` property is a functional duplicate of the ImageArray property but returns data using the variant type, which requires more memory and processor resource than the Int32 type returned by `ImageArray`.

`ImageArrayVariant` was included in the ASCOM interface to support scripting languages that required data elements to be of variant type rather than as integer type. However, this restriction no longer applies and the `ImageArrayVariant` property is now redundant.

The ImageArrayVariant method will be retained in the Camera interface but will be marked as deprecated in favour of `ImageArray`.

A remark will be added saying that applications should be prepared for `ImageArrayVariant` to take some time to return when large images are being retrieved. In addition, the word safearray will be deleted from the ImageArray title and a remark will be added that C++ driver developers should return a safearray.

A further remark will be added noting the improvement in Alpaca image retrieval times when using the ImageBytes mechanic.

4 CoverCalibrator Interface

4.1 Interface Changes

Currently this 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, pollable boolean completion variables will be added for cover and calibrator operations that can return text error descriptions through exceptions / errors when necessary.

4.1.1 New CalibratorChanging completion property

We will add a new boolean completion variable `CalibratorChanging` that returns true while the calibrator is in the "not ready / changing" state during the `CalibratorOn` and `CalibratorOff` operations.

```
public bool CalibratorChanging { get; }
```

The `CalibratorStatus`.NotReady state documentation will be updated to add that it must be kept in sync with `CalibratorChanging`.

4.1.2 New CoverMoving completion property

Add a new boolean completion variable `CoverMoving` that returns true while the cover is in motion when the `OpenCover` or `CloseCover` operations are underway.

```
public bool CoverMoving { get; }
```

The `CoverStatus.Moving` state documentation will be updated to add that it must be kept in sync with `CoverMoving`.

4.2 Documentation Changes

CalibratorStatus and **CoverStatus**: New notes will say that these properties must not throw exceptions / return errors.

HaltCover: A note will be added saying that this is expected to be a short-lived synchronous call.

5 FilterWheel Interface

5.1 Documentation Changes

Position Set: The `Position` property setter will be explicitly defined as being asynchronous with the `Position` property getter as the operation's completion property.

6 Focuser Interface

6.1 Documentation changes

Move: `Move` will be defined as asynchronous with `IsMoving` as the operation's completion property.

Halt: A note will be added that `Halt` must be a short-lived synchronous call.

Link: The `Link` property will be clearly marked as deprecated.

7 ObservingConditions Interface

7.1 Documentation Changes

Refresh: A note will be added to specify that `Refresh` must be a short synchronous call that triggers a refresh and that it must not wait for long running processes to complete. It will be a client responsibility to poll `TimeSinceLastUpdate` to determine whether / when the data is refreshed.

8 Rotator Interface

8.1 Documentation Changes

Move, MoveAbsolute, MoveMechanical: These methods will be defined as asynchronous with `IsMoving` as their completion variable.

9 Telescope Interface

9.1 Documentation Changes

FindHome: `FindHome` will be clearly defined as asynchronous noting that some drivers have been asynchronous for a long time and that apps have always needed to support this behaviour.

These notes will be added to the `FindHome` definition:

- `Slewing` must be set true while finding home.
- `FindHome` and `AtHome` must throw not implemented exceptions if `CanFindHome` is false.

Park: `Park` will be clearly defined to be asynchronous noting that some drivers have behaved asynchronously for a long time and that apps have always needed to support this behaviour.

These notes will be added:

- `Slewing` must be set true while parking.
- `Park` and `AtPark` must throw not implemented exceptions if `CanPark` is false.

SideofPier Set: This is already defined as asynchronous and the description will be expanded to make this even more clear.

PulseGuide: This is already defined as asynchronous and the description will be expanded to make this even more clear.

SlewToTarget: This synchronous method will be deprecated in favour of its asynchronous counterpart.

SlewToCoordinates: This synchronous method will be deprecated in favour of its asynchronous counterpart.

SlewToAltAz: This synchronous method will be deprecated in favour of its asynchronous counterpart.

CanSlew and CanSlewAsync: These will be revised to deal with the deprecated synchronous methods and to note that they must now be tied together.

CanSlewAltAz and CanSlewAltAzAsync: These will be revised to deal with the deprecated synchronous methods and to note that they must now be tied together.

Unpark: `Unpark` will be clearly defined to be asynchronous noting that some drivers have operated asynchronously for a long time and that apps have always needed to support this behaviour.

These notes will be added:

- `Slewing` must be set true while unparking to act as the completion variable, even if the telescope is not moving.

10 Video Interface

By agreement with the original author this interface will be marked as deprecated.

10.1 Documentation Changes

StartRecording and StopRecording: Make clear that these are asynchronous using `CameraState` as the completion variable.

Appendix 1 - New DeviceState Property

1 Context

Many ASCOM properties provide information about current device state and fall into two categories:

- **Configuration information** – These are set prior to an operation commencing and stay fixed for the lifetime of an operation such as `Camera.BinX` and `Telescope.SiteLatitude`
- **Operational information** – These change while an operation is in progress such as `Telescope.RightAscension`, `Focuser.Position` and `ObservingConditions.WindGust`

The DeviceState property will return all the device's **operational** property values in a single call to reduce latency and network bandwidth. 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.

2 Use Cases

The `DeviceState` property is intended to improve ASCOM Interface support for two primary use cases:

- Status reporting in client user interfaces
- Progress monitoring for processes initiated by the client.

3 DeviceState Behaviour

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.

3.1 Devices

A device must return all operational values that it definitively knows but can omit entries where value are unknown. Devices must not throw exceptions / return errors when values are not known.

An empty list must be returned if no values are known.

3.2 Client Applications

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.

4 Time Stamps

An optional `string TimeStamp` property will be defined so that the device can record the time at which the state was measured, if known. The 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.

5 DeviceState Interface Definition

The COM and Alpaca interfaces are functionally equivalent and return an enumerable collection of `IStateValue` objects.

5.1 IStateValue Interface

The `IStateValue` interface is defined as follows:

```
public interface IStateValue
{
	string Name { get; }
	object Value { get; }
}
```

5.2 COM Interface Definition

The ASCOM COM driver interface definition is:

```
public ArrayList DeviceState { get; }
```

The **ArrayList** type is used to ensure compatibility with COM clients and devices and provides an enumerable list of `IStateValue` objects.

Each ArrayList element must be a COM registered class that implements the `IStateValue` interface and exposes two properties:

```
public string Name { get; }
public object Value { get; }
```

`IStateValue.Name` is the name of an operational property. The name is **case sensitive** and must match the property name's spelling and casing in the ASCOM interface specification.

The `IStateValue.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.

5.3 Alpaca Interface Definition

The Alpaca device response uses the standard Alpaca message structure consisting of `Value`, `ClientTransactionID`, `ServerTransactionID`, `ErrorNumber` and `ErrorMessage` keys. The content of the `Value` key is a JSON array of `IStateValue` objects with `Name` and `Value` keys defined identically to those in the COM interface.

Here is a formatted example of a Camera `DeviceState` Alpaca JSON response:

```
{
	"Value":[
			{"Name":"CameraState","Value":0},
			{"Name":"CCDTemperature","Value":10},
			{"Name":"CoolerPower","Value":0},
			{"Name":"HeatSinkTemperature","Value":10},
			{"Name":"ImageReady","Value":false},
			{"Name":"IsPulseGuiding","Value":false},
			{"Name":"PercentCompleted","Value":0},
			{"Name":"TimeStamp","Value":"2023-06-14T11:17:50.0Z"}
		],
	"ClientTransactionID":123,
	"ServerTransactionID":456,
	"ErrorNumber":0,
	"ErrorMessage":""
}
```

6 Operational Properties

The following properties will be returned by the `DeviceState` property:

Interface	DeviceState Name	Comment
ICamera	CameraState	
	CCDTemperature	
	CoolerPower	
	HeatSinkTemperature	
	ImageReady	
	IsPulseGuiding	
	PercentCompleted	
	TimeStamp	Time of status, if known, in ISO 8601 format.
ICoverCalibrator	CalibratorState	
	CoverState	
	CalibratorReady	
	CoverMoving	
	TimeStamp	Time of status, if known, in ISO 8601 format.
IDome	Altitude	
	AtHome	
	AtPark	
	Azimuth	
	ShutterStatus	
	Slewing	
	TimeStamp	Time of status, if known, in ISO 8601 format.
IFilterWheel	Position	
	TimeStamp	Time of status, if known, in ISO 8601 format.
IFocuser	IsMoving	
	Position	
	Temperature	
	TimeStamp	Time of status, if known, in ISO 8601 format.
IObservingConditions	CloudCover	
	DewPoint	
	Humidity	
	Pressure	
	RainRate	
	SkyBrightness	
	SkyQuality	
	SkyTemperature	
	StarFWHM	
	Temperature	
	WindDirection	
	WindGust	

	WindSpeed	
	TimeStamp	Time of status, if known, in ISO 8601 format.
IRotator	IsMoving	
	MechanicalPosition	
	Position	
	TimeStamp	Time of status, if known, in ISO 8601 format.
ISafetyMonitor	isSafe	
	TimeStamp	Time of status, if known, in ISO 8601 format.
ISwitch	GetSwitch0 GetSwitch1 GetSwitch2 … GetSwitchN GetSwitchValue0 GetSwitchValue1 GetSwitchValue2 … GetSwitchValueN	(Assumes that the number of available switches (N) has already been determined by the application)
	TimeStamp	Time of status, if known, in ISO 8601 format.
ITelescope	Altitude AtHome AtPark Azimuth Declination IsPulseGuiding RightAscension SideOfPier SiderealTime Slewing Tracking UTCDate	
	TimeStamp	Time of status, if known, in ISO 8601 format.
IVideo		Interface deprecated - no change

7 Client Toolkit Support

As a convenience for application developers all ASCOM client toolkit devices will provide an additional property that presents the device's DeviceState response as a class. The name of the additional property will follow the format: *{DeviceType}*`DeviceState` where *{DeviceType}* is Camera, Telescope etc. e.g. `CameraDeviceState` and `TelescopeDeviceState`.

The signature of these additional properties will be:

```
public {DeviceType}DeviceState {DeviceType}DeviceState { get; }
```

7.1 The DeviceState Class

Within the ASCOM device interfaces all operational information values are defined using value types, which do not allow an "unknown" state to be represented. To address this the *{DeviceType}*`DeviceState` classes will expose nullable value types so that the client can detect the "unknown" state in addition to the property's actual value if available.

7.1.1 Nullable Property Behaviour

Using the telescope device `AtPark` property as an example, if the device returns a value for the `AtPark` property:

- `TelescopeDeviceState.AtPark.HasValue` will return `true`.
- `TelescopeDeviceState.AtPark` will return the value.
- `TelescopeDeviceState.AtPark.Value` will return the value.

If the device does not return a value for the `AtPark` property:

- `TelescopeDeviceState.AtPark.HasValue` will return `false`.
- `TelescopeDeviceState.AtPark` will return `null`.
- `TelescopeDeviceState.AtPark.Value` will throw an exception.

7.2 DeviceState Property Operation

When the client gets the *{DeviceType}*`DeviceState` property the toolkit will:

- Create a response class with all property values set to null.
- Call the device's DeviceState property.
- Populate the response class's properties with the information returned by the device.

This approach ensures that any operational properties that the device does not return will have `null` values in the response class representing the "unknown" state.

Please note that the *{DeviceType}*`DeviceState` class instance data is immutable. To obtain updated information from the device read the toolkit *{DeviceType}*`DeviceState` property again.

7.3 Telescope Device State Example

This is the definition of the class returned by DriverAccess's `TelescopeDeviceState` property:

```
public class TelescopeDeviceState : ITelescopeDeviceState
{
    public TelescopeDeviceState() { } // Other initiators omitted for clarity

    public double? Altitude { get; private set; } = null;

    public bool? AtHome { get; private set; } = null;

    public bool? AtPark { get; private set; } = null;
```

```
    public double? Azimuth { get; private set; } = null;

    public double? Declination { get; private set; } = null;

    public bool? IsPulseGuiding { get; private set; } = null;

    public double? RightAscension { get; private set; } = null;

    public PierSide? SideOfPier { get; private set; } = null;

    public double? SiderealTime { get; private set; } = null;

    public bool? Slewing { get; private set; } = null;

    public bool? Tracking { get; private set; } = null;

    public DateTime? UTCDate { get; private set; } = null;

    public DateTime? TimeStamp { get; private set; } = null;
}
```