ASCOM Device Hub User and Technical Information Guide

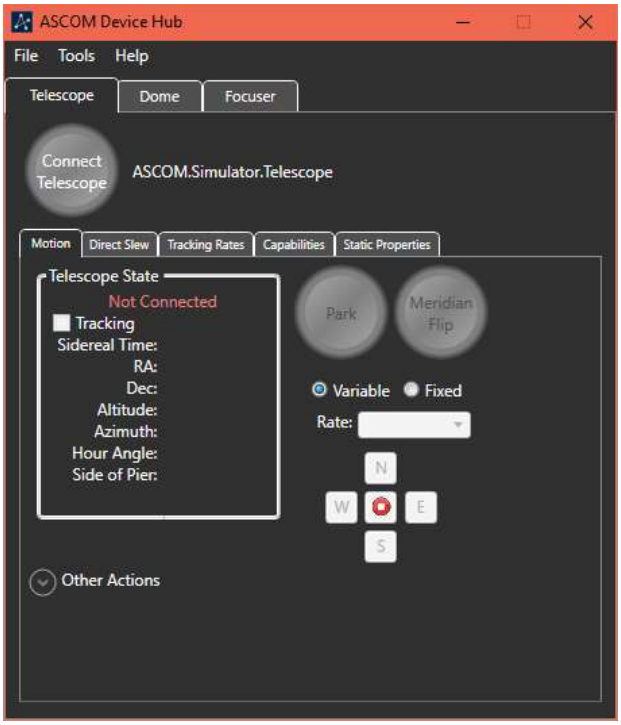
For as long as I have been using ASCOM, POTH has been installed as part of the Platform. It was originally written in 2003 and in 2018 is showing its age. The development tool and the language that it was written in were retired by Microsoft about 10 years ago. It is a testament to the original author and subsequent contributors that POTH is still useful today. However, I decided to dust off my software development skills to write a replacement for the venerable tool. For lack of a better name I am calling it the ASCOM Device Hub. It supports control of telescopes, domes, and focusers, as POTH does, but is written in C# using Visual Studio 2019.

The internal architecture and the screen design of the Device Hub is very different from POTH. It uses the .NET Framework, as do most ASCOM applications that are written today, and the user interface design utilizes Microsoft's Windows Presentation Foundation (WPF). The use of WPF allows a lot of flexibility to implement clean, powerful, and intuitive user controls and forms. As a consequence, the look of the Device Hub may at first be a bit unfamiliar. However I hope that it allows users to quickly become productive with it.

Launching the Device Hub as an ASCOM Client

The Device Hub can simply be used to simultaneously control a telescope, a dome, and a focuser. In addition, the dome can be slaved to the telescope to keep the dome's window-on-the-sky centered above the end of the telescope and following the telescope as it tracks and slews across the sky.

When first launched, the Device Hub main window appears like the picture at right:



You can see the menu bar with its File, Tools and Help options across the top of the main window, just below the title bar. The File menu has an Exit item for shutting down the Device Hub. The Tools menu has a Setup item for selecting and configuring the devices and a View Activity Log item for making the Device Hub Activity Log visible. The Help menu item displays this document.

Below the menu bar are the device selection tabs. The Telescope tab is shown at right. Clicking on

Dome or Focuser will switch to the view for that device.

When the Device Hub is not visible on the screen, say when it is minimized, it resides in the System Tray. You can display the main window by clicking on the ASCOM icon in the Tray.

Application and Device Settings

One of the first tasks will be to customize the application and device settings for the Device Hub. To launch the Setup dialog, select Setup from the Tools menu.

The Setup dialog window will appear. The selected tab will be the same as the active device when the Setup dialog was launched, unless that device is currently connected. So if the Telescope device tab was the selected tab in the main window, but no telescope was connected, the Telescope Setup tab will be active when the Setup dialog is first launched.

Application Configuration

Most of the configuration settings are in the device setup tabs, but there are a few that relate to the entire application. They are displayed on the Device Hub Setup tab.

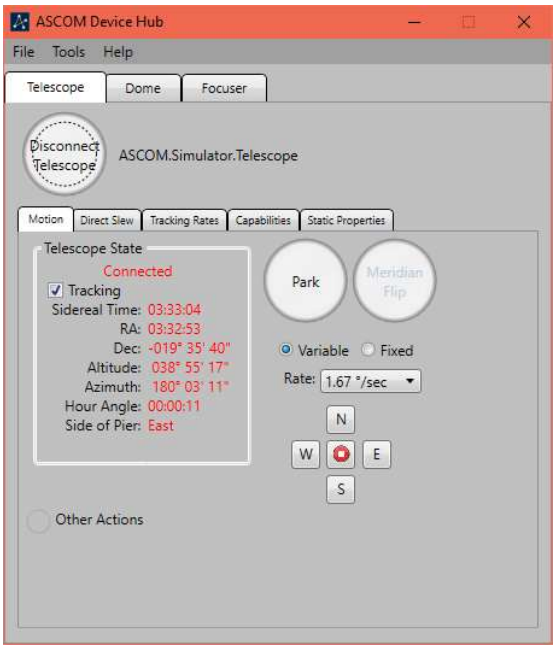
Device Hub supports 2 visual themes. They are labelled Standard and Custom. The Standard theme uses a mostly black on light gray color scheme with normal control colors and styles, but with some color changes for emphasis.

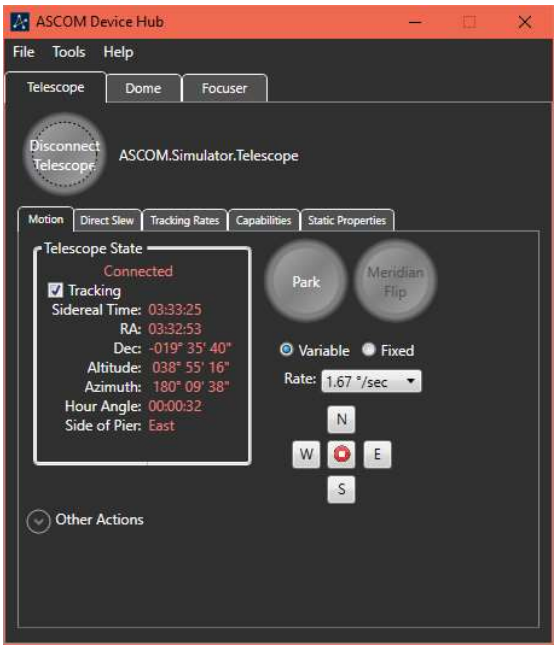
The Custom theme uses a mostly white on dark gray color scheme with custom styles for many controls and other colors for emphasis.

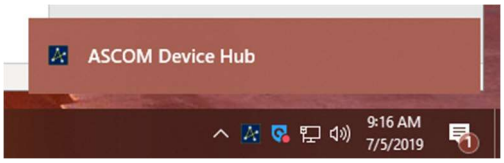
The examples below show both the Standard and Custom visual themes, with the Standard theme on the left.

When you change the selected theme, the change is made as soon as the OK button is pressed.

The figures below show both the Standard and Custom visual themes.





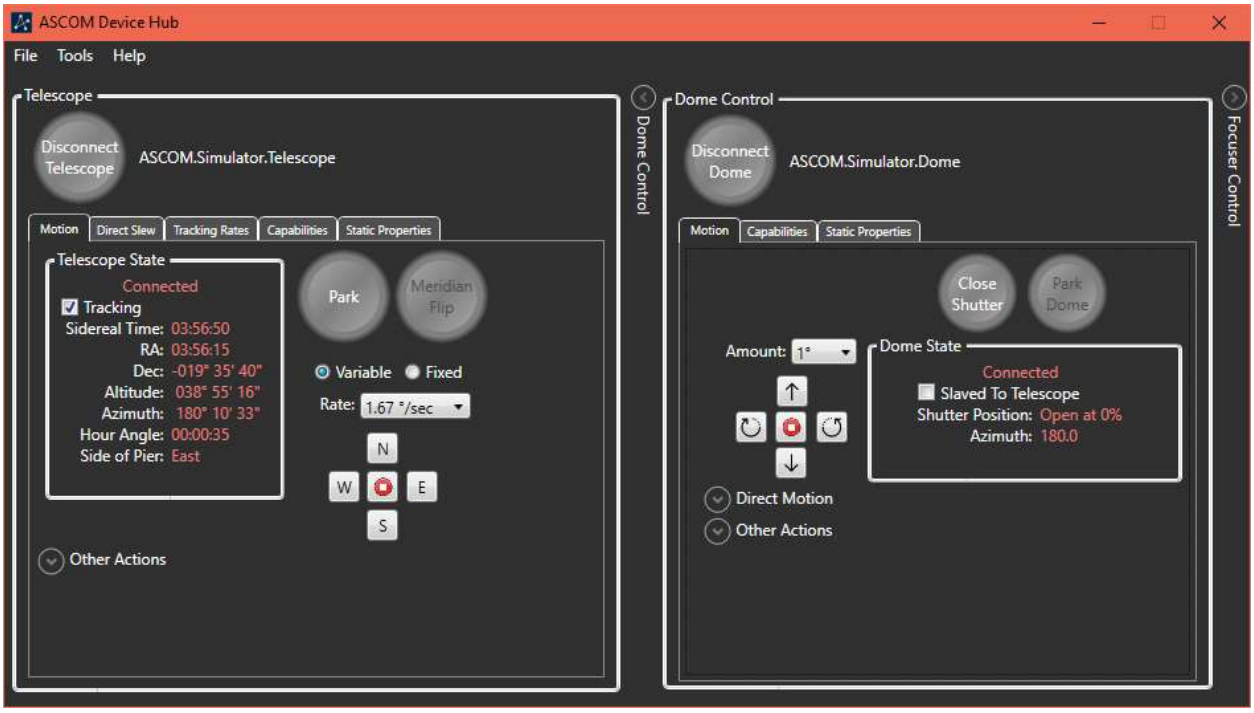


When a Device Hub driver is connected from another application, the Device Hub application is placed in the Notification Area, also known as the System Tray. Different versions of Windows manage the Notification area differently. For example, on Windows 10 the icon for the Device Hub is initially shown with a notification window that disappears after a few seconds. Once the window has gone away, the icon may be moved to the notification overflow area. The option to Suppress Activation Notification prevents the notification window from being displayed. It may have a different effect on other versions of Windows where notifications are managed differently.

By default, the device-specific data is organized in separate tabs. This minimizes the screen area that the application occupies but does not allow you to look at the telescope information and the focuser information at the same time, for example. The option to Use Expanded Screen Layout allows you to see one, two, or all 3 devices in a side-by-side arrangement.

When the Use Expanded Screen Layout is selected, the device tabs are replaced by expander controls which occupy the right hand side of the Device Hub Main window. You can expand the view for either the dome or focuser by clicking on the expander for that device. By default, you can only see 2 devices at the same time. So clicking on the Focuser Control expander will automatically collapse the Dome Control expander. In order to see all three devices, simply hold the Ctrl key down when clicking on the expander for the 3rd device.

The figure below shows the application when both the Telescope and Dome are visible while the Focuser pane is collapsed.

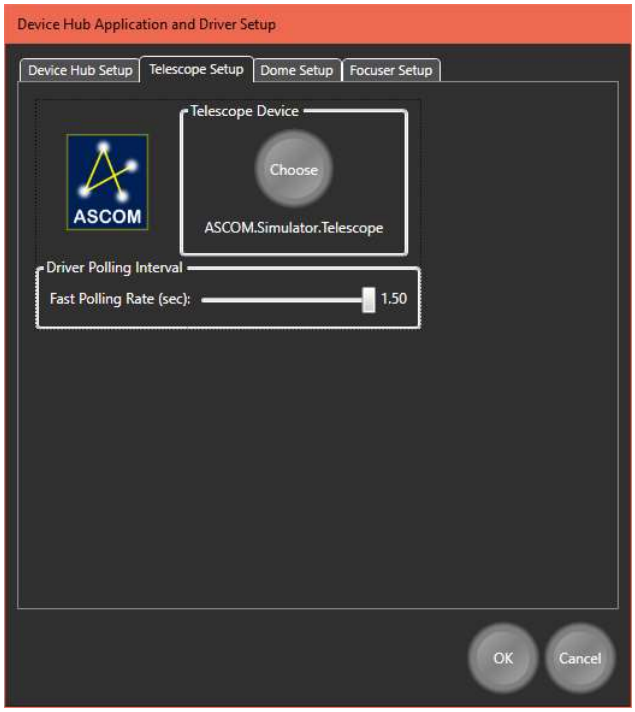


Device Configuration

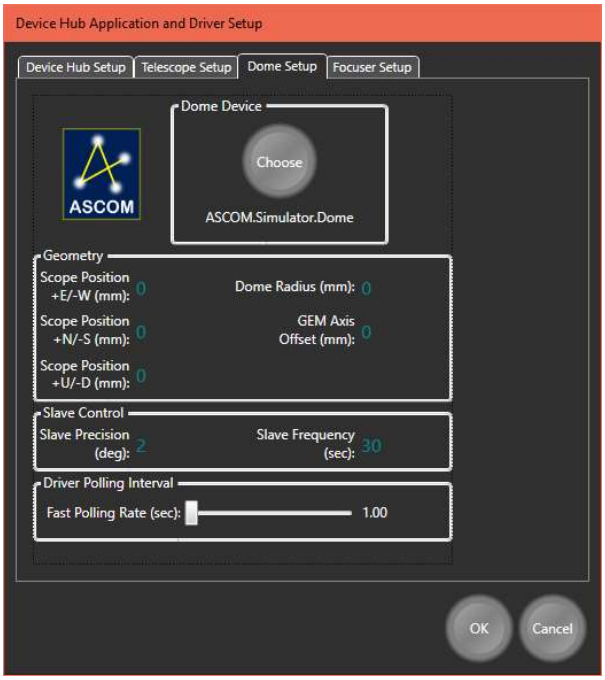
One of the first tasks will be to select and configure the devices that are to be controlled by the Device Hub. The Setup item on the Tools menu is used to accomplish this step.

For the Telescope Setup, the only task is to select the telescope driver. Clicking on the round Choose button displays the ASCOM Telescope Chooser to allow you to select and configure the telescope. A slider control is also available to allow you to the interval between polls of the connected driver. The fast update is used when the telescope is slewing. When the slew completes, the rate is slowed to its normal value.

There are additional options to modify when configuring the Dome or the Focuser.



In order to synchronize the pointing position of the dome, the Device Hub needs some information about the configuration of both the telescope and the dome. This is the same information that POTH uses so if you have POTH correctly configured then you can just transfer the values from POTH to the Device Hub.



If your dome or telescope are new then you must very carefully and accurately measure the distances from the intersection of the mount's RA and Dec axes to the geometric center of the dome. In the following discussion consider point A as the being at the intersection of imaginary lines drawn through RA and Dec axes of the mount. If your mount has a hole through the internal portion of the counterweight bar to support a polar alignment scope, Point A would be in the exact center of that hole. The origin point is at the geometric center of the dome.

Having determined these two points, you need to accurately measure how far apart they are and enter the values into the Dome Setup dialog. In the example on the right, point A is 9 mm east of the dome origin, 48 mm north of the dome origin and 44 mm above the dome origin.

Other Dome configuration settings include the radius of the dome, the distance from the intersection of the RA and DEC axes (Point A) to the nearest point along the centerline of the telescope's optical path. This is the GEM Axis Offset value (760 mm) in the screenshot. It should be 0 for a non-GEM mount.

The Slave Frequency is a measure of how often the slaving function of Device Hub should consider whether it needs to move the dome to keep the opening over the telescope. These are very dependent on the orientation of the scope and dome themselves. For a telescope with a narrow field-of-view where the shutter is near the end of the telescope, it may be possible to crease the Slave Frequency interval and/ or the Slave Precision (slop) value. Typically, these values should be set as large as possible while still keeping the dome's opening completely over the pointing position of the telescope.

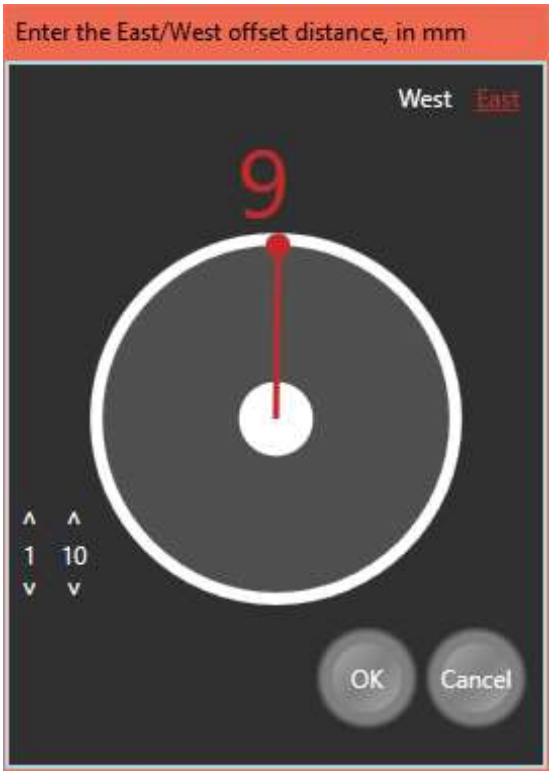
The Fast Polling Rate changes how often Device Hub polls the dome driver for updated status values when the dome is slewing.

There are only three setup parameters for the Focuser. The first is the focuser driver's identifier. The second is an adjustment value for the focuser's reported temperature. It is only useful if 1) your focuser reports its temperature and 2) the focuser driver does not provide a method for calibrating that temperature. Changing the value only affects the displayed temperature. It does not affect the focuser's

internal temperature setting or its adjustment of the focuser position with changes in temperature. The value can also be adjusted from the Focuser Motion Tab while the focuser is connected. The 3rd focuser configuration parameter is the polling interval when the focuser is moving.

Numeric Data Entry

When looking at the Dome configuration settings, you may have noticed that there did not seem to be a way to change the values. By clicking on the number (not the identifying text), a pop-up editing window is displayed. This type of editing window is used throughout the application.



The gray and white circle with the red hand in the center of the dialog is a rotary slider control. The position of the hand corresponds to the value that is displayed above the circle. The value is adjusted by dragging the hand with the mouse pointer. The number above the slider changes as the hand is dragged. The text at the upper right is used to adjust the sign of the number (West is negative). East, underlined in red, is selected. To the lower left of the rotary slider are some up and down buttons which can be used for fine adjustment of the value. They are especially useful when the number has a large range.

As expected, pressing the Cancel button closes the dialog without changing the value and pressing OK transfers the new, changed value to the parent window.

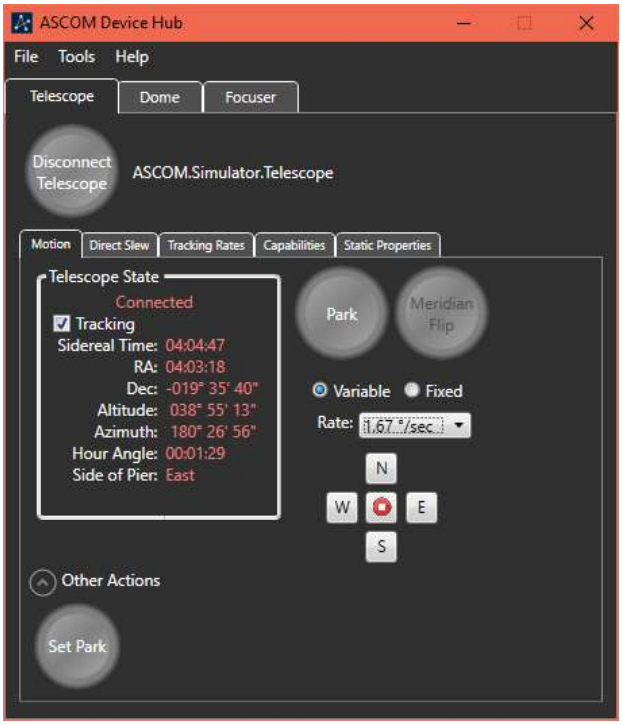
Some numbers can have multiple components. For example, a declination value has a sign, as well as components for entering degrees, minutes, and seconds. When using the rotary slider to enter a new declination, all the components are displayed above the rotary slider. You can select each component individually and drag the slider's hand to adjust that value.

Once you have configured your devices, the Setup dialog window can be dismissed. You are now ready to connect the Device Hub with your devices. For the remainder of this document, the ASCOM telescope, dome, and focuser simulators are used as examples.

Telescope Control - MotionTab

Clicking on the Connect Telescope button will immediately connect the Device Hub with the configured telescope and begin communicating with the telescope. This allows the values to be filled in with the data that is being read from the telescope driver.

The numeric values that are grouped in the Telescope State box are continuously updated while the Device Hub is connected to the telescope. In addition, the tracking state can be changed. To the right of the Telescope State group box are buttons to allow you to slew the scope to the Park position and stop tracking and to perform a Meridian Flip (for a GEM mount).



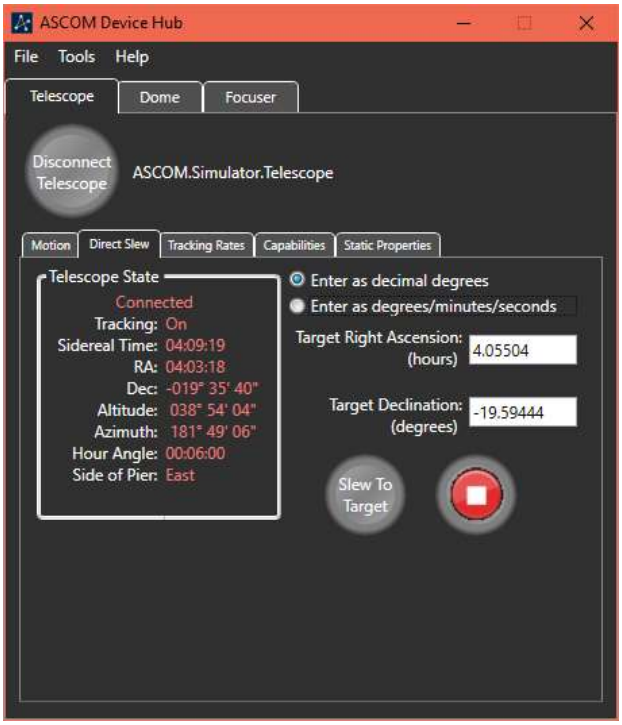
Also to the right of the group box are the movement controls. Two types of movement are supported. When Variable movement is selected, the Rate dropdown list is populated with rates that are reported as being valid by the telescope driver. Movement is accomplished by pressing and holding one of the directional buttons. While the button is pressed, the telescope will move at the specified rate. When the button is released, the telescope will resume tracking at the designated rate, usually the Sidereal rate. When the Fixed rate option is selected, the Amount dropdown list is re-populated with a different list of values. These values are movement amounts and range in value from 1 arc-minute to 40 degrees. In this mode, one click of a button moves the mount by the selected amount in the selected direction. Movement does not start until the button is released.

At the bottom of the Telescope State group box is some empty space that is reserved for transient indicators. For example, when the mount is actively slewing, the word “Slewing” will appear in that area. While the mount is moving toward the Park position the word “Parking” will appear in that place. Finally, if the mount has tracked past the meridian and the counterweight bar is pointing above the horizon, the flashing text “Weight Up” is displayed in that area.

The Other Actions expander has a button to set the Park position to the current mount altitude and azimuth. The button is only enabled if the telescope supports the CanSetPark capability.

Telescope Control – Direct Slew Tab

The Direct Slew tab provides a way for the user to enter absolute target coordinates and to slew the telescope to those coordinates. As a convenience, a Telescope State group box is also displayed on this tab. However, you must navigate to the Motion tab to change the Tracking flag.



The state of the Tracking flag determines whether the coordinates for the slew are celestial (right ascension and declination) or terrestrial (azimuth and altitude). Tracking must be On in order to perform a direct RA/Dec slew. If Tracking is Off direct slews will to azimuth and altitude.

Coordinates can be entered either as decimal numbers or in sexagesimal format (degrees, minutes, and seconds).

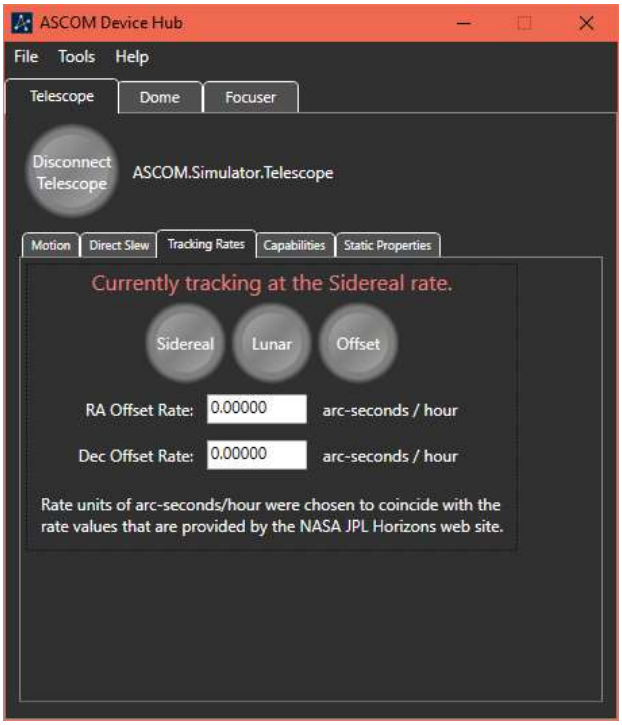
To change the degrees, minutes, or seconds, you click on the displayed value to pop-up a rotary slider window. To enter as decimal degrees, first select the “Enter as decimal degrees” radio button and enter the new target value in the displayed text box.

When you are satisfied with the target values you can click on the “Slew To Target” button to initiate the slew. The button on the right allows you to immediately abort the slew. While the slew is in progress, the word Slewing is displayed between the two buttons and the actual pointing position of the telescope is displayed below the buttons.

Telescope Control – Tracking Rates Tab

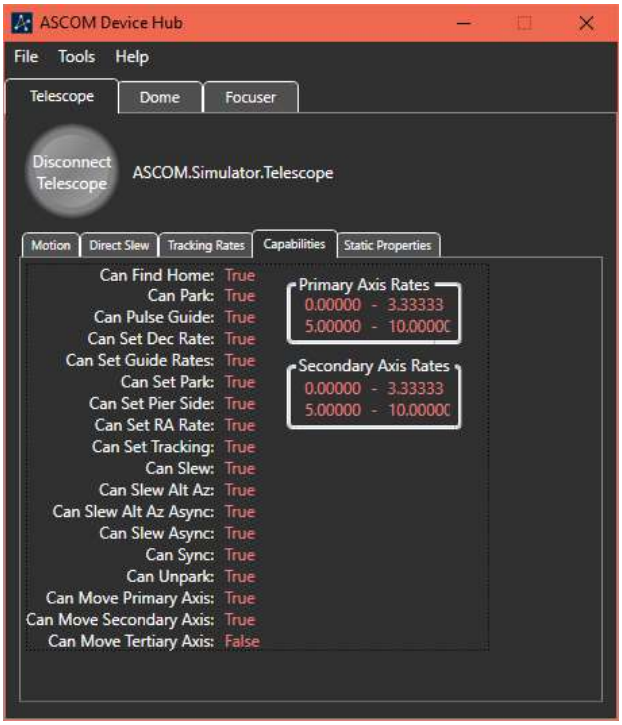
The Tracking Rates tab allows you to alter the rate at which the telescope moves when tracking. The options allow for tracking at the Sidereal rate or the Lunar rate. In addition, if the driver supports changing the RA and DEC rates, the tracking rate can be altered to allow tracking of comets or other Near Earth Objects. When Offset tracking is used, the offsets are applied to base Sidereal rate.

The offset rates are not changeable when the Device Hub is in Offset mode. To change either of the rates, first select Sidereal, change the rate(s), and then return to Offset mode.



Telescope Control – Capabilities Tab

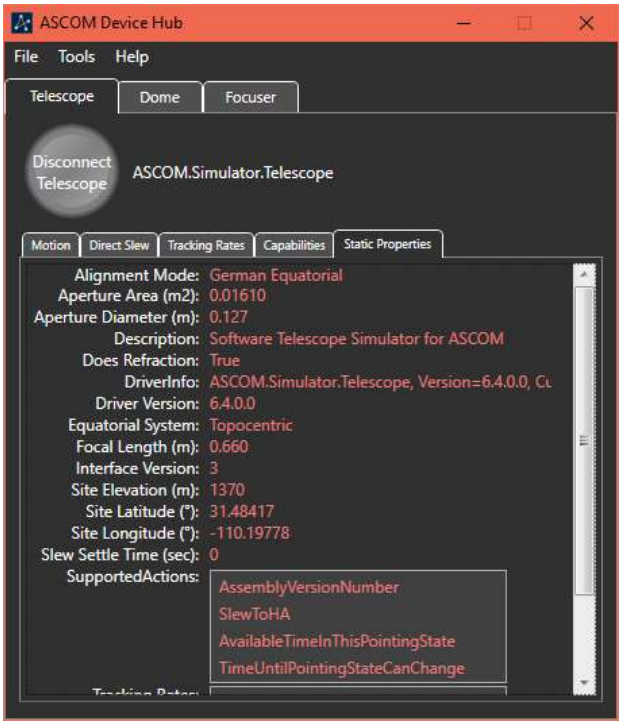
In order to support a wide range telescopes with different abilities, the ASCOM standard provides for properties with indicate which abilities are supported by the driver and the mount. Most of these properties have names that start with Can. These Capabilities are static and are not changeable by client applications. Since they do not change they are only read at connect time. A complete list of these properties is viewable on the Capabilities Tab.



Telescope Control – Static Properties Tab

There is also a large list of properties with numeric or textual values that do not often change. However some of these properties can be modified by client applications. These properties are grouped together and displayed on the Static Properties tab page.

Since the amount of information is variable and can exceed the available display space, a vertical scroll bar is provided.

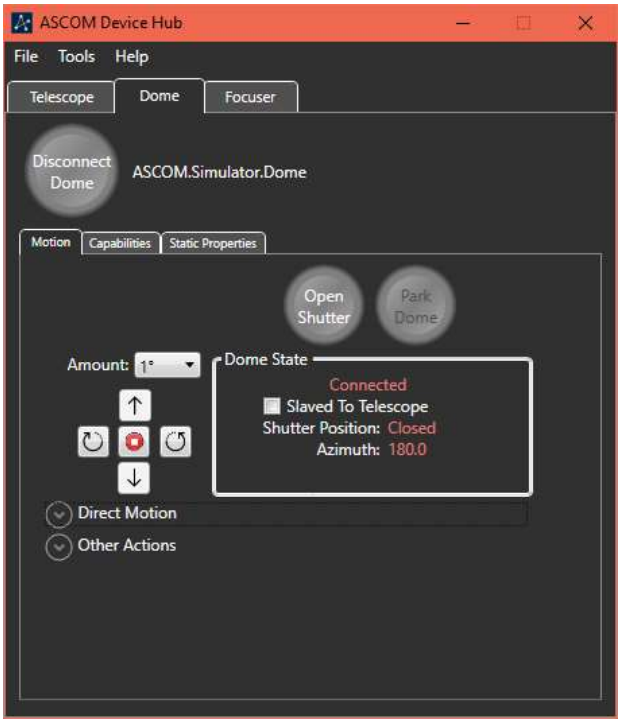


Dome Control – Motion Tab

When you connect the Device Hub with your dome, the Device Hub begins communicating with the dome to present the capabilities of the dome as well as its current state. The Dome Motion tab presents the critical up-to-date information and allows you to change the state of the dome and to fully exercise the dome's capabilities.

For a fully functional dome you can control it in the following ways:

- Open or close the shutter
- Park the dome
- Send the dome to its home position
- Adjust the shutter's altitude
- Jog the dome in either azimuth or altitude by a selected amount
- Stop a rotational move or shutter move that is currently occurring



- Synchronize the dome's azimuth at a specified position
- Slave the dome to a connected telescope

The Other Actions control group contains an option to switch between different methods for calculating the dome azimuth when slaved to a telescope. One of the calculation methods gives a solution that compares to what POTH calculates. The other option uses a different mathematical algorithm to calculate the dome's position. The two methods give slightly different results. Thanks to Tom How for providing the basis of this method. Any differences between Tom's algorithm and my implementation of his solution are most likely due to errors that I made. Feel free to use whichever option works best with your dome.

Also in the Other Actions control group is an adjustment value that is added to the calculated dome azimuth value before that value is used to adjust the dome. This offset may be useful if the dome slit is almost, but not quite, centered over the scope. This value is saved in the Device Hub Dome driver profile and can be changed on the fly. It takes affect with the next dome calculation and can adjust the dome's azimuth by as much as 20 degrees in either direction.

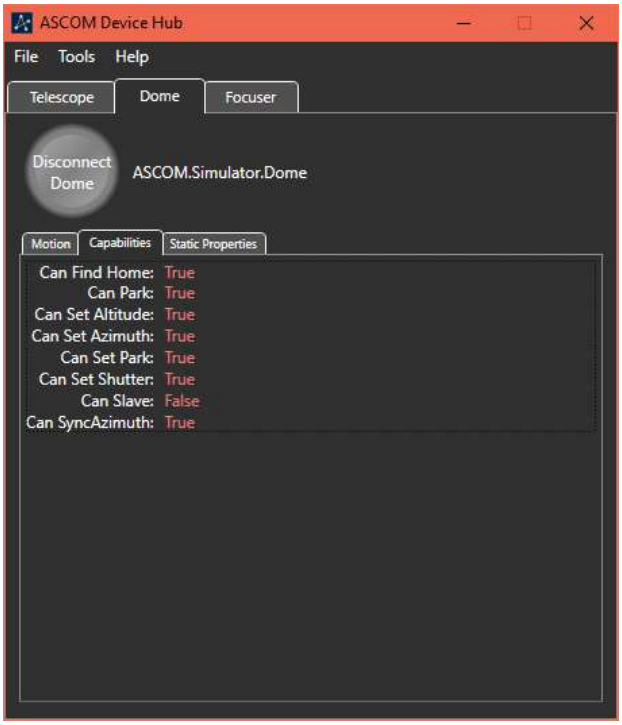
"Sync To Azimuth" controls are also displayed when the Other Actions control group is expanded. This provides a way to sync the dome driver's position with a known azimuth.

Expanding the Other Actions group cause the Direct Motion group to automatically contract, and vice versa.

In order to successfully slave the dome to a telescope, the telescope must be connected, and it must be reporting its pointing azimuth and elevation. The dome must be connected and have one of the capabilities CanSetAltitude or CanSetAzimuth be set to True. Of one of these capabilities is set to False then slaving adjustments will exclude that axis. For example, if CanSetAltitude is False then the dome will only be slaved in Azimuth.

Dome Control – Capabilities Tab

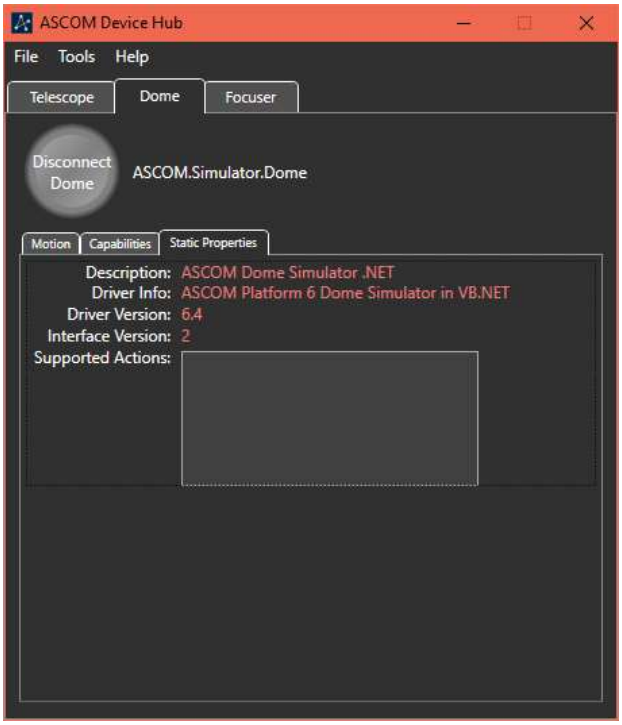
Like other ASCOM device types, a dome has capabilities that are specified through a series of CanXXX properties. The values of these properties reflect the abilities of the dome itself as well as the dome driver implementation. For example, an observatory with a roll-off roof would have the CanSetShutter capability set to True and the other capabilities set to False.



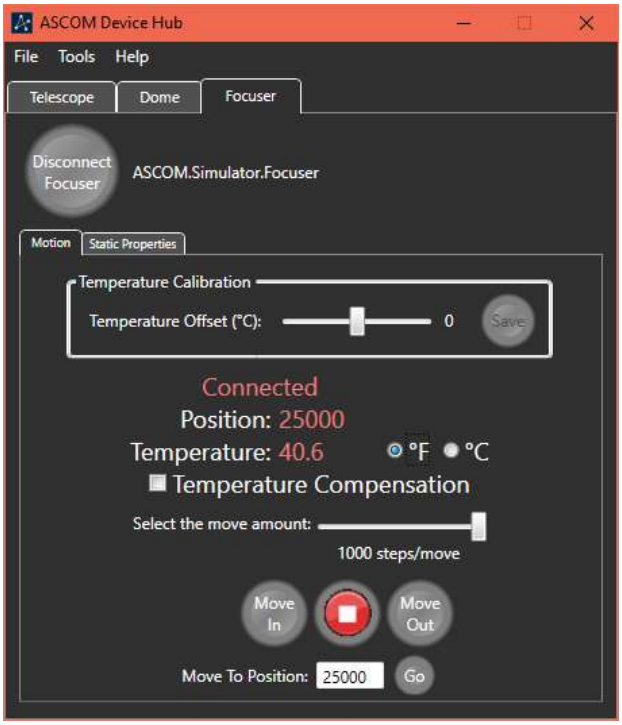
The ASCOM Dome Interface Specification allows for a dome to be capable of being slaved directly to a telescope via a hardware connection between the telescope and the dome controller. Since one of the major benefits of the Device Hub is to perform that slaving itself, enabling or disabling hardware slaving is not supported.

Dome Control – Static Properties Tab

In addition to the Capabilities, a dome driver also has a few static properties. These properties and the values for the current version of the Dome Simulator are listed in the screenshot. The values of these properties are provided by the driver and are not changeable by the Device Hub or a client application.



Focuser Control – Motion Tab



When you connect the Device Hub with your focuser, the Device Hub begins communicating with the focuser to present its capabilities as well as its current state. The Focuser Motion tab presents current information from the focuser driver and allows you to interact with the focuser.

Focuser interactions include enabling or disabling temperature compensation (if supported by the focuser and driver) and changing the focuser's position by using the Move In or Move Out buttons.

The Temperature Compensation checkbox is hidden if the focuser does not support this feature.

The movement amount for the Move In and Move Out buttons is specified by using the slider above the buttons. The movement amounts range from 1 step to 1000 steps, in convenient increments.

If the focuser is an absolute focuser you can also change its position by entering a position in the text box that is labeled "Move To Position" and pressing the Go button. The "Move To Position" controls are hidden if the focuser operates as a Relative focuser.

Between the Move In and Move Out buttons is a button to abort movement. From a practical standpoint small moves happen so quickly that stopping movement may not be possible, but this button may be useful for aborting a large move.

This screen also provides the ability to change the temperature scale between Celsius and Fahrenheit and to adjust and save the temperature offset calibration value. This is the same value that can be set from the Focuser Setup Tab that was previously discussed. The calibration offset value is always in degrees Celsius. The adjustment range is ±10 degrees.

Focuser Control – Static Properties

A focuser is a simpler device than a telescope, or even a dome, so the list of optional capabilities is much smaller. So, the capabilities are listed on the Static Properties screen. The only optional feature of a focuser, as defined in the IFocuserV3 interface, is whether the focuser supports internal temperature compensation.

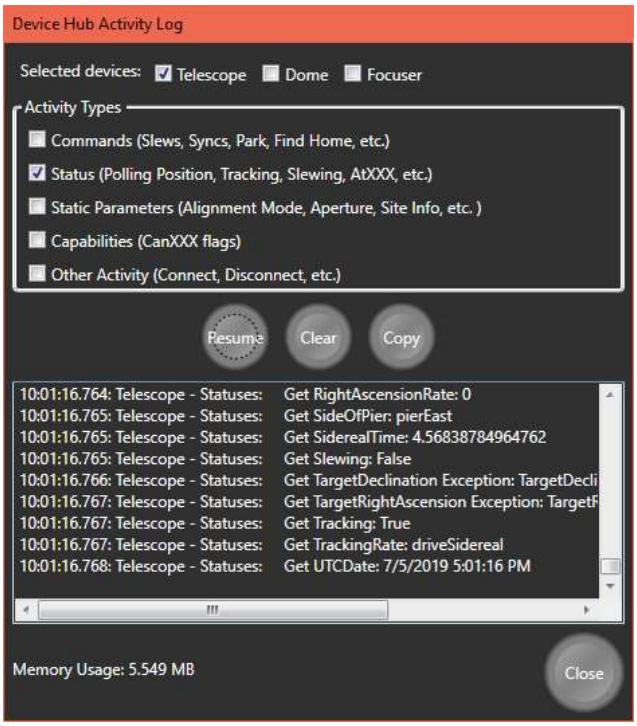
One other characteristic of the focuser is whether it operates as an absolute or relative focuser. For more information about this property, please read the ASCOM Developer Help document.

Logging Device Activity

The Device Hub can display communications between itself and any connected driver. This is accomplished with the Activity Log. The Activity Log can be viewed by selecting the View Activity Log option from the Tools menu.

Once the Activity Log dialog is displayed you can select which devices to monitor and which classes of messages to display for the selected devices.

You can Pause/Resume the display of new traffic. You can erase all logged messages with the Clear button and you can Copy all logged messages to the Windows Clipboard. The copied information can then be pasted into any program that accepts text data from the Clipboard.



Use of the Clipboard is the only way to save logged information. It is not automatically saved to a disk-based file. Otherwise, any logged information is lost when the Activity Log is closed.

The capacity of the Activity Log is limited to 125,000 characters. Once that capacity is reached, the oldest data is removed whenever new data is added in order to stay below the limit.

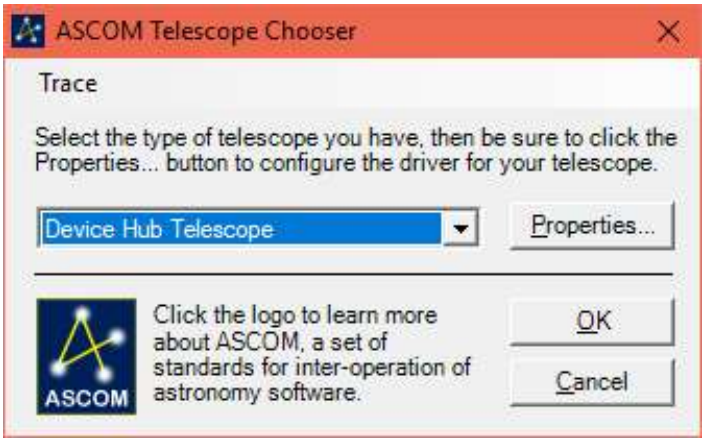
In addition the current amount of system memory being used by the Device Hub is displayed at the lower left part of the Activity Log dialog window.

Connecting to the Device Hub as a Device

In addition to behaving like an ASCOM client application, the Device Hub exposes a telescope, a dome, and a focuser that other applications can connect to and interact with. This capability provides several advantages.

It allows you to connect the Device Hub to your telescope and your dome. You can then connect to the Device Hub telescope from a planetarium program and use the planetarium program to slew the telescope with the dome following the telescope as it moves from target to target.

It also allows the capability of allowing multiple programs to be connected to each device, even if the



device driver only supports a single connection.

To connect to the device hub as a telescope you simply select the Device Hub Telescope from the Telescope Chooser dropdown list.

You should also find Device Hub Dome listed in the Dome Chooser and Device Hub Focuser listed in the Focuser Chooser.

Technical Architecture and Design of the Device Hub

The Device Hub application is written in C# using Visual Studio 2019 Community Edition. The ASCOM templates were used to generate the starter code for the local server and for each of the exposed drivers. The structure of the code is a bit different from other local servers due to the choice of Windows Presentation (WPF) for the user interface. WPF applications do not expose a Main entry point as Windows Forms applications do. The application's entry point is hidden in an App object. The App object is defined by the application definition in App.xaml and App.xaml.cs, its code-behind class. The code-behind class overrides the OnStartup method to start up the local server.

In addition to using WPF, the application is organized using the Model-View-ViewModel (MVVM) architecture. This design allows for separation of the U/I and business objects. The business object classes have no knowledge of or dependence on the user interface. This allows the business object classes (view models) to be completely tested from Unit Test projects without any user interface.

Each Window or UserControl (the main U/I components) is a view. For most views, the code-behind is completely empty. Instead, marshalling data between the view and the viewmodel is accomplished by databinding. Also, instead of using event handlers to perform actions as requested by the U/I, the viewmodels implement ICommand methods that are hooked to the various controls via WPF data binding. Below is a simple example:

```
<Button Grid.Row="5" Grid.Column="1" Width="24" Height="24" Margin="0,5"
       VerticalContentAlignment="Center" HorizontalContentAlignment="Center"
       ToolTip="Abort Slew"
       Command="{Binding StopMotionCommand}"
       IsEnabled="{Binding Status.Connected, FallbackValue=False}">
    <Image Margin="2" Source="/ASCOM.DeviceHub;component/Images/Stop.ico" />
</Button>
```

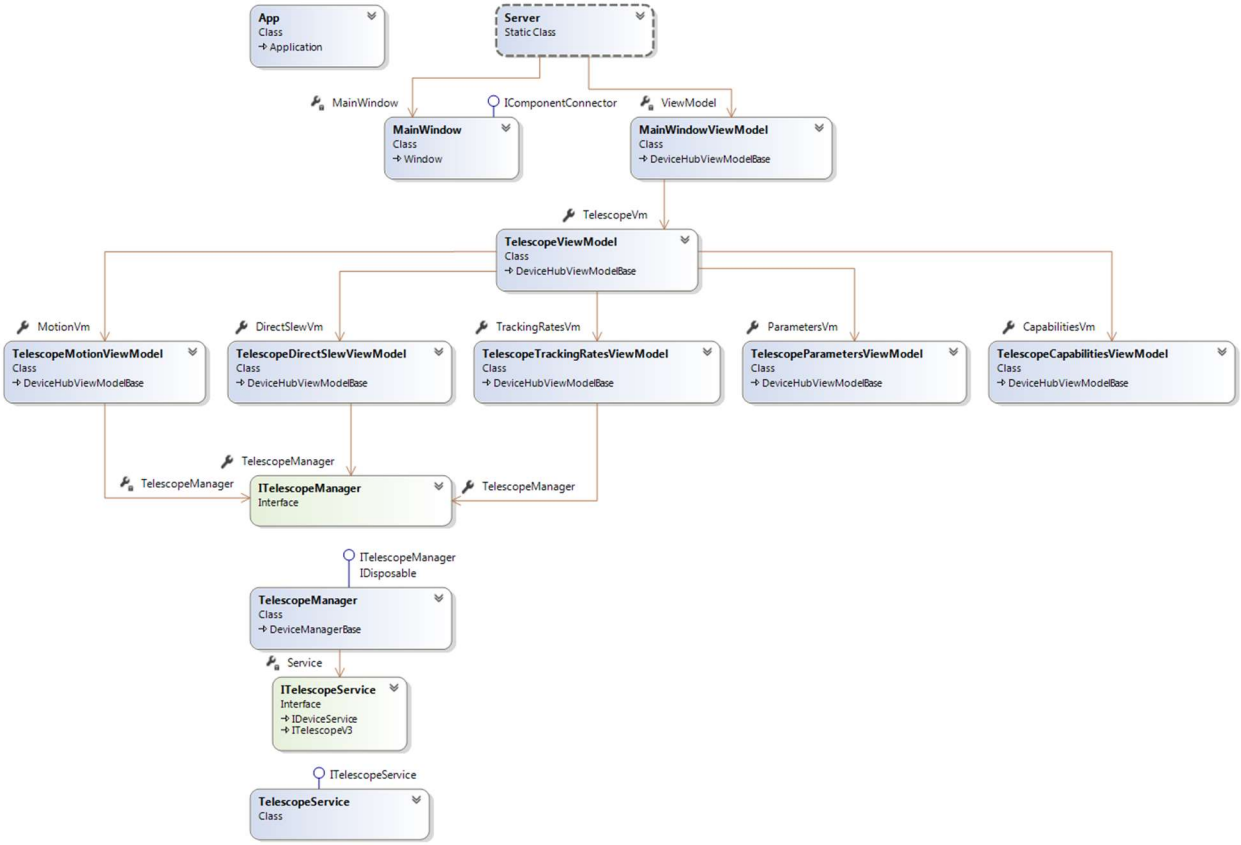
Above is the XAML description of an example button object. The Command attribute is bound to the viewmodel's StopMotionCommand property. That property is declared as follows:

```
private ICommand _stopMotionCommand;

public ICommand StopMotionCommand
{
get
    {
        if ( _stopMotionCommand == null )
        {
            _stopMotionCommand = new RelayCommand( param => this.StopMotion( param ) );
        }

        return _stopMotionCommand;
    }
}

private void StopMotion( object param )
{
    if ( IsVariableJog && _jogInProgress )
    {
        MoveDirections direction =
            ( param == null ) ? MoveDirections.None : (MoveDirections)param;
        TelescopeManager.StopJogScope( direction );
    }
}
```

The RelayCommand class implements the ICommand interface. It resides in the Business Object Classes folder of the DeviceHub project. The class supports definition of an Action object, StopMotion in the example above, and a “can execute” predicate. Although the above example does not define the predicate, most instances of RelayCommand do. These predicates are used, automatically by the WPF data binding logic, to enable or disable the bound control based on the predicate’s return value.

Another consequence of designing for testability is the use of services to provide access to logic that is external to the application classes. Several services have been incorporated into the architecture. Each of the ASCOM device objects is wrapped in a service. In addition, there are services for displaying the activity log, for loading and saving the application settings, and for displaying Windows message boxes and other modal dialogs. These services are injected during the application startup, but mock versions of each service that implement the same interfaces are injected during unit testing. This allows for the creation of unit test classes and methods that allow for thorough, automated testing of each layer of software between the views and the services. This includes the view models and the device managers.

Below is a simple class diagram showing the application skeleton of the classes involved in telescope control:



Unfortunately, the Visual Studio class diagram tool does not provide the ability to define the dependency between the App class and the static Server class. The best that I can do is to simply show the App class. The dependency does exist, however. The App's instance overrides the base class' OnStartup method to invoke the Server class Startup method and pass it any command-line parameters.

The Server.Startup method instantiates the MainWindow object and the MainWindowViewModel object and connects them, via the MainWindow's DataContext property. This allows properties and commands in the viewmodel to be accessible from the view.

The remainder of the startup method is very similar to that which is created by from the LocalServer template.

One generated class that is substantially different is the GarbageCollection class. GarbageCollection is instantiated from the Server class, during startup, and cancelled when the main view is closed. The Device Hub uses the System.Threading.Tasks.Task class to run the garbage collector on a worker thread. At the time of this writing, the garbage collection interval is coded at 10 seconds. However, due to the capabilities of Task objects and their CancellationTokens, the garbage collector can be immediately cancelled without waiting for the garbage collector's sleep timer to expire. This means that disposed objects are cleaned up more quickly during shutdown.

The MainWindowViewModel instantiates the TelescopeViewModel which in turn instantiates the viewmodels for each of the Telescope control tabs (Motion, DirectSlew, TrackingRates, Parameters, and

Capabilities). The TelescopeView.xaml definition defines the view to be used for displaying those viewmodels.

The Resources element of the TelescopeView (which is a UserControl) declares a DateTemplate which tells the WPF rendering engine which view to use for displaying that viewmodel.

```
<UserControl.Resources>
    ▪
    ▪
    <DataTemplate DataType="{x:Type local:TelescopeMotionViewModel}">
        <local:TelescopeMotionView />
    </DataTemplate>
    ▪
    ▪
```

The viewmodel instance is bound to the Content property of a ContentControl which is displayed in the appropriate TabItem.

```
▪
▪
<TabControl Grid.Row="2" Background="Transparent" Height="360"
              HorizontalAlignment="Stretch"
              SelectionChanged="TabControl_SelectionChanged">
    <TabItem Header="Motion" HeaderTemplate="{StaticResource headerText}"
              Style="{StaticResource DarkTabItem}" Width="Auto">
         <ContentControl Content="{Binding MotionVm}" Style="{StaticResource Content}"
                      HorizontalAlignment="Stretch" />
     </TabItem>
▪
▪
```

Some of the telescope viewmodels only provide information from the telescope for display others need to affect changes to the state of the telescope. For example, the TelescopeCapabilitiesViewModel holds the values of the Capabilities properties that are read from the device when it is initially connected. The values of these properties are displayed by the TelescopeCapabilitiesView. This viewmodel needs no further interaction with the telescope.

On the other hand, the TelescopeMotionViewModel needs to interact with the telescope to display the current telescope state AND to change property values and call methods on the telescope object that affect the telescope's state. This interaction is supported by the TelescopeManager. There are also DomeManager and a FocuserManager classes that support access from the viewmodels to those devices. These objects are singleton objects that are created in the MainWindowViewModel and injected into the created viewmodels as constructor arguments. During unit testing of a viewmodel a mock version of the device manager is injected into the VM being tested.

The design of the device managers makes use of the capability of C# to split a class definition over multiple files. So, there is a TelescopeManager.cs file as well as a TelescopeManagerAccess.cs file. Both files define properties and methods for the TelescopeManager class. Where necessary, properties and methods in TelescopeManager.cs use calls in TelescopeManagerAccess.cs to communicate with the

device service (more in the next paragraph), rather than communicate directly with the device service. The methods in TelescopeManagerAccess perform checks to ensure that we are connected to the device and to forward the details about the access to the activity logger. The properties and methods in TelescopeManagerAccess are also used by the exposed Telescope device driver to access the served Telescope driver.

The device managers are the most complex classes in the application and their testability is a requirement. They communicate with the ASCOM device objects through a service which wraps the ASCOM calls to the driver. Each service has an abstract definition and two concrete definitions. One concrete definition is for the normal operation of the application and the other is a mock definition for unit testing. For the most part the concrete TelescopeService is a simple wrapper which passes through any property values or method calls to the associated ASCOM property or method. The concrete MockTelescopeService, however has the capability to allow the Unit Testing infrastructure to initialize the service to support each of the individual tests.

The above discussion presents some detail about the responsibilities of the classes that are included in the previous class diagram. The organization of classes for the Dome and Focuser devices are organized in a similar manner.

The above discussion, while fairly detailed, is incomplete. It does not mention how information is propagated between the device managers which generate the data by reading it from the device and the VMs that need to provide the data for display. This job is the responsibility of the Messaging subsystem.

Messages are like events but are more powerful since any class can subscribe to a message and any class can generate a message. Rather than design a Messaging system from scratch, I decided to use one that is part of an open source project where the author has generously made the source code available.

One thing that I discovered when I started writing WPF MVVM applications is that there are a lot of common needs that they all share. One solution to making an aspiring MVVM developer more productive is to utilize a 3rd party library. One such library is the MVVM Light Toolkit from Laurent Bugnion. This tookit is available as a NuGet package that can be added to a Visual Studio project. The author also makes it available for download from a GitHub repository. After reading the license under which the author makes the code available, I extracted only the classes of the messaging component and incorporated them into an assembly, MvvmMessenger, which is part of the DeviceHub solution. This decouples the Device Hub from dependence on the MVVM Light Tookit and relieves us from the burden of distributing the entire toolkit with the Device Hub.

The TelescopeManager begins communicating with the Telescope (via the TelescopeService) when it is connected. The Capabilities and Parameters information are immediately read from the device and forwarded by means of messages. The following code fragments show how the capabilities properties are read from the device and forwarded to viewmodels that either display the values or use the values to make decisions about how to interact with the device.

```
private void ReadInitialTelescopeDataTask()
{
    // This task talks to the telescope on a worker thread but updates the U/I on the main
    // thread.

    // Wait a second for the telescope to settle before reading the data.

    Thread.Sleep( 1000 );

    Capabilities = new TelescopeCapabilities();
    Capabilities.InitializeFromManager( this );

    Parameters = new TelescopeParameters();
    Parameters.InitializeFromManager( this );

    DevHubTelescopeStatus status = new DevHubTelescopeStatus( this );

    Messenger.Default.Send(
        new TelescopeCapabilitiesUpdatedMessage( Capabilities.Clone() ) );
    Messenger.Default.Send( new TelescopeParametersUpdatedMessage( Parameters.Clone() ) );
    Status = status;
    Messenger.Default.Send( new TelescopeStatusUpdatedMessage( status ) );
}
```

This method fragment is from the TelescopeManager. It is called on a worker thread immediately upon successfully connecting with the device. It creates an instance of the TelescopeCapabilites class and initializes it by making calls to the ITelescopeService object to read the capabilities property values from the ASCOM Telescope object. Further down in the code it sends a message where the message payload is a clone of the Capabilities object.

One of the obvious recipients of this message will be the TelescopeCapabilitiesViewModel. Here is the messaging-related code from the VM:

```
public TelescopeCapabilitiesViewModel()
{
    Messenger.Default.Register<TelescopeCapabilitiesUpdatedMessage>( this
                                    , ( action ) => UpdateCapabilities( action ) );
    Messenger.Default.Register<DeviceDisconnectedMessage>( this
                                    , ( action ) => InvalidateCapabilities( action ) );
}
▪
▪
▪
private void UpdateCapabilities( TelescopeCapabilitiesUpdatedMessage action )
{
    SetCapabilities( action.Capabilities );
}
▪
▪
▪
private void SetCapabilities( TelescopeCapabilities capabilities )
{
    // Make sure that we update the Capabilities on the U/I thread.

    Task.Factory.StartNew( () => Capabilities = capabilities
                    , CancellationToken.None
                    , TaskCreationOptions.None
                    , Globals.UISyncContext );
}

protected override void DoDispose()
{
    Messenger.Default.Unregister<DeviceDisconnectedMessage>( this );
    Messenger.Default.Unregister<TelescopeCapabilitiesUpdatedMessage>( this );
}
```

The VM's constructor registers a subscriber for the message and specifies the message handler method, UpdateCapabilities, in this case. So, when the Telescope Manager sends the message, UpdateCapabilities is called to receive it. The Capabilities object is extracted from the payload and passed to SetCapabilities which updates the class's Capabilities property on the main User Interface thread.

WPF has the same limitation as Windows Forms in that controls can only be updated by code that is running on the same thread that created them. Therefore, views and viewmodels are always created on the main thread and viewmodel properties that are bound to controls in the view can only be updated by code that is running on that thread.

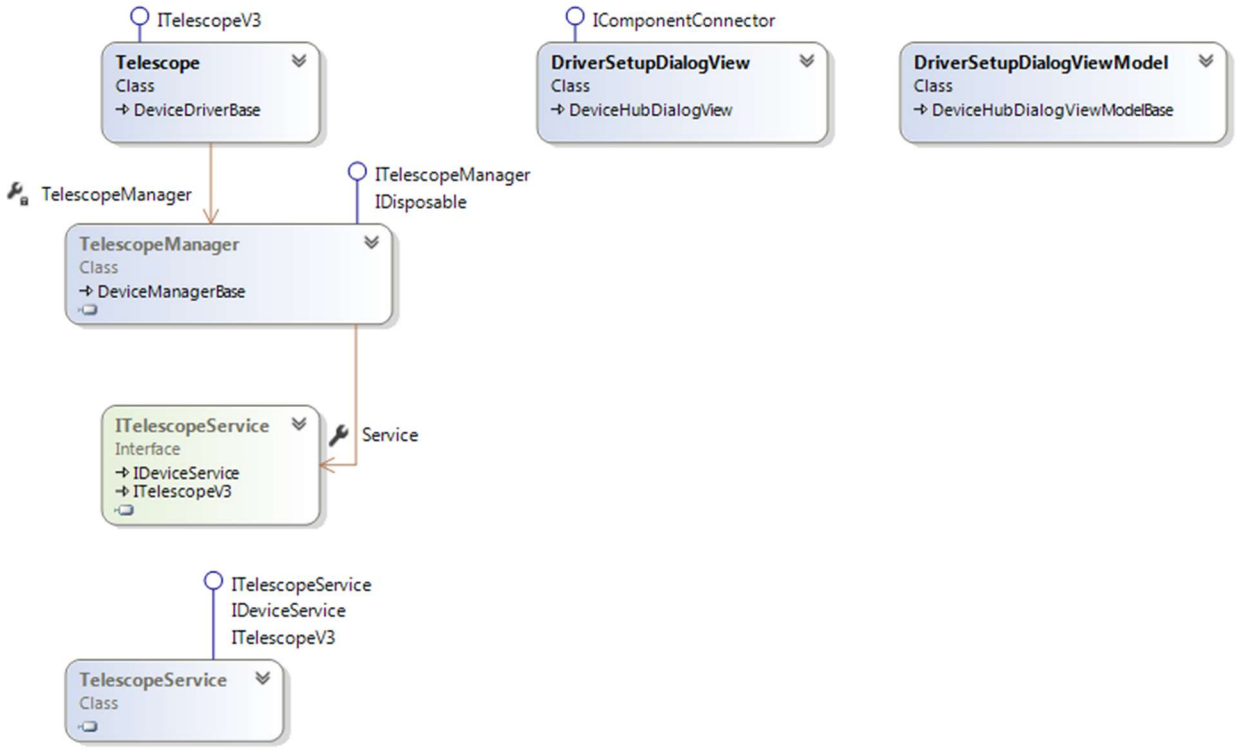
One of the abilities of the Task and TaskFactory classes is to execute the thread method on the thread that is given by the synchronization context. The property Globals.UISyncContext holds a synchronization context from the U/I thread and should be specified for any task that will be updating properties that are bound to the U/I.

All the VM classes derive from a base class that implements the IDisposable interface. The base class's Dispose method calls an empty, virtual method called DoDispose. Each VM can override DoDispose to allow cleanup to occur. In this case, the DoDispose method simply unsubscribes from any messages that

the class previously subscribed to. If it had created any child VM's they would be Disposed and nulled at this time.

Device Driver Design

As has been previously discussed, the Device Hub provides a telescope driver, a dome driver, and a focuser driver that other applications can connect to. These drivers sit atop and use the facilities of the Device Hub application. Below is a class diagram of the DeviceHub Telescope driver. The dome and focuser drivers are organized in a similar manner.



The Telescope driver implements the ITelescopeV3 interface and derives from DeviceDriverBase which derives from ReferenceCountedObject. DeviceDriverBase provides convenience methods that are used for data and state validation by the device drivers. It also provides common methods for logging.

The driver communicates with the physical telescope through the same TelescopeManager and TelescopeService objects that the DeviceHub application uses.

The DriverSetupDialogView and DriverSetupDialogViewModel classes are instantiated and used by the Telescope driver's SetupDialog method. This allows an application that is using the driver the ability to choose which driver will be used for the served telescope and to configure that device.