Introduction to Developing ASCOM Drivers

Introduction

ASCOM driver development is not terribly difficult but it isn't the simplest way to start C# or VB.NET development. There seem to be a lot of people who are starting this as almost their first .NET based software development project. These notes are an attempt to help.

Prerequisites and Requirements

I'm going to assume that you have Windows 7 64 bits but will try to note where things are different.

I'm going to assume that you have the following skills:

- Using a PC, including finding and manipulating files and folders and setting permissions.
- Using the Internet, including searching for data and downloading and installing applications.
- Some experience with software development using a modern language such as C, C++, Visual Basic or Delphi.
- Some familiarity with C# development.
- A willingness to learn.

If you don't have these skills then it's a good idea to get some practice first; you will need to be able to do these things and it's not fair on the ASCOM people to expect them to try to teach you these basic software development skills.

I'm not covering basic software development or C#. You need to learn this for yourself.
No amount of tutorials, help or guidance can be a substitute for thinking about what is going on.

I'm going to cover this using C# only, using the C# Visual Studio Express download from Microsoft. You must download and install this. You must also download and install ASCOM Platform 6 and the ASCOM Platform 6 developer components.

If you're using the Visual Basic edition of Visual Studio Express then you will need to work in VB throughout. It isn't possible to mix VB and C# in the express versions. The basics are the same, except that you need to use VB instead of C#. The code examples can be converted using a free code converter such as http://www.developerfusion.com/tools/convert/csharp-to-vb/

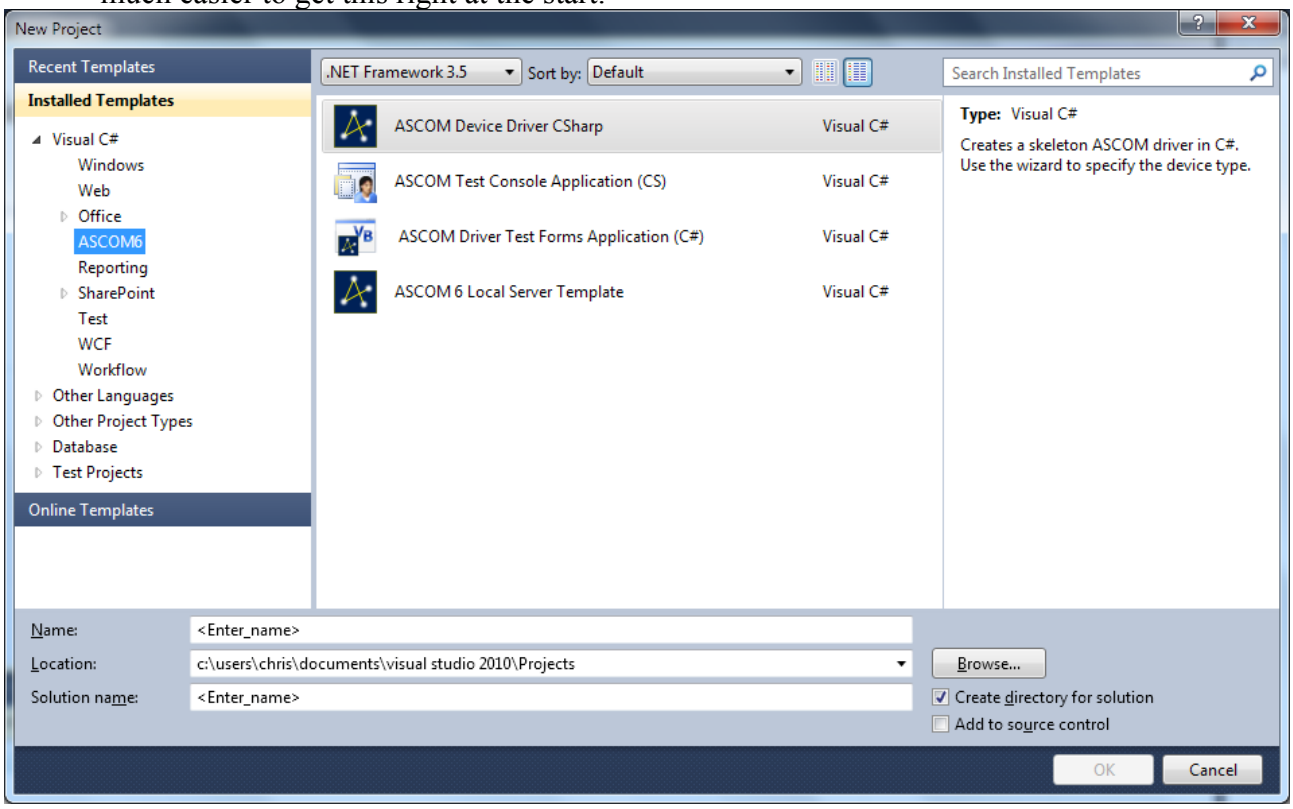
This is based on a simple focuser driver example. The principles apply to any other sort of ASCOM driver, you will need to use the appropriate device in the chooser, such as

```
string id = ASCOM.Dome.Choose("");
ASCOM.Dome dome = new ASCOM.Dome(id);
```

Starting the Driver

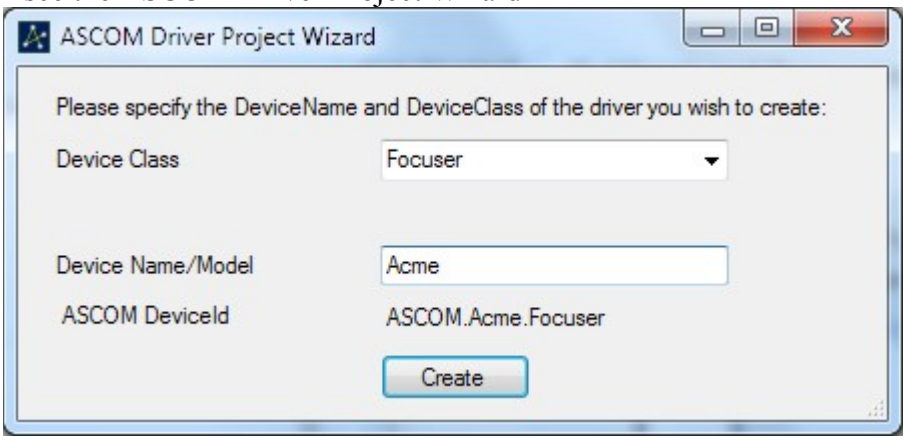
Use the ASCOM templates! They provide a very good way to get the basic functionality implemented.
Start Visual Studio and select New Project....

- Select the ASCOM6 templates and select the ASCOM Device Driver CSharp template.
- Set the project name. Choose this carefully because it will become your driver name. It's much easier to get this right at the start.



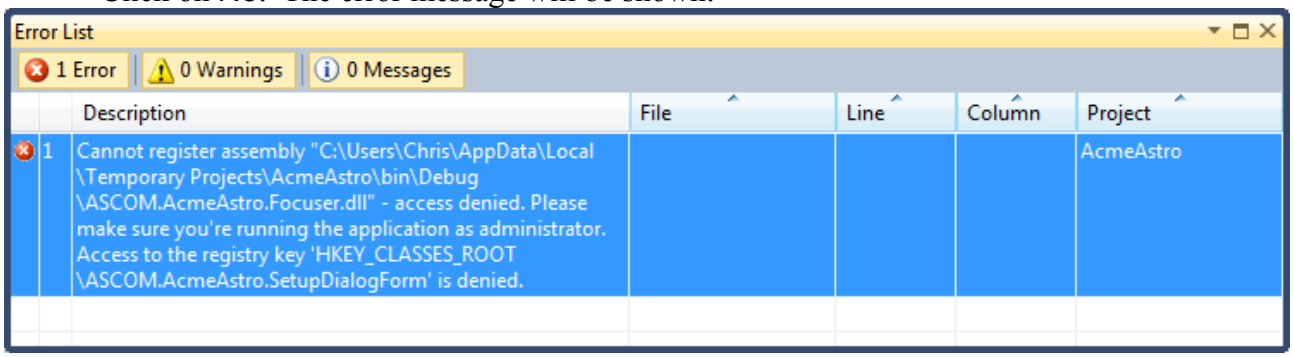
This is the sort of thing you will get. Click on OK.

- You will see the ASCOM Driver Project Wizard



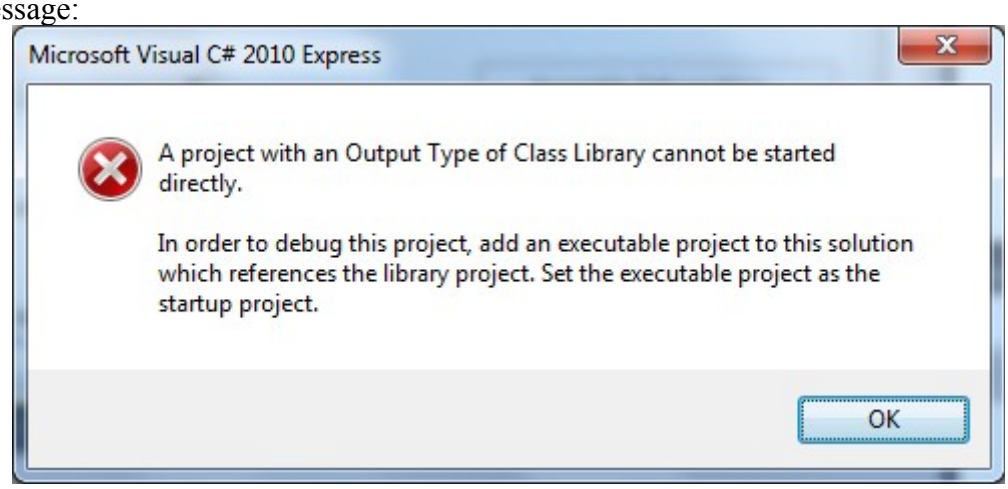
- Set the Device Class and the device Name/Model. Use the same device name as you specified for the project, then click on Create.
- You will get a project and will be shown a help file.
- You need to implement the driver, follow the instructions in the help file.
- Click on the Start Debugging button or press F5.
- You will probably get your first error message:

- Click on No. The error message will be shown:



- The message says that the assembly cannot be registered; this is because on W7 or Vista access to some parts of the registry are restricted to administrators. The good news is that this is only done after the driver has been built successfully. If you're using XP you won't see this.

- Close down Visual Studio, saving your project, and restart it as administrator. The simplest way to do this is bring up the context menu on the short cut to Visual Studio and select Run As Administrator.
- Select your project in the recent projects and build again. Now you get a different error message:



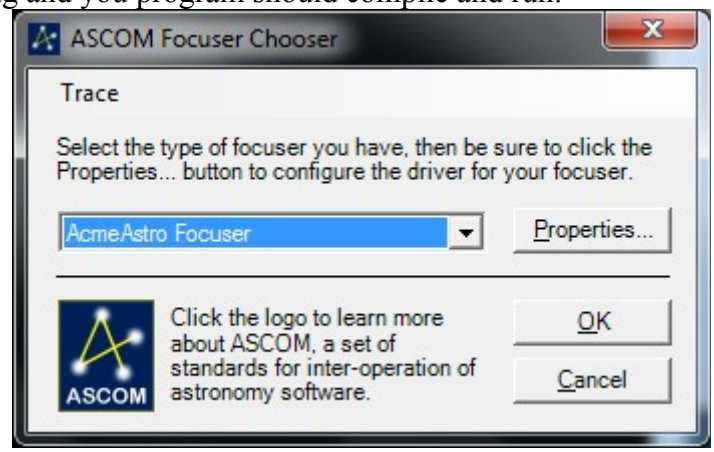
- This means that your driver has been successfully built and registered but it cannot be run because it's a dll and it needs an executable project. The intent was to run a script at this point but that doesn't work in the Express editions.

So you need to create an executable project:

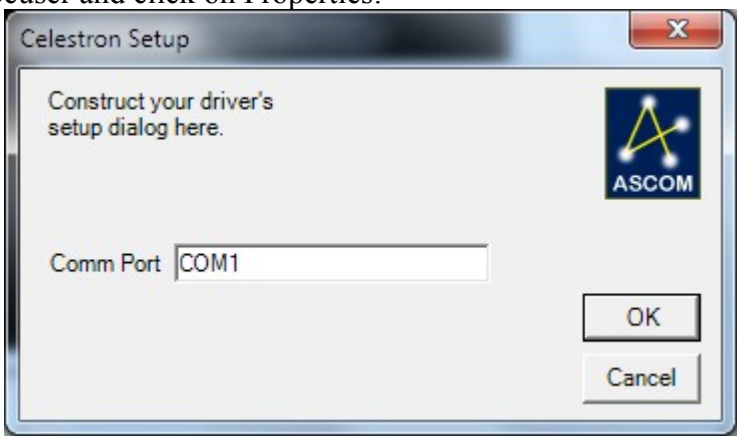
- In the Solution Explorer, select the Solution and use the context menu to select Add - New Project...
- Select the ASCOM6 templates and choose the ASCOM Test Console Application. Set a project name and click on OK.
- You will get the same Wizard as for the driver, select the same driver type and project name and click on Create.

- You will get the skeleton of a driver test application, it can be used to choose and connect to the driver.
- Use the context menu to set FocuserTest to be the Startup Project.

- Click on Debug and you program should compile and run:



- Select your focuser and click on Properties:



- There you are! Your driver is working! It isn't doing much yet but it's a start.

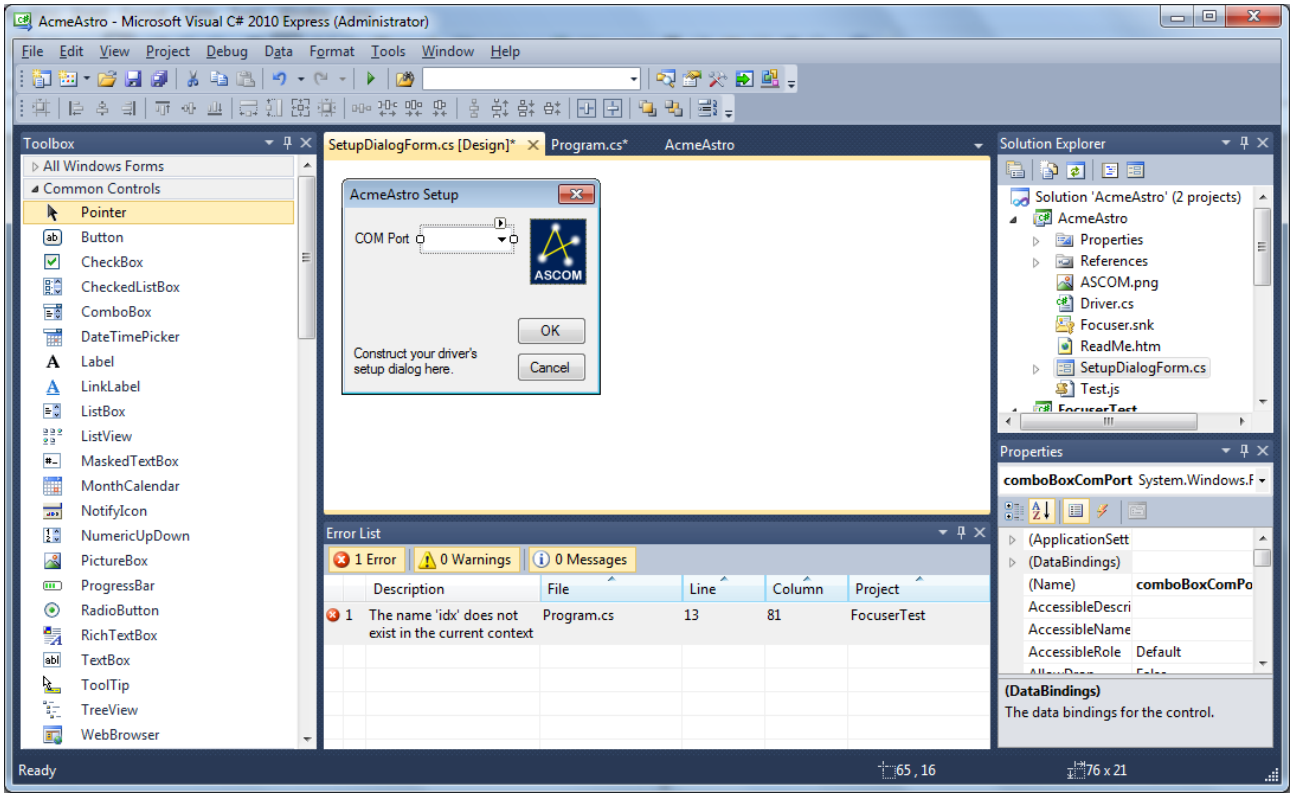
I will cover adding functionality to your new driver next.
Notes:

- If you aren't making a focuser select the driver type you are using everywhere that we have mentioned a focuser.
- I've skimmed a number of basic development concepts, if you aren't familiar with these search on the web. Typing your question into a search engine should come up with something useful.
- Good ways to get help are to highlight the error and press F1 or use the context menu and select Show Error help.
- Another useful way is to search using the error message. It's worth getting several opinions though.
- No amount of advice or tutorials can be a substitute for thought.
- Experiment and expect to throw a lot of code away.

Developing the Setup Dialog

The setup dialog is how the driver properties are set, in this case we are setting a serial port that's used to communicate with the focuser hardware.
The template adds a label and text box with the text box containing the Com port name. This canbe used as is but it may be better to replace the text box with a combo box that contains the available com ports. To do this:

Open the SetupDialogForm in the design mode and replace the text box with a combo box called comboBoxComPort.



Move to the SetupDialogForm code and add code to the SetupDialogForm constructor to fill the combo box with the Com port names:

```
comboBoxComPort.Items.Clear();
using (ASCOM.Utilities.Serial serial = new Utilities.Serial())
{
    foreach (var item in serial.AvailableCOMPorts)
    {
        comboBoxComPort.Items.Add(item);
    }
}
```

Set the current Com port.

```
if (! String.IsNullOrEmpty(Properties.Settings.Default.CommPort))
{
    comboBoxComPort.SelectedItem = Properties.Settings.Default.CommPort;
}
```

Save the com port name in the CmdOkClick event handler:

```
private void CmdOkClick(object sender, EventArgs e)
{
     Properties.Settings.Default.CommPort = (string)comboBoxComPort.SelectedItem;
      Close();
}
```

That's enough to be able to select and save the com port.

Setting up the driver

Start by going through the public properties setting the ones that aren't implemented so they throw the ASCOM.PropertyNotImplemented exception, here's an example:

```
public bool TempComp
{
    get { throw new ASCOM.PropertyNotImplementedException("TempComp", false); }
    set { throw new ASCOM.PropertyNotImplementedException("tempComp", true); }
}
```

Set properties such as Name, Description and DriverInfo so they return suitable strings:

```
public string Name
{
    get { return driverDescription; }
}
```

Set any other capabilities appropriately. You will need to think about this because it's where you start to implement the capabilities of your hardware. It's a good idea to have the ASCOM device interface specification open so you can refer to it as you go.

Note: ALL the properties and methods must have the System.NotImplemented exception replaced by something, even if all that's required is an ASCOM.PropertyNotImplemented exception.

It's a good idea to set things so it does very little, then extend things once you have things working.

Connecting to the Hardware:

This happens in the Set Connected property:
Add a field to hold the serial port:

```
private Serial serialPort;
```

I won't describe everything that's done, here is a pretty minimal connect property

```
public bool Connected
{
    get
    {
        if (serialPort == null)
            return false;
        return serialPort.Connected;
    }
    set
    {
        if (value)
        {
            // check if we are connected, return if we are
            if (serialPort != null && serialPort.Connected)
                return;
            // get the port name from the profile
            string portNameProperties.Settings.Default.CommPort;
            if (string.IsNullOrEmpty(portName))
            {
                // report a problem with the port name
                throw new ASCOM.NotConnectedException("no Com port selected");
            }
            // try to connect using the port
            try
            {
                serialPort = new Serial();
                serialPort.PortName = portName;
                serialPort.Speed = SerialSpeed.ps9600;
                serialPort.Connected = true;
            }
            catch (Exception ex)
```

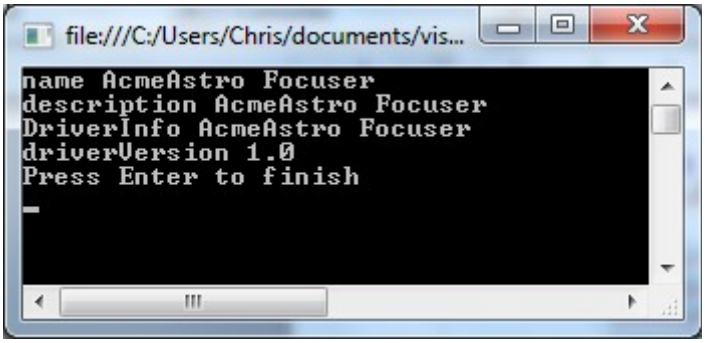
```
                    {
                        // report any error
                        throw new ASCOM.NotConnectedException("Serial port connection
error", ex);
                    }
                }
                else
                {
                    // disconnect
                    if (serialPort != null && serialPort.Connected)
                        serialPort.Connected = false;
                }
            }
        }
```

Additional things would be to send a command to the hardware to check that it's there.

It's now possible to extend the focuser test application to check a few things:

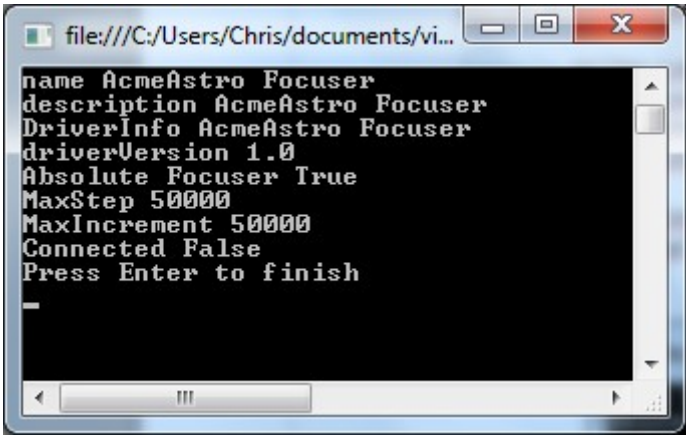
```
        static void Main(string[] args)
        {
            string id = ASCOM.DriverAccess.Focuser.Choose("");
            ASCOM.DriverAccess.Focuser focuser = new ASCOM.DriverAccess.Focuser(id);

            Console.WriteLine("name " + focuser.Name);
            Console.WriteLine("description " + focuser.Description);
            Console.WriteLine("DriverInfo " + focuser.DriverInfo);
            Console.WriteLine("driverVersion " + focuser.DriverVersion);

            Console.WriteLine("Press Enter to finish");
            Console.ReadLine();
        }
```

debug and you get:



Add a few more things:

```
            Console.WriteLine("Absolute Focuser " + focuser.Absolute);
            Console.WriteLine("MaxStep " + focuser.MaxStep);
            Console.WriteLine("MaxIncrement " + focuser.MaxIncrement);
            Console.WriteLine("Connected " + focuser.Connected);
```



Keep adding tests as you add more functionality:

```
            focuser.Connected = true;
            Console.WriteLine("Connected " + focuser.Connected);
```

If you get errors set breakpoints and debug to see what's happening.
You may find that you can't debug from the test program to the driver, the simplest way is to set breakpoints in the driver a run to those. **It's important to make sure that both the driver and the test application are using the same version of .NET, if they aren't you can't debug into the driver.**

Now to send some commands:
A good way to implement sending commands is to put the command in the CommandString method:

```
        public string CommandString(string command, bool raw)
        {
            if (!this.Connected)
                throw new ASCOM.NotConnectedException();
            serialPort.ClearBuffers();
            serialPort.Transmit(command);
            return serialPort.ReceiveTerminated("#");
        }
```

This assumes that every command is terminated with a '#' character AND that this character cannot appear in a message.
You may need to add message terminators to the command.
Every method that sends a command calls CommandString, for example:

```
        public int Position
        {
            get
            {
                // position command is P#, returns nnnn#
                string ret = CommandString("P#", true);
                return int.Parse(ret);
            }
        }
```

One thing that trips people up is that Visual Studio re-registers the driver every time it's compiled so you need to run the setup dialog to select the COM port each time, otherwise you get an error in connected because the com port name is not set.

I can't run this fully because I don't have hardware that implements this protocol. Obviously a real driver would need to implement a real protocol.

There are a number of things missing:
Very little error checking.
No checks on the validity of data.

Here's the complete driver:

```
using System;
using System.Collections;
using System.Globalization;
using System.Runtime.InteropServices;
using ASCOM.DeviceInterface;
using ASCOM.Utilities;

namespace ASCOM.AcmeAstro
{
    //
    // Your driver's ID is ASCOM.AcmeAstro.Focuser
```

```
//
// The Guid attribute sets the CLSID for ASCOM.AcmeAstro.Focuser
// The ClassInterface/None addribute prevents an empty interface called
// _Focuser from being created and used as the [default] interface
//
[Guid("d762e5e0-f093-4852-b081-8e0206192c45")]
[ClassInterface(ClassInterfaceType.None)]
[ComVisible(true)]
public class Focuser : IFocuserV2
{
    #region Constants
    //
    // Driver ID and descriptive string that shows in the Chooser
    //
    internal const string driverId = "ASCOM.AcmeAstro.Focuser";
    // TODO Change the descriptive string for your driver then remove this line
    private const string driverDescription = "AcmeAstro Focuser";
    private Serial serialPort;
    #endregion

    #region ASCOM Registration
    //
    // Register or unregister driver for ASCOM. This is harmless if already
    // registered or unregistered.
    //
    private static void RegUnregASCOM(bool bRegister)
    {
        using (var p = new Profile())
        {
            p.DeviceType = "Focuser";
            if (bRegister)
                p.Register(driverId, driverDescription);
            else
                p.Unregister(driverId);
        }
    }

    [ComRegisterFunction]
    public static void RegisterASCOM(Type t)
    {
        RegUnregASCOM(true);
    }

    [ComUnregisterFunction]
    public static void UnregisterASCOM(Type t)
    {
        RegUnregASCOM(false);
    }
    #endregion

    #region Implementation of IFocuserV2

    public void SetupDialog()
    {
        using (var f = new SetupDialogForm())
        {
            f.ShowDialog();
        }
    }

    public string Action(string actionName, string actionParameters)
    {
        throw new ASCOM.MethodNotImplementedException("Action");
    }
```

```
public void CommandBlind(string command, bool raw)
{
    throw new ASCOM.MethodNotImplementedException("CommandBlind");
}

public bool CommandBool(string command, bool raw)
{
    throw new ASCOM.MethodNotImplementedException("CommandBool");
}

public string CommandString(string command, bool raw)
{
    if (!this.Connected)
        throw new ASCOM.NotConnectedException();
    serialPort.ClearBuffers();
    serialPort.Transmit(command);
    return serialPort.ReceiveTerminated("#");
}

public void Dispose()
{
    throw new System.NotImplementedException();
}

public void Halt()
{
    // halt command is H#
    CommandString("H#", false);
    //throw new System.NotImplementedException();
}

public void Move(int value)
{
    // move command is Mnnnn# where nnn is the target position
    CommandString(string.Format("M{0}#", value), false);
    //throw new System.NotImplementedException();
}

public bool Connected
{
    get
    {
        if (serialPort == null)
            return false;
        return serialPort.Connected;
    }
    set
    {
        if (value)
        {
            // check if we are connected, return if we are
            if (serialPort != null && serialPort.Connected)
                return;
            // get the port name from the profile
            string portName;
            using (ASCOM.Utilities.Profile p = new Profile())
            {
                p.DeviceType = "Focuser";
                portName = p.GetValue(driverId, "ComPort");
            }
            if (string.IsNullOrEmpty(portName))
            {
                // report a problem with the port name
                throw new ASCOM.NotConnectedException("no Com port selected");
            }
```

```
                    // try to connect using the port
                    try
                    {
                        serialPort = new Serial();
                        serialPort.PortName = portName;
                        serialPort.Speed = SerialSpeed.ps9600;
                        serialPort.Connected = true;
                    }
                    catch (Exception ex)
                    {
                        // report any error
                        throw new ASCOM.NotConnectedException("Serial port connection
error", ex);
                    }
                }
                else
                {
                    // disconnect
                    if (serialPort != null && serialPort.Connected)
                        serialPort.Connected = false;
                }
            }
        }

        public string Description
        {
            get { return driverDescription; }
        }

        public string DriverInfo
        {
            get { return driverDescription; }
        }

        public string DriverVersion
        {
            get
            {
                Version version =
System.Reflection.Assembly.GetExecutingAssembly().GetName().Version;
                return String.Format(CultureInfo.InvariantCulture, "{0}.{1}", version.Major,
version.Minor);
            }
        }

        public short InterfaceVersion
        {
            get { return 2; }
        }

        public string Name
        {
            get { return driverDescription; }
        }

        public ArrayList SupportedActions
        {
            get { return new ArrayList(); }
        }

        public bool Absolute
        {
            get { return true; }
        }
```

```
        public bool IsMoving
        {
            get
            {
                // ismoving command is ?#, returns 1# if moving, 0# if not
                string ret = CommandString("?#", true);
                return (ret == "1");
            }
        }

        // use the V2 connected property
        public bool Link
        {
            get { return this.Connected; }
            set { this.Connected = value; }
        }

        public int MaxIncrement
        {
            get { return 50000; }
        }

        public int MaxStep
        {
            get { return 50000; }
        }

        public int Position
        {
            get
            {
                // position command is P#, returns nnnn#
                string ret = CommandString("P#", true);
                return int.Parse(ret);
            }
        }

        public double StepSize
        {
            get { throw new ASCOM.PropertyNotImplementedException("StepSize", false); }
        }

        public bool TempComp
        {
            get { throw new ASCOM.PropertyNotImplementedException("TempComp", false); }
            set { throw new ASCOM.PropertyNotImplementedException("tempComp", true); }
        }

        public bool TempCompAvailable
        {
            get { throw new ASCOM.PropertyNotImplementedException("TempCompAvailable",
false); }
        }

        public double Temperature
        {
            get { throw new ASCOM.PropertyNotImplementedException("Temperature", false); }
        }

        #endregion
    }
}
```

This example code was originally developed using the Platform 6 templates so there may be slight

changes, in particular the SP1 templates have added code to help with checking that the hardware is connected.
It may be better to separate the hardware control into a separate class. This will allow more flexibility such as implementing different hardware control whilst using the same interface code. One good use of this is to implement a hardware simulator class.

Making an Installer

The Installer Script Generator that's available from the Developer Tools is a good way to get an installer generated. It uses the Inno Setup.
The Install script generator generates an Inno Install script, to use it:

- Start the script generator
- Set the Technology. For the driver described here it will be .NET assembly (dll).
- Set the Short Name, usually something of the form Name driver type such as Acme Focuser
- Set the driver type – in this case Focuser
- Set the version, the most significant version number should reflect the platform version, so currently 6.n.n
- Use the Source Folder browse button to navigate to the source folder for the driver.
- Set the Main driver file and Readme file. You must have both so it could be a good idea to write a readme file for the users.
- Check the option to install the source code if you plan to provide this to your users.
- Set the driver developer's name and a suitable email address.
- Note that every filed must be filled in before the Save button is made active.
- Click on Save. This will generate the inno script file, save it to the driver source and start Inno setup with the script file loaded.

Check The Script! We have done what we can to make it a good installer but you are responsible for what it does. The most likely problems are:

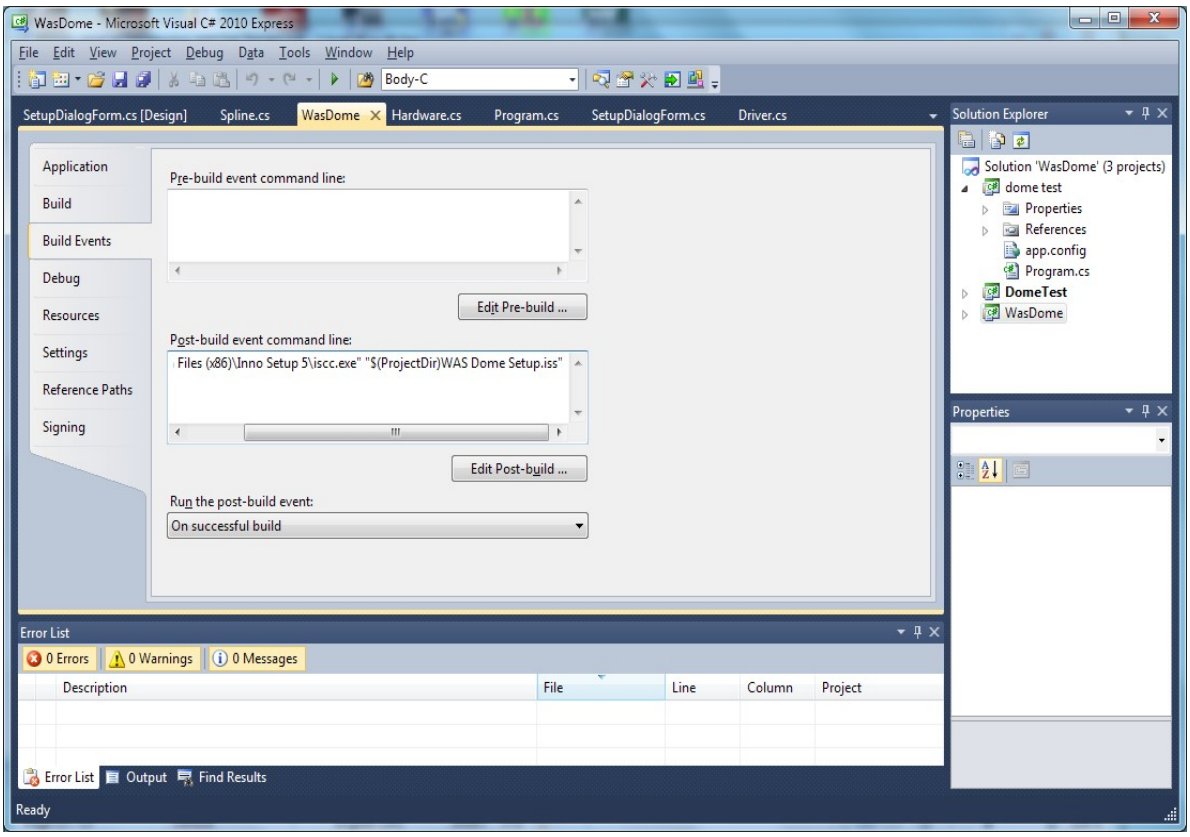
- Incorrect file paths. The driver file is assumed to be in {source folder}\bin\Release but this may not be the case, in particular it may be in bin\Debug.
- There is a bug in the version check. The version check line should be:
  `if (U.IsMinimumRequiredVersion(6,0)) then    // this will work in all locales`
  it needs to be (6, 0), not (6.0)

Make any other changes you need to.
Compile the script, then test it to ensure that the driver is installed correctly.

Generating the Installer File Automatically

It's possible to generate the installation file automatically by running the install script as part of the build process. To do this:

- Open the driver properties and select the Build events tab.
- Add the command to run the inno setup script to the Post build event command line: for example
  `"C:\Program Files (x86)\Inno Setup 5\iscc.exe" "$(ProjectDir)WAS Dome Setup.iss"`



The path to the inno setup program and the name of the iss file will need to be changed of course.

- This will now regenerate the installer every time the program is built.

Chris Rowland August 2011, March 2012