Generating Multiple ASCOM Drivers

Introduction

There is a need for more than one driver of the same type to be used at the same time. This is required if there are two cameras, focusers or filter wheels on the system, for example main and guide cameras.

At present the only way this can be done is by using devices from different manufacturers, so they have different drivers.

This note describes a way this can be done using the LocalServer pattern. It isn't a full description because a lot of detail will depend on how your drivers are controlled but should give you a good starting position.

Summary

- Two (or more) differently named driver dlls are generated using the same source code.
- These drivers have different guids, progIds and Served Class names applied by setting these as Attributes of the driver class and using conditional compilation to select the appropriate ones for each driver instance.

This is complex to describe but the additional steps required to implement it are relatively simple, they are outlined below.

Generating multiple drivers:

- Start by developing the driver using the appropriate template.
- Convert this driver to a Local Server driver using the LocalServer template and following the instructions and guide lines for this.
- Ensure that the ProgId and ServedClassName Attributes are used to specify the ProgId and the class name as shown in the Chooser.
- When you have this working you can add a second driver:
- Make a copy of the driver project file in the same project folder, for example AcmeCamera.csproj becomes AcmeCamera-2.csproj
- Add this project to the solution.
- Open the new project properties and:
    - In the Application tab change the Assembly name, so ASCOM.Acme.Camera becomes ASCOM.Acme2.Camera. This will cause this instance of the driver to be built to the file ASCOM.Acme2.Camera.dll.
    - In the Build tab add a Conditional compilation symbol, e.g. Camera2
- In the driver.cs file use this symbol to set different ServedClassName, ProgId and Guid values for the two instances of the driver, the class will look a bit like this:

```
namespace ASCOM.Acme
#if Camera2
    [Guid("55F55852-04F7-43E2-9A54-38B99DB612F7")]
    [ProgId("ASCOM.Acme2.Camera")]
    [ServedClassName("Acme Camera 2")]
#else
    [Guid("b3d08b43-5ed1-480b-86fc-7ff5ca40bf3f")]
    [ProgId("ASCOM.Acme.Camera")]
    [ServedClassName("Acme Camera")]
#endif
    [ClassInterface(ClassInterfaceType.None)]
```

```
public class Camera : ReferenceCountedObjectBase,
                      ICameraV2
{
```

- modify the constructor so that the driver Id is set from the ProgId attribute:

```
public Camera()
{
      driverId = Marshal.GenerateProgIdForType(this.GetType());
 }
```

  You may find that this is already done with the latest template.
- Build the solution; you should have two instances of the driver dll – ASCOM.Acme.Camera.dll and ASCOM.Acme2.Camera.dll.
- Register both drivers, this can be done by running the server with the /register command line option.
- Try it. The Chooser should show two devices and they can be selected and configured separately.
- An application should be able to use either driver as it's ProgId, load that one and get the appropriate device with it's setup.

Handling Multiple Connections to Multiple Devices

It is important that the connections to the devices are handled correctly, both with multiple connections to one device and connections to the different devices. The exact way this will work will vary depending on how the connection is to be made but the basic principle is that the connection to the device is only lost when the last driver disconnects.
This can be done by keeping a count of the number of connections for each device, incrementing the count when a connection is made and decrementing it when the driver disconnects. The connection is only really disconnected when the count goes to zero.
It's also necessary to ensure that each device can be identified. There will be some property of the device that can be used to do this, such as the COM port name, the USB ID, IP address and so on. This can be handled in the SharedResources class, some examples are given in the latest LocalServer template.

Shared Code

The only code that must be shared is the code that counts the number of connections to each device. It may be better to put all the code that's common in a single class. This can include the device interface code and the Setup Dialog. It is important that the device and driver are identified, this can be done by specifying the driverId in the class constructor.

Installer

Add the additional driver to the [Files] list in the installer. You can use a task to make loading the additional drivers optional.

Multiple Drivers without using LocalServer

It should be possible to use the principle of generating multiple drivers without using the LocalServer. This hasn't been tested but the process should be:

- Make a driver in the usual way.
- Make a copy of the project file and modify it as described above
- Set up different Guid, ProgId and servedClassName attributes using conditional compilation as described above.
- Ensure that the ServedClassName and ProgId attributes are used to register for ASCOM.

That should be it, the drivers should be registered separately, both for COM and ASCOM, and can be chosen and used independently. There are a couple of problems with this:

- Only one connection can be made to each device.
- The device may only be able to connect to 32 or 64 bit applications because the device driver is 32 or 64 bits only.