Making a Local Server based Driver

Introduction

The LocalServer template provides a way to allow multiple driver types to use the same hardware and also allow multiple applications to connect to the same driver. It can also allow multiple drivers of the same type to be generated. It will also allow a driver to communicate across the 32 to 64 bit boundary. This point is important; many devices will have low level driver software that's only 32 (or 64) bit compatible and that will force the driver to only operate in that mode. The Local server will allow it to work with applications that work in the other mode.

Difference from the Platform 5 version of the local \server

- The driver dlls and the local server executable are expected to be in the same folder, with nothing else. This would normally be a sub folder of the ASCOM driver folder, such as .../ASCOM/Camera/Acme.
- The “friendly” name of the driver, displayed in the chooser, is specified using the ServedClassName attribute. This is provided as part of Platform 6 but can be defined independently if the driver has to work with earlier platform versions.
- The ProgId is also specified using an attribute, this is to aid in making a driver that can handle multiple devices.

Implementing a Local Server Driver:

Here's a guide to generating drivers based on the LocalServer model, the examples assume that the Acme company is generating these drivers.

- Start by generating a normal dll driver using the appropriate template. There are some things that will help with making this work:
    - Create the driver project in a sub folder of the solution folder. VS does this automatically.
    - Make sure the namespace and driver name are of the format ASCOM.manufacturer and ASCOM.manufacturer.driver, e.g ASCOM.Acme and ASCOM.Acme.Camera.
    - Put any device specific code in it's own class.
    - Build and debug the driver.
    - It's a lot easier debugging a dll driver because it can run in the same process as the test application, this makes setting breakpoints easier.
    - Do this for all the driver types you need to so, arranging things so they are all in the same solution.
- When you have as much done as possible create a local server using the template.
- Follow the instructions in the LocalServer readme file, in particular make sure that the namespace is the same as the one used for the drivers.
- You should find that the section about renaming things does not apply because everything is already named correctly.
- Make sure that the driver dlls are copied to the same folder as the local server. This can be done with a post build step as described in the readme or by modifying the build location.
- Move the hardware control to the LocalServer SharedResources class; note this it needs to be static so that all the devices use a single copy of the connection to the hardware.
- Put locks in place so that only one driver can get access to the hardware at a time; the exact way will depend on the hardware but it's almost certain that sending a command and getting it's response must be atomic and will need something to prevent a second command being

sent before the reply from the first one has been received.
- Check that the drivers are being copied to the local server bin folder.
- Set the startup project to the local server, set the command line argument to /register and run it.
- Check that the drivers have been registered.
- Test. You will probably find that the LocalServer is running in a different process and it's not loaded.
- Use Debug – Attach to Process to attach the debugger to the local server.
- If you are trying to debug the startup process then adding the line
  ```
        MessageBox.Show("wait");
  ```
  at a suitable place will force things to wait while you attach the debugger.
- You will need to handle counting and referencing to items that are already connected, this can be done by putting common code in the Shared Resources file and/or setting up your own handling of common objects and handles.
- Make sure that your hardware interfaces are disposed of correctly, if you don't the LocalServer may not close when all the drivers close.
- Check that everything closes even if you don't disconnect or if a driver crashes.

I hope this helps, it's almost impossible to give chapter and verse on this because each implementation will be different.

Example:

Here's a class that handles devices where there's a handle to the hardware and the device is identified by a string:

```
public static class SharedResources
{
    private static Dictionary<string, ConnectedDevice> connectedDevices = new
Dictionary<string, ConnectedDevice>();

    /// <summary>
    /// List of connected devices, keyed by a string ID
    /// </summary>
    public static IDictionary<string, ConnectedDevice>  ConnectedDevices
    {
        get { return connectedDevices; }
    }

    /// <summary>
    /// Gives the camera handle and the number of connections
    /// </summary>
    public class ConnectedDevice
    {
        public ConnectedDevice(IntPtr handle)
        {
            this.handle = handle;
            this.connections = 1;
        }
        public IntPtr handle;
        public int connections;
    }
}
```

A new device is added like this:

```
SharedResources.ConnectedDevices.Add(usbSerial,
          new SharedResources.ConnectedDevice(handle));
```

an existing device is used like this:

```
// handle the case where we are already connected to this device by looking for it's
serial id in
// the connected devices.
// if it is use the existing camera interface (if you can)
SharedResources.ConnectedDevice cc;
if (SharedResources.ConnectedDevices.TryGetValue(name, out cc))
{
    hDevice = cc.handle;
    cc.connections++;
}
```

And when disconnect is called:

```
// find the camera we are connected to
SharedResources.ConnectedDevice cc;
if (SharedResources.ConnectedDevices.TryGetValue(name, out cc))
{
   cc.connections--;
   if (cc.connections <= 0)
   {
       // we only disconnect if there are no more connections
       acmeDevice.Disconnect(handle);
       SharedResources.ConnectedCameras.Remove(name);
   }
}
```

Similar code is used for dispose.
This code is intended to support multiple devices of the same type; this is covered in a different document.

Support

This is fairly advanced C# programming so it's quite likely that you will have problems. Your best resource is the internet. A search using some part of a problem report such as an error message is quite likely to give lots of information. Reliable sites are the Microsoft documentation sites and Stack Overflow.
The ASCOM-Talk Yahoo group is also a good source of information and advice. Please bear in mind a few things:

- Most questions have been answered before. It's worth searching for an answer before posting.
- Ask a specific question, and give full information. Saying “My driver doesn't work” is unlikely to get a useful reply.
- Remember that the people who you are expecting to answer are just as busy as you so make their job as easy as you can. Don't expect them to do a web search for you.
- We can't teach you programming or C# syntax. We aren't set up for this. There are plenty of books, courses and web sites that can help.