Alpaca Pi software suite

Table of Contents

AlpacaPi Overview

AlpacaPi is a suite of software that implements the Alpaca protocol for control of astronomy telescopes, cameras etc. *AlpacaPi* was designed to run primarily on Raspberry-Pi hardware but will run on any Linux system.

Alpaca is a network protocol based on HTML and JSON for sending commands and data back and forth between client programs and device drivers (servers). These can run on the same computer or on separate computers.

The *AlpacaPi* suite is written in C++. It takes advantage of the C++ inheritance to simplify the implementation. The top level takes care of all of the network communications and parses the incoming commands. It then figures out which device the command is for and calls the appropriate handler. With the exception of the management driver, only level 3 instances are created. (see class structure below)

AlpacaPi implements all of the drivers in one application. On startup, it looks to see what is attached, for example if it discovered 2 ZWO cameras and 1 ATIK CameraDriver, it would create 3 camera drivers (0,1,2) that could be used interdependently. I have actually done this and it works just fine.

In my normal setup, one Raspberry-Pi is connected to a ZWO camera, a Moonlite NiteCrawler focuser and a ZWO filterwheel. In this case, it would create 4 devices, a camera, a focuser, a rotator, and a filterwheel. Actually it would create 5 devices because every driver needs a managment driver. For those that are not familiar with the Moonlite NiteCrawler focusers, they have a focuser and rotator in the same unit so both a focuser and rotator device are created.

This is the output of my discovery dump program for this example.

```
192.168.1.164   :6800   Camera         ZWO ASI1600MM Pro      0 AlpacaPi - V0.3.7-beta build #71
192.168.1.164   :6800   Filterwheel    ZWO EFW-8              0 AlpacaPi - V0.3.7-beta build #71
192.168.1.164   :6800   Focuser        NiteCrawler Focuser    0 AlpacaPi - V0.3.7-beta build #71
192.168.1.164   :6800   Management     ManagementDriver       0 AlpacaPi - V0.3.7-beta build #71
192.168.1.164   :6800   Rotator        NiteCrawler Rotator    0 AlpacaPi - V0.3.7-beta build #71
```

The "0" means the device number, in each case there is only one of that class of device so the number is always 0.

AlpacaPi Class Structure

Top level	Device type level	Actual hardware	Comments

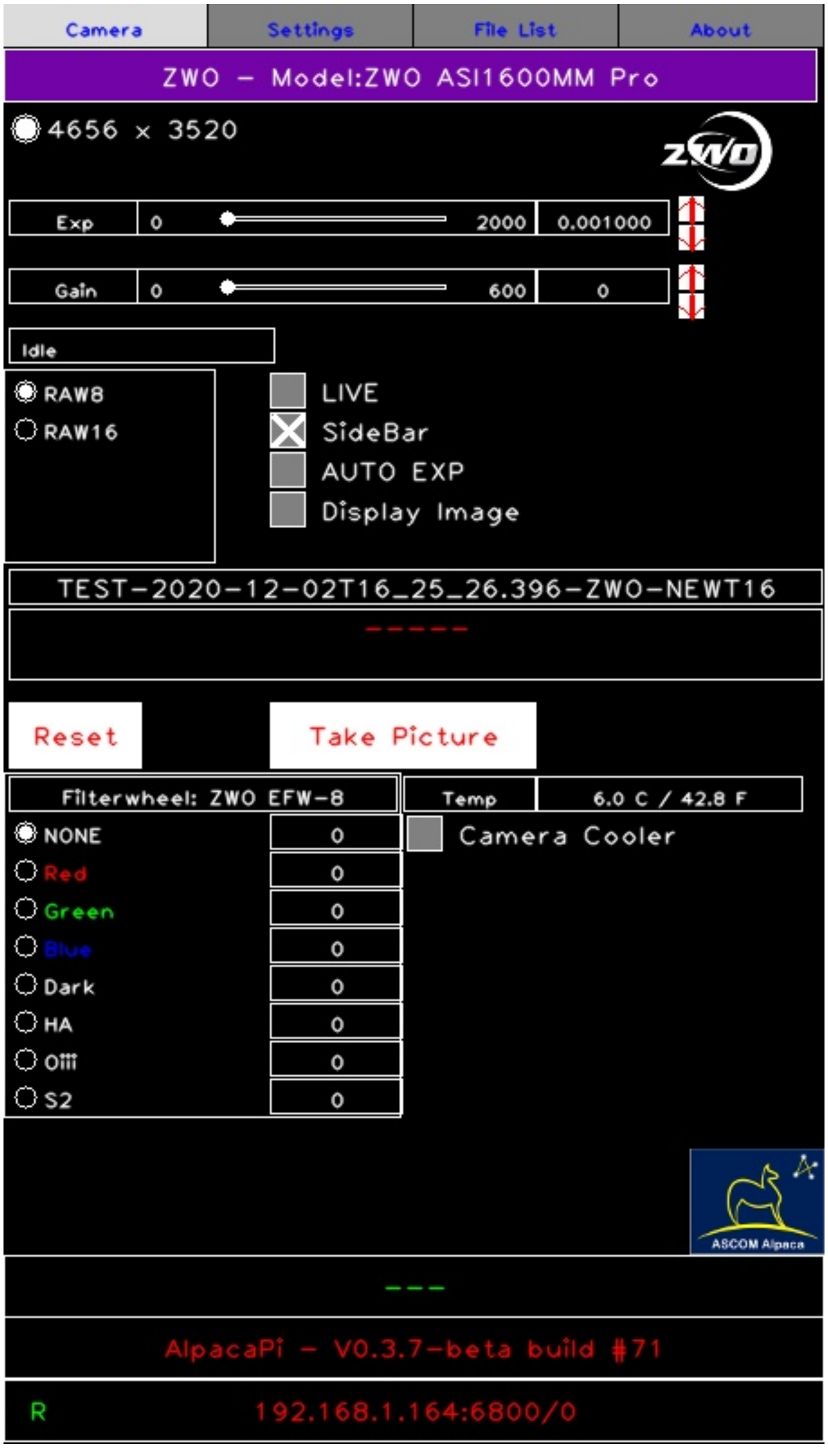
		control	
alpacadriver			
	cameradriver		
		cameradriver_ASI	ZWO cameras
		cameradriver_ATIK	ATIK cameras
		cameradriver_FLIR	FLIR grasshopper series cameras
		cameradriver_QHY	QHY cameras
		cameradriver_SONY	SONY A7 R IV DLSR camera
		cameradriver_TOUP	TOUPTECH cameras
	domedriver		
		domedriver_rpi	
		domedriver_ror_rpi	
	filterwheeldriver		
		filterwheeldriver_ZWO	ZWO filterwheel
	focuserdriver		
		focuserdriver_nc	Moonlite focuser, NiteCrawler and High Speed Stepper
	managementdriver		
		The management driver does not have any hardware to control, so a 3rd layer is not required	
	obsconditionsdriver		
		obsconditionsdriver_rpi	
	rotatordriver		
		rotatordriver_nc	Moonlite NiteCrawler
	switchdriver		
		switchdriver_rpi	Utilizing a Raspberry Pi Relay board

Client applications

Once you have drivers running, you need an application to control it. The drivers have no direct user interface are designed to be only talked to through a network connection.

As stated above, a client application needs to know what device to talk to. You could specify the exact IP address and port if you know it, or use the discovery protocol. One thing to note, the discovery protocol only works on your local subnet. For most people this is fine, but if you want to control a REMOTE device, you will need to specify the IP and port.

The client would issue a discovery protocol query and look for the devices it knows how to talk to. In this example we are running my camera gui program. As you can see from the client dump above, it has found a camera at ip 192.168.1.164 on port 6800. It also noticed that there was a filter wheel on the same device. This camera application is smart enough to know how to talk to both of them. If a filter wheel does not exist, that interface is not displayed as in the 2nd image.



ZWO – Model:ZWO ASI1600MM Pro

● 4656 x 3520

Exp	0	2000	0.001000
Gain	0	600	0

Idle

● RAW8
○ RAW16

□ LIVE
☒ SideBar
□ AUTO EXP
□ Display Image

TEST-2020-12-02T16_25_26.396-ZWO-NEWT16

Filterwheel: ZWO EFW-8	
● NONE	0
○ Red	0
○ Green	0
○ Blue	0
○ Dark	0
○ HA	0
○ oiii	0
○ S2	0

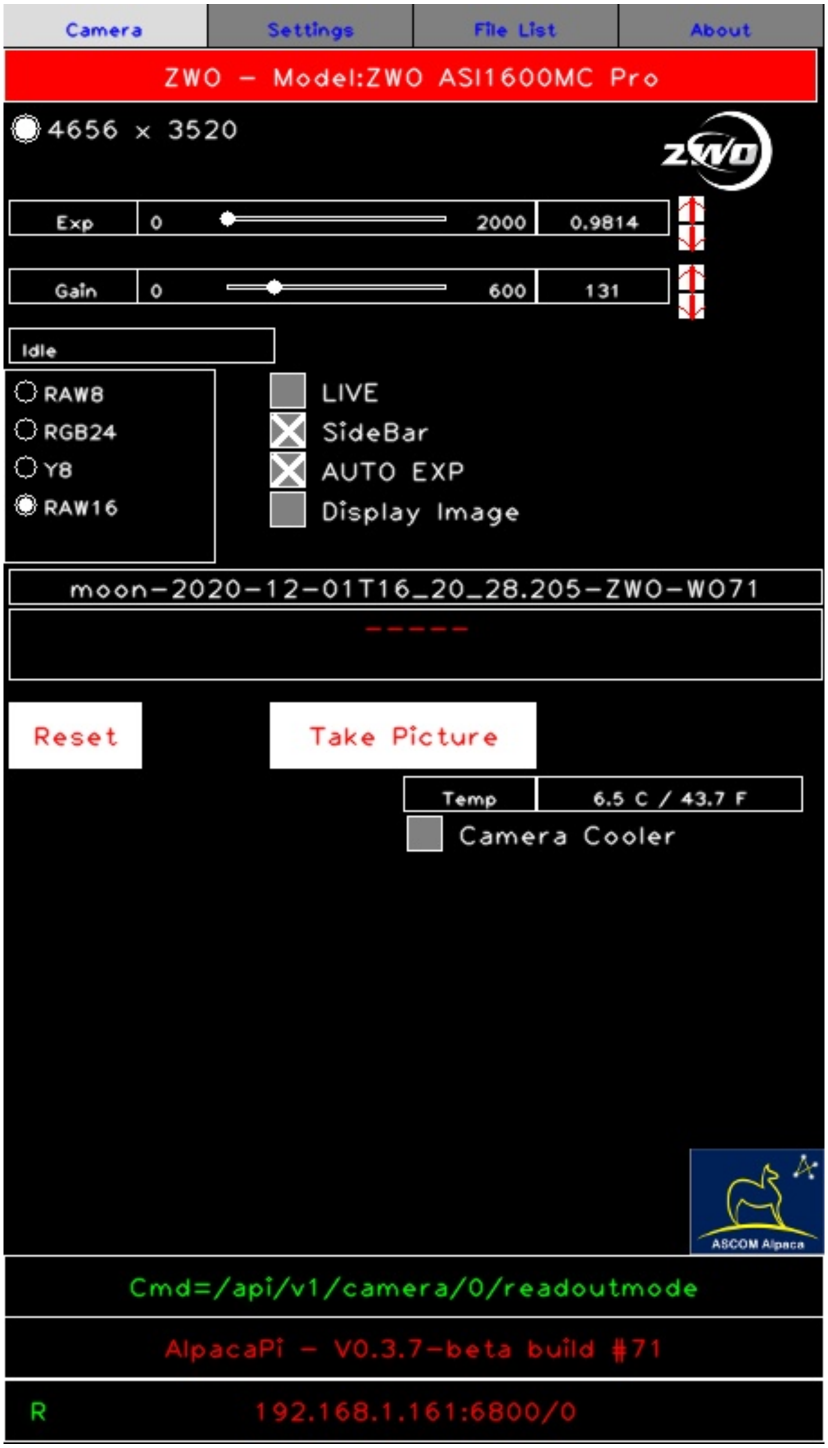
Temp 6.0 C / 42.8 F

□ Camera Cooler

ASCOM Alpaca

AlpacaPi – V0.3.7-beta build #71

R 192.168.1.164:6800/0



ZWO - Model:ZWO ASI1600MC Pro

● 4656 x 3520

Exp 0 2000 0.9814

Gain 0 600 131

Idle

○ RAW8
○ RGB24
○ Y8
● RAW16

□ LIVE
☒ SideBar
☒ AUTO EXP
□ Display Image

moon-2020-12-01T16_20_28.205-ZWO-W071

Temp 6.5 C / 43.7 F

□ Camera Cooler

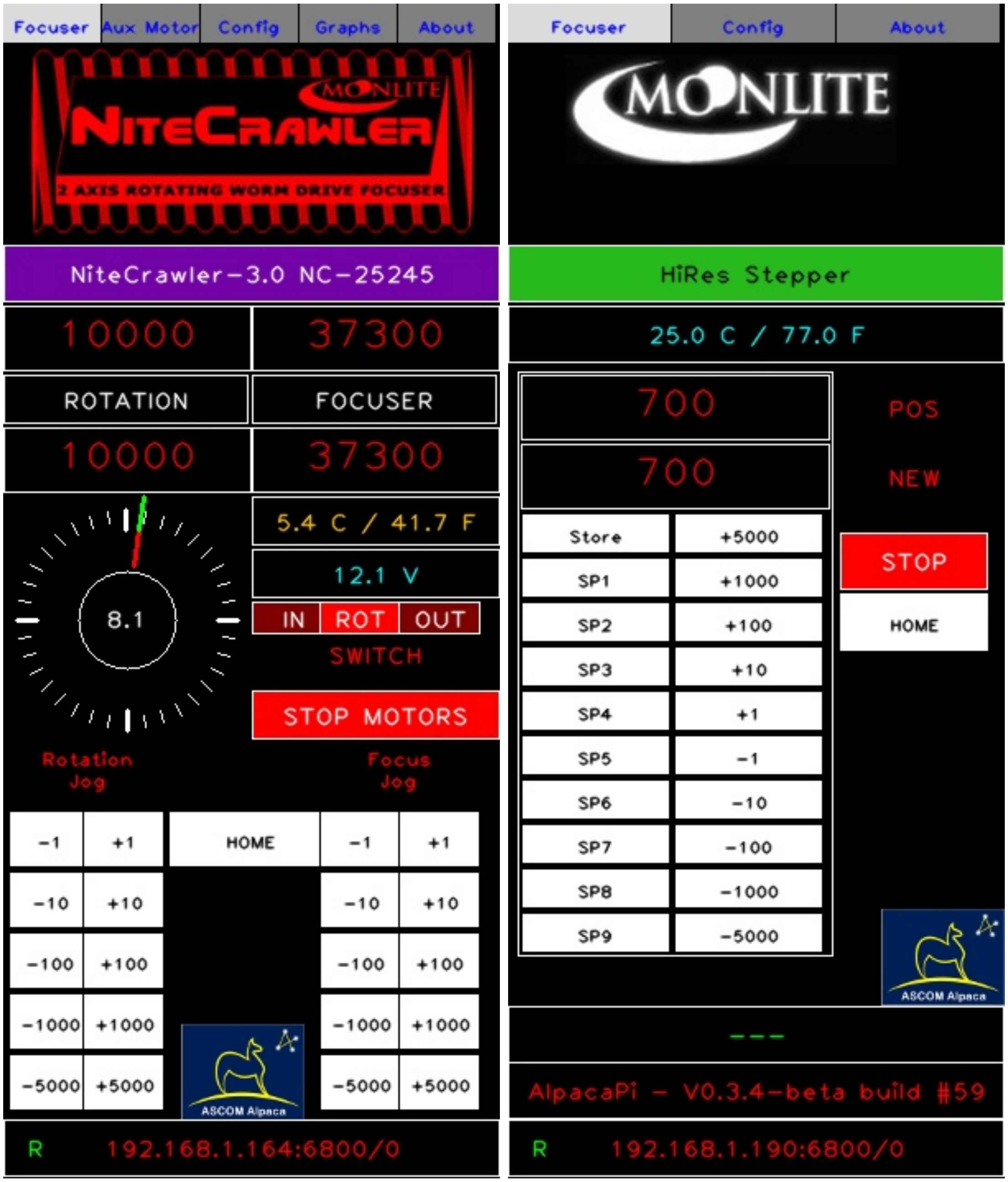
Cmd=/api/v1/camera/0/readoutmode

AlpacaPi - V0.3.7-beta build #71

R 192.168.1.161:6800/0

My focuser application is specifically designed to look exactly like the MoonLite NiteCrawler application. As in the case of the camera and the filter wheel. This app recognizes that there is both a focuser and a rotator on the same device. It will also work with the normal MoonLite focuser controller that does not have the rotator.

The image on the right is for a focuser that does not have an integrated rotator. Again, designed to look just like the MoonLite application.



Alpaca operation

The alpaca devices drivers are the "server" they sit and "listen" for a request exactly like a web server.

Each alpaca driver must listen on 2 ports, normally implemented in 2 different threads.

One thread is the "discovery" protocol which MUST listen on port 32227, this is simply a UDP connection listening for broadcast messages. When one request received, it reports back a very simple JSON packet that looks like {"alpacaport": 6800}

The number, 6800 in this case, is the TCP/IP port that the other thread is listening on.

Technically the discovery thread is not really required. I know of some implementations that do not use it. However it puts the responsibility on the client program to be told the address and port manually.

The other thread listens on its own port (in this case 6800) for HTTP commands

On your client machine, you can then go to that IP address with a web browser as follows http://ipaddress:6800/setup
(refer to https://ascom-standards.org/api/?urls.primaryName=ASCOM%20Alpaca%20Management%20API)

It also responds to Alpaca commands which look like this http://192.168.1.89:6800/api/v1/telescope/0/interfaceversion

- **192.168.1.89** is the IP address of the alpaca server (device)
- **6800** is the port
- **api** is the string telling the server it is an api command
- **v1** is the Alpaca version
- **telescope** is the device type
- **0** is the device number (in case the driver is controlling more than one)
- **interfaceversion** is the command

Refer to https://ascom-standa...paca Device API The Alpaca driver never initiates any communication. It is the "server"

Normal operation:

1. Run a client program such as a camera controller.
2. This program, the client, would send out the discovery query via a broadcast to port 32227, it would then listen for responses. For each response, it would use the management interface to query what kind of device it is. In this example, we are looking for cameras.
3. For each camera we find, open up a controller window to control THAT camera. If no cameras are found, report the fact and exit.
4. Refer to the application section above for an example.
5. The client then, on its own, periodically sends commands to the camera driver asking for information or sending a command. For example, get camera type, specs etc and display to the user. The user then clicks on TAKE PICTURE. The client sends the command to the driver/server to do that operation. Note: EACH COMMAND is a HTTP get/put command, the TCP port is opened, the command is sent (using HTTP), the response is received and the connections is CLOSED. All of the responses are JSON
6. The client polls the driver asking if its done, when it is, it asks for the picture data, gets the data and does what it wants with it. (i.e. save it).

Connecting ASCOM to ALPACA.

ASCOM programs know NOTHING about Alpaca, this is where the ASCOM/Alpaca bridge comes in. It runs on your Windows machine and it uses the discovery protocol to find Alpaca devices on the network. It then allows you to create a "virtual" ASCOM device on your Windows machine that "translates" all of the ASCOM commands to Alpaca commands.

It is VERY IMPORTANT to note that there is a one-to-one translation. That is the Alpaca commands are the EXACT same commands ASCOM already uses. The ONLY difference is the method that the data is moved back and forth. In normal ASCOM, this is done with the Windows COM interface. With Alpaca, it is HTTP/JSON using what is called a RESTful interface. (I dont think the "RESTful" part is relevant, since I implement everything at the TCP/IP level).

Once this "virtual" devices has been created, your existing ASCOM programs will be able to talk to ALPACA devices.

NOTE: I do not use Windows, and except for the use of the ASCOM test program called CONFORM, I have never used ASCOM in any way nor have ever used an ASCOM program to control anything in my observatory.