Image Acquisition on PixInsight
Do we really need another tool for controlling cameras?

David Raphael

INTRODUCTION

I think it depends on whether or not you are content with what you use today. Personally, I like a number of the tools that are available commercially. In fact, if it wasn't for Emanuele (a PixInsighter) – I may have never thought of working on this. But not long after it was suggested on the PixInsight forums – I realized that PixInsight would make a fantastic image acquisition platform!

PixInsight allows me to extract every ounce of data possible without destroying the data in the process. I realized that this is exactly what I wanted for image acquisition as well! I believe that it accomplishes this with several key features:

- Object Oriented Design
- Component Based Architecture
- Scriptability
- Programmability
- Community

These features allow PixInsight to be very fluid. It moves with the desires and philosophy of the users. This may seem like a peculiar way to describe a piece of software, but it reminds me very much of the text editor EMACS[1].

Just like EMACS became more powerful as users contributed functionality – PixInsight becomes more powerful as users contribute! This is a very organic growth model.

Visual Studio is to MaximDL what EMACS is to PixInsight – at least in my crazy way of looking at the astronomical software landscape. Of course most C++ developers use Visual Studio – but the really good hackers use something like EMACS, vi or even something else...they want to get closer to the code...just like we want to get closer to the pixels! It is like the difference between using the command line and a UI-the UI is definitely easy to get started with, but the command line offers maximum flexibility for power users.

Along with the awesome extensibility that is afforded by this approach – PixInsight runs on every Operating System I would ever run. In fact, it even runs on FreeBSD! I don't particularly like being tied to one platform for astronomical work.

Finally, I am tired of having so many applications running at the same time while doing my imaging. These applications have various quirks requiring you to do strange things like turn UAC[2] off! I really would like to have one system doing all of the work that was designed to do everything in harmony-not

---

1*For those of you unaware of what EMACS is – it is basically one of two standard text editors that hackers / programmers / sysadmins have been using for over 30 years – vi being the other.*

2 User Account Control – A Microsoft security infrastructure that prevents applications from executing as Administrator without user approval.

a bunch of glued together pieces that seem to break every time you upgrade one piece.

So now that I've ranted – let's look at how I am implementing ImageAcquisition on top of PixInsight!

IMPLEMENTATION

First, I've divided up the effort into several building blocks:

- Image Acquisition Settings
- Expose Image
- Imaging Session
- Auto-guide
- Focus Control
- HFD/FWHM
- Auto-Focus
- Plate-Solve
- Slew
- Dome Control

This is not a comprehensive list – I simply made a list of all the items that I felt belonged in the image acquisition bucket.

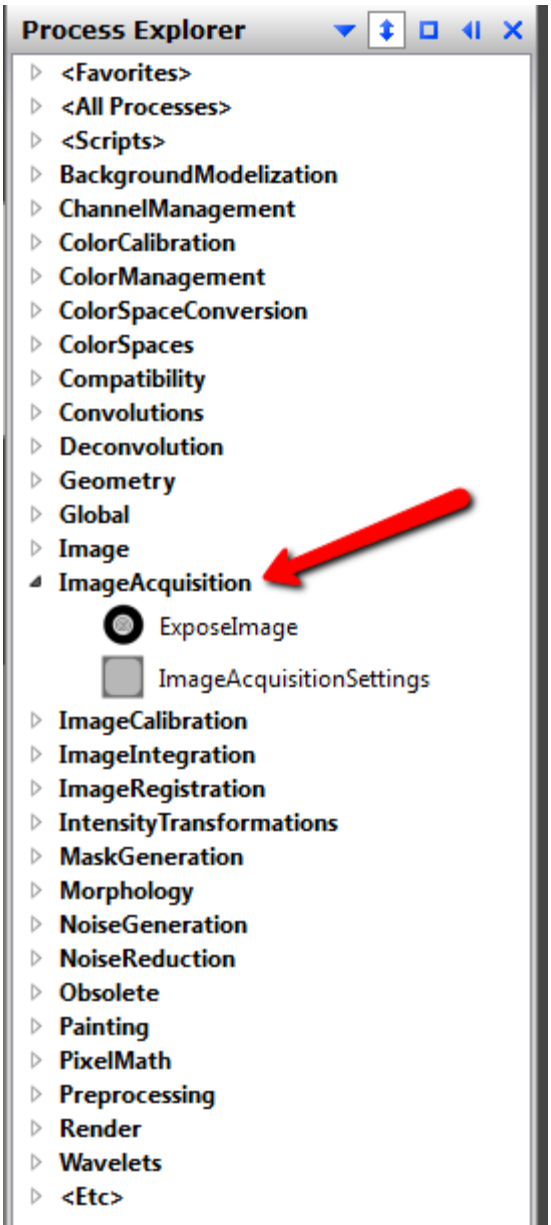
In addition to these items, I also had to design a device driver interface. I considered a number of options, but ultimately I decided that it would be best to have a PixInsight device layer:

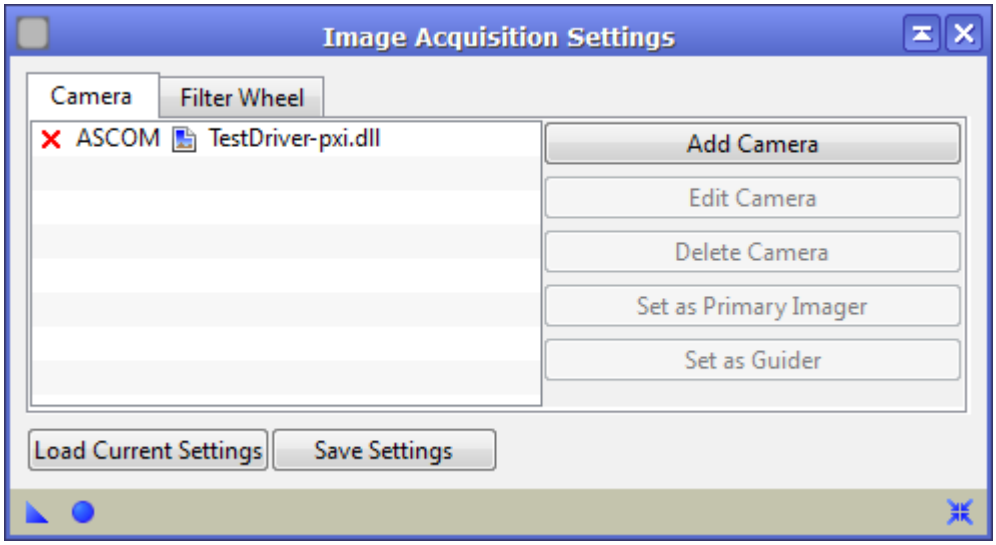
- IPixInsightDevice
- IPixInsightCamera
- IPixInsightFilterWheel
- *IPixInsightWhatever...*

I designed this layer as a simple C++ interface that developers can adapt their own device drivers accordingly. I have implemented ASCOM support already, and I plan on adding support for other devices myself.

So far, I've only implemented a basic camera control through the ExposeImage process and persistent settings through the ImageAcquisitionSettings process.

You can see the ImageAcquisitionSettings process below:

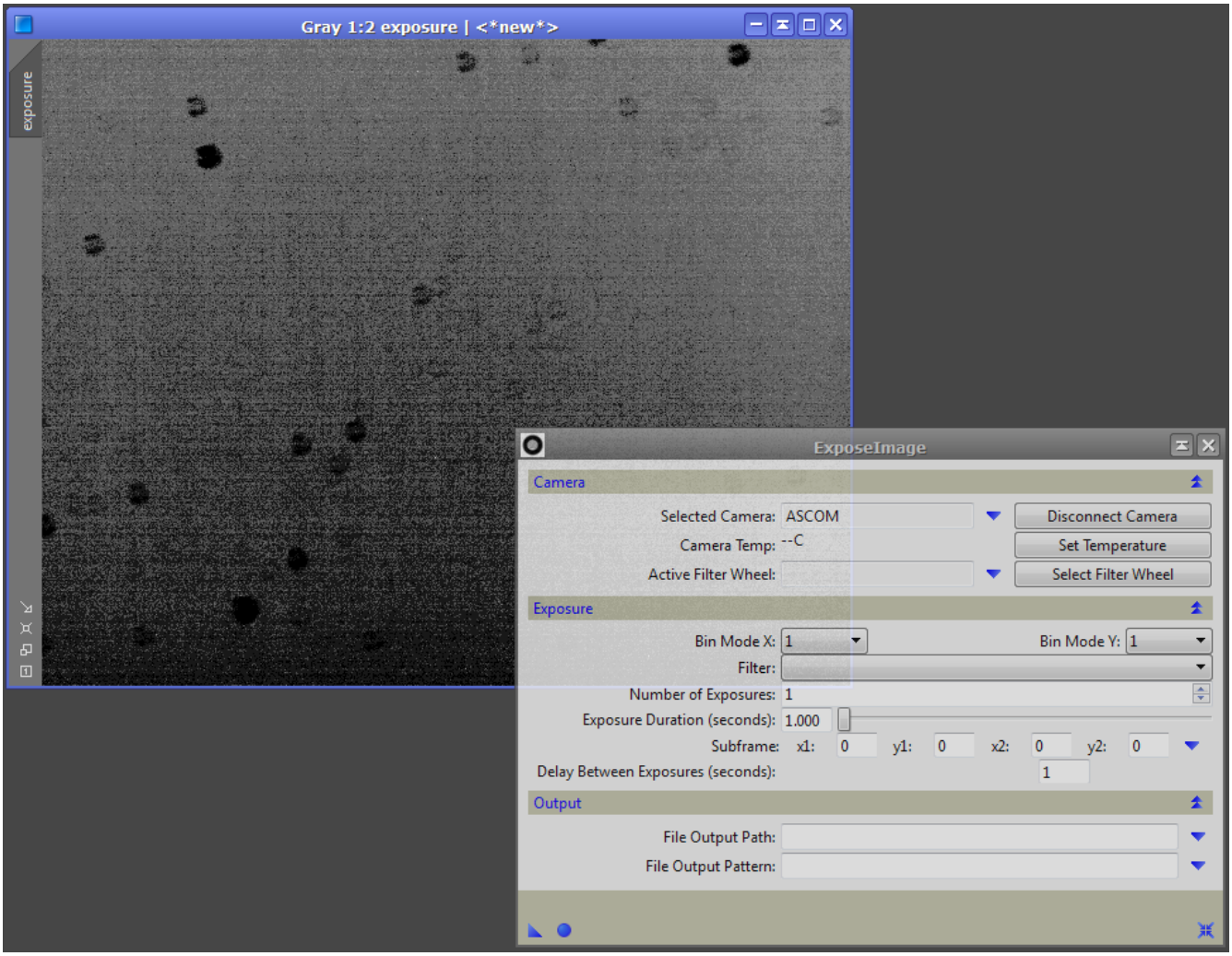




This allows you to configure as many different cameras and guiders as you want. You can save these settings and reload them when you open and close PixInsight. Additionally, you can save your settings through the standard PixInsight mechanism via a process icon.

Below is the ExposeImage process with a sample image that was taken. Look at all of that dark nebula![3]

3 This is not a deep sky image. It is just an image from an inexpensive camera sitting on my desk, pointed at the wall.



If this were just another application with a basic UI for controlling a camera – I would say that this is not very impressive. However, what really gets me excited – is the fact that you can leverage the PixInsight platform's genius right away! For example – this screenshot demonstrates the power of an object oriented platform:

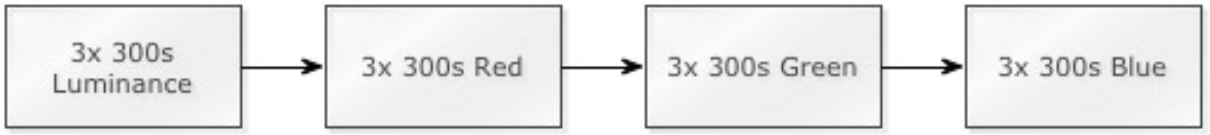

ProcessContainer
Proc Id Mask Start UTC | Time
0 <Root>
1 ExposeImage
2 ExposeImage
3 ExposeImage
var p = new ProcessContainer;
with (p)
{
var p_0 = new ExposeImage;
with (p_0)
{
exposureDuration = 1;
exposureCount = 1;
cameraName = "cam_name";
filterWheelName = "";
setTemperature = -1.5e+001;
filter = "NONE";
binModeX = 1;
binModeY = 1;
subFrameX1 = 0;
subFrameY1 = 0;
subFrameX2 = 0;
subFrameY2 = 0;
delayBetweenExposures = 1.0e+000;
fileOutputPath = "";
fileOutputPattern = "";
}
add(p_0);
var p_1 = new ExposeImage;
with (p_1)
{
exposureDuration = 2;
exposureCount = 1;
cameraName = "cam_name";
filterWheelName = "";
setTemperature = -1.5e+001;
filter = "NONE";
binModeX = 1;
binModeY = 1;
subFrameX1 = 0;
subFrameY1 = 0;
subFrameX2 = 0;
subFrameY2 = 0;
delayBetweenExposures = 1.0e+000;
fileOutputPath = "";
fileOutputPattern = "";
}
add(p_1);
var p_2 = new ExposeImage;
with (p_2)
{
exposureDuration = 3;
exposureCount = 1;
cameraName = "cam_name";
filterWheelName = "";
setTemperature = -1.5e+001;
filter = "NONE";
binModeX = 1;
binModeY = 1;
subFrameX1 = 0;
subFrameY1 = 0;
subFrameX2 = 0;
subFrameY2 = 0;
delayBetweenExposures = 1.0e+000;
fileOutputPath = "";
fileOutputPattern = "";
}
add(p_2);
}
JavaScript
Copy Code
Edit Info

For those of you unfamiliar with the power of PixInsight's processing container – you need to check it out. You can drag the little triangle of any process over to a process container and it will allow you to chain together multiple processes that can be executed with a single click. Here I have created a series of exposures with different lengths.

While this example is not very exciting, it already allows you to acquire a series of images with different length and different filters! Not too bad...

BIG PICTURE
So how does all this "extensibility" and "object oriented" stuff help me?

Astrophotographers are some of the most brilliant people I've ever seen in a community. They are also some of the most opinionated! This leads to a lot of divisive information as to what is the best approach to everything that we do. Let's look at a simple example of the following image sequence:

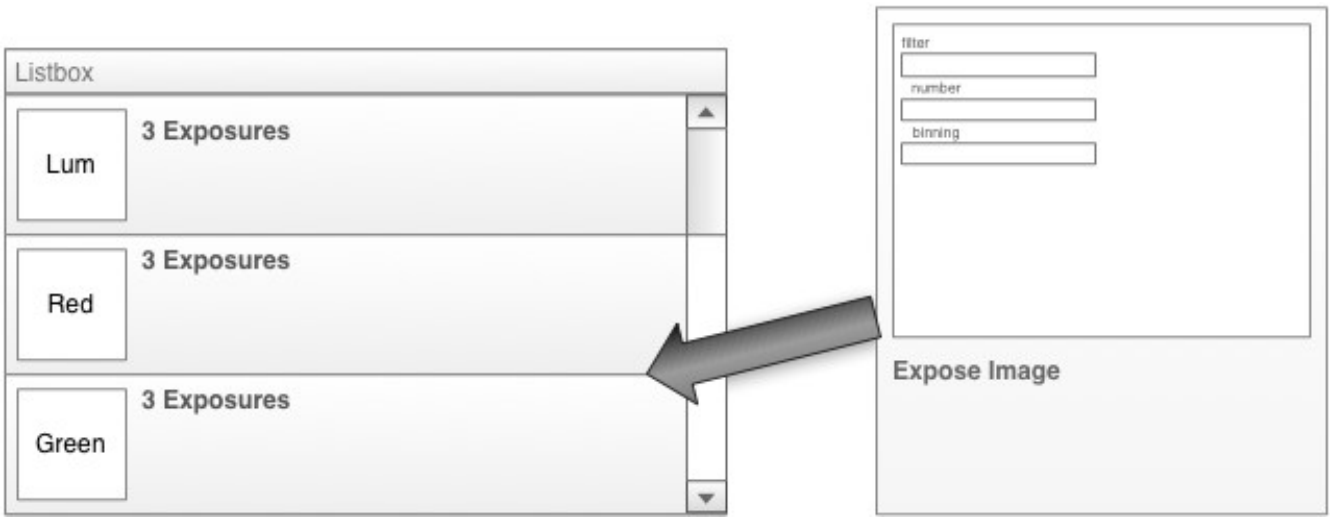


We are planning to take a basic sequence of 3x5 minute sub exposures. So with a traditional, non-object-oriented approach – we would see an interface that looks something like this:

Filter	Number of Exposures	Duration	Bin
☐Luminance	3	300s	1x1
☐Red	3	300s	1x1
☐Green	3	300s	1x1
☐Blue	3	300s	1x1

I've simplified this for the sake of discussion. A lot of applications implement a similar table.

For the basic use case of taking these images in sequence, this works pretty well. Let's contrast this with an object oriented interface:



Here, a user defines the behaviors of individual exposure sets and “drags and drops” adds them to a list of objects to execute.

This is actually slightly less efficient than just tabbing through a spreadsheet like interface and entering the appropriate values. So why would we want this over a more efficient approach?

Actually, it is only more efficient in the simplest of use cases. Let's explore a more complex use case – exposure + auto-focus. First, we need to determine how we want to auto-focus across a series of images. Remember that comment I made about astrophotographers being opinionated? Here is a great example. Ask a forum about auto-focus. Specifically, ask them how often they focus. You will get a variety of answers. The reality is that there is not one correct answer. It depends highly on the conditions of the imaging session. So let's add auto-focus to our traditional interface:

Filter	Number of Exposures	Duration	Bin
☐ Luminance	3	300s	1x1
☐ Red	3	300s	1x1
☐ Green	3	300s	1x1
☐ Blue	3	300s	1x1

☐ Auto-focus between each set

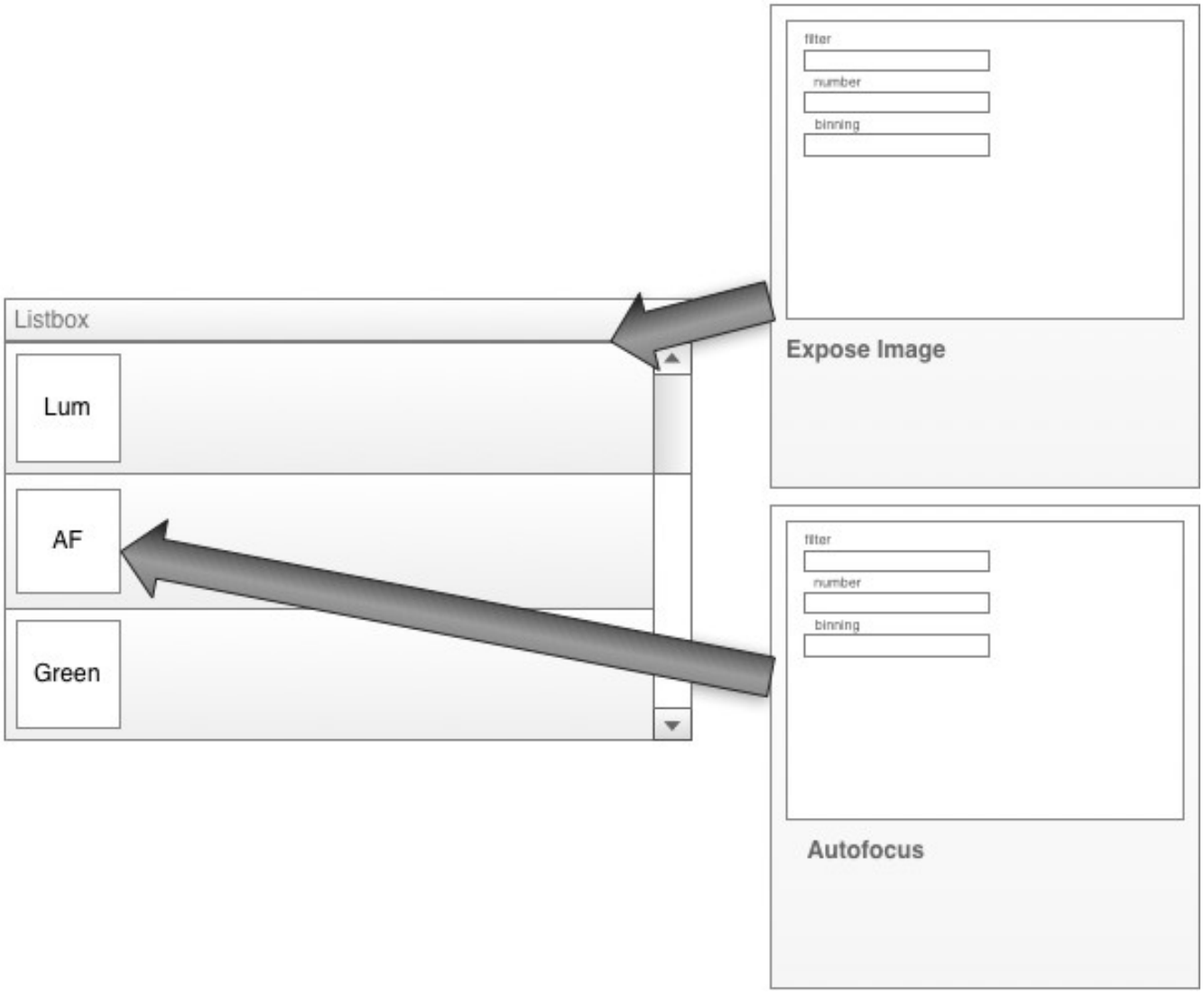
We have added a simple check box. But what if we want to auto-focus every frame? The simple answer is that you can just do 1 exposure in each filter and you will have that. But that will quickly create a lot of duplication in the list. You will have to have 12 items in the list. I know – we could just add another option that says *Auto-Focus between each frame.* But there are so many more use cases – how about auto-focus based on temperature? Frame-to-frame FWHM change? And so on...

So what we really need is the ability to compose behaviors – we need the ability to create systems that can adapt to the situation as needed.

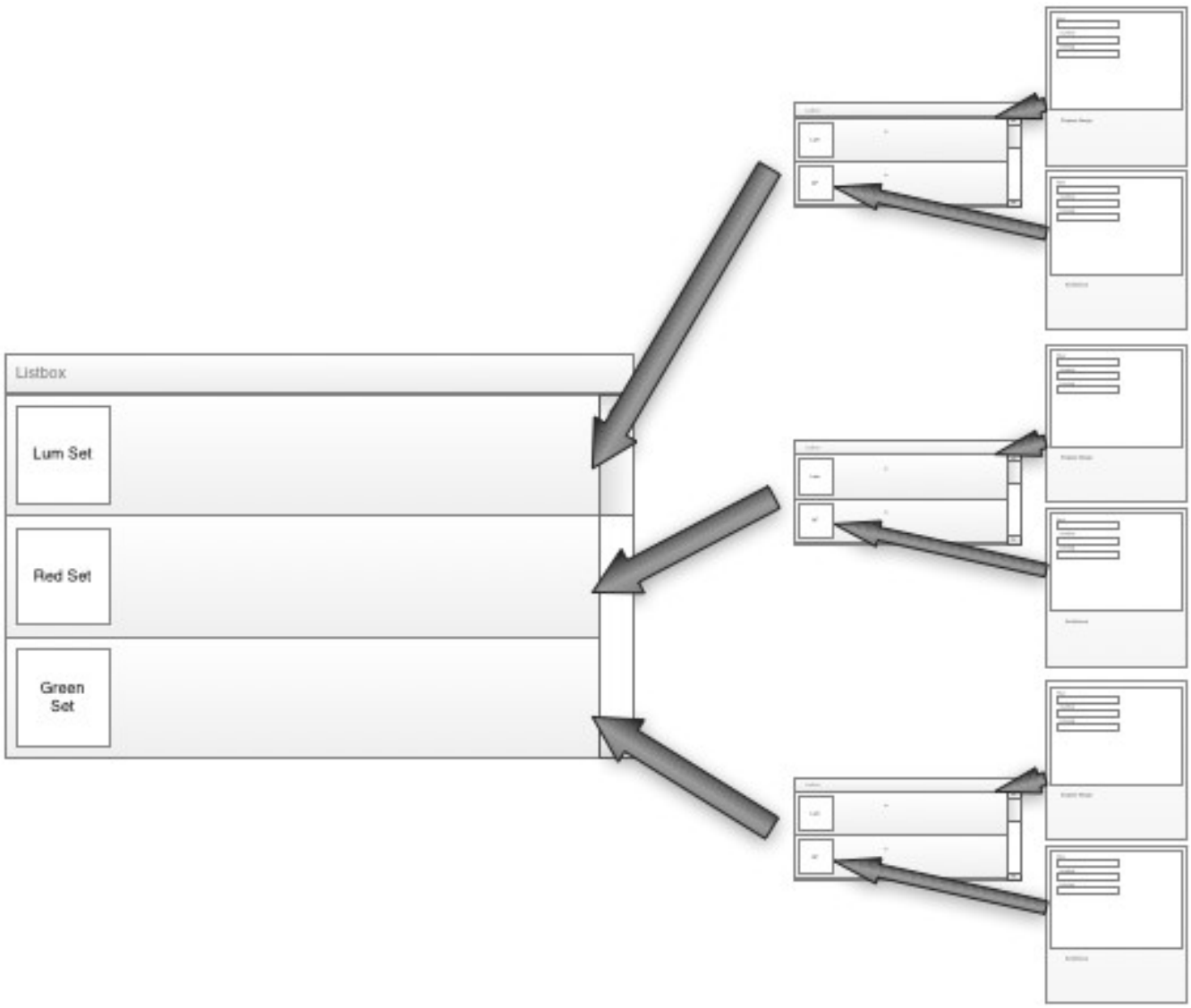
This is very difficult to do with traditional approaches. You tend to see a hybrid of lists and popup windows buried in the interface. Ultimately these approaches are mixing object oriented approaches into a traditional interface. And I believe this epitomizes non-intuitive interfaces. I have to dive deeper and deeper down a wormhole to perform certain actions with these interfaces. They become Swiss Army Knives. When features are appended to already working interfaces, users are disrupted and have to re-learn depending on the complexity of the disruption.

So what does it mean to “compose” behaviors?

Let's consider our previous example involving auto-focus. In our object oriented world, we do not add an auto-focus behavior to our expose process. Instead, we invoke auto-focus just like we invoke the exposure process. It might look something like this:



But here is where I think it gets really interesting: When we want to create more complex behaviors, we can do so by composition! Let's take a look:



So now we have created a more complex behavior by "composing!" The key to making this work is ensuring that each component sticks to doing 1 thing well. Also, I believe that we will end up defining two type of objects in these systems: Action Objects and Management Objects. We've talked about Action Objects already – so what is a Management Object? In our example we show a "Listbox." Really this is just a "Process Container" in PixInsight. But there are more complex management objects we would need to do something like conditional execution of an action. One such object might conditionally execute a "Listbox" based on the result of an action. Another object may perform a more supervisory role such as closing a dome and terminating an imaging session based on a weather probe.

I've only scratched the surface of the power offered by composition and an object oriented approach.

PUTTING IT TO THE TEST

Recently at NEAIC, Vicent Peris, the editor of this magazine, asked me to add a feature to my list – auto-flats. I told him that this would be something easily scripted! There is no need to hard code this into the system-at least not just yet.

Here is a simple script that exposes images until a target mean pixel value is reached. This is not a very thorough example – I spent about 5 minutes writing this -

```
var theDuration = .1;
var running = true;
while( running && theDuration < 2 )
{
   var p_0 = new ExposeImage;
   with ( p_0 )
   {
      exposureDuration = theDuration;
      exposureCount = 1;
      binModeX = 1;
      binModeY = 1;
      subFrameX1 = 0;
      subFrameY1 = 0;
      subFrameX2 = 0;
      subFrameY2 = 0;
      fileOutputPath = "c:\\tmp";
      fileOutputPattern = "test";
   }

   p_0.executeGlobal();

   var window = ImageWindow.activeWindow;
   console.show();
   console.writeln( "<end><cbr><br><b>" + window.currentView.fullId + "</b>" );
   console.writeln( "exposure..." + theDuration );
   console.writeln( "Calculating mean pixel value..." );
   var img = window.currentView.image;
   var statistics = new ImageStatistics();
   with(statistics)
   {
     meanEnabled = true;
   }
   statistics.generate(img);
   console.writeln( "mean pixel value:" + statistics.mean );
   console.flush();
   if( statistics.mean > .0006 )
   {
      console.writeln( "target exposure value:" + theDuration );
      break;
   }
   theDuration += .1;
}
```

I don't know about you...but this gets me really excited!
This is just one example. With this same approach, we can build things like Auto-focus, Plate-Solving, etc...the options are only limited by the community's creativity!

PHILOSOPHY
When I started working on this project, I struggled to decide how I wanted to license it. I have decided

to open source the project. I don't plan on making any money from the software directly. I believe that the community has not created enough open source initiatives. It reminds me of mainstream software 20 or 30 years ago – we are too small of a community not to be open. I believe that Juan has done the right thing with PCL by completely open sourcing it. This gives us a cross platform library to standardize against. Additionally, since it is based around good standards based C++ code, it is easy to create code that can be used across many projects.

CONTRIBUTING

There is still a lot of work to do in order to get ImageAcquisition ready for the field. I have open sourced all of the code – it is available here:

https://github.com/ceterumnet/ImageAcquisition

I hope that you all take a look at this module once we release it to the public. If you are feeling adventurous and want to contribute to the project – feel free to PM me on the PixInsight forums!