SI-401239: MMPAD Programmer's Guide -				
Date	Action	By	Revision	Reviewed by
4/27/2023	Create	YF	A	

Introduction

This is a preliminary document for interfacing to the Sydor MM-PAD.

See also: MMPAD User Manual
Python Utilities and Examples (on Egnyte)
Sample Images (on Egnyte)

Egnyte link: https://sydortechnologies.egnyte.com/fl/Nj2HlNNSjW

Data Format

When the MM-PAD starts an image acquistion (a "run"), it will stream data to disk, typically saving 1000 frames per file, and then auto incrementing the digits at the end of the file name by +1000.

File structure of saved data, on the server:

/mnt/raid/mmpad/set-<setName>/run-<runName>/frames/

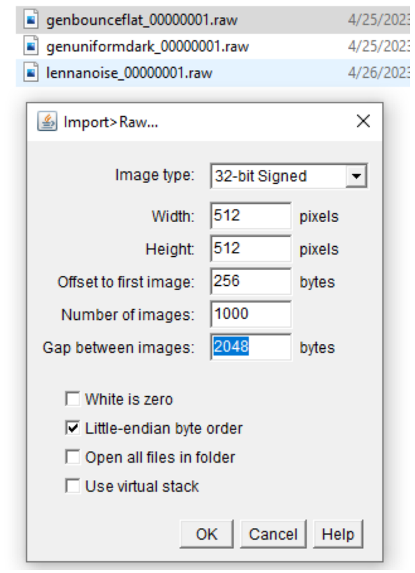
<runName>_00000001.raw
<runName>_00001001.raw
<runName>_00002001.raw … etc…

An example run with set name "MMmodules08FEB2023", and run name "MM10MM20",
with number of frames to capture set to 1000, results in two directories on the server Raid driver at:

/mnt/raid/mmpad/set-MMmodules08FEB2023/run-MM10MM20/frames, and
/mnt/raid/mmpad/set-MMmodules08FEB2023/run-MM10MM20/metadata

Inside the frames folder is one file:
MM10MM20_00000001.raw that is 1,026,000 kB in size.

The raw file format contains 256 byte header, 512x512x4 bytes of image data, followed by a 1792 byte footer per image. Image files may be concatenated to create longer sequences, but note that the main rtsup server is currently limited to files of 1000 images each. To read images into imageJ, use the following Import Raw Dialog:



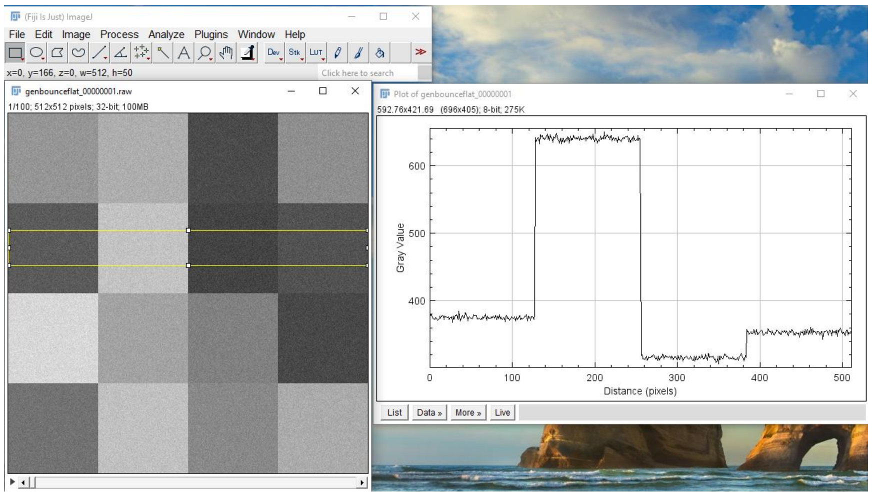
genbounceflat_00000001.raw	4/25/2023
genuniformdark_00000001.raw	4/25/2023
lennanoise_00000001.raw	4/26/2023

Import>Raw...

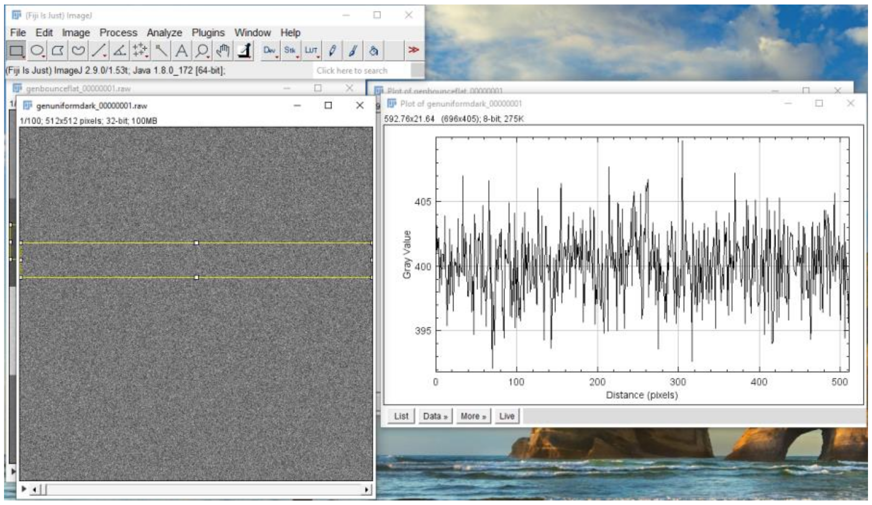
Image type: 32-bit Signed
Width: 512 pixels
Height: 512 pixels
Offset to first image: 256 bytes
Number of images: 1000
Gap between images: 2048 bytes

[ ] White is zero
[✓] Little-endian byte order
[ ] Open all files in folder
[ ] Use virtual stack

3 Example Images

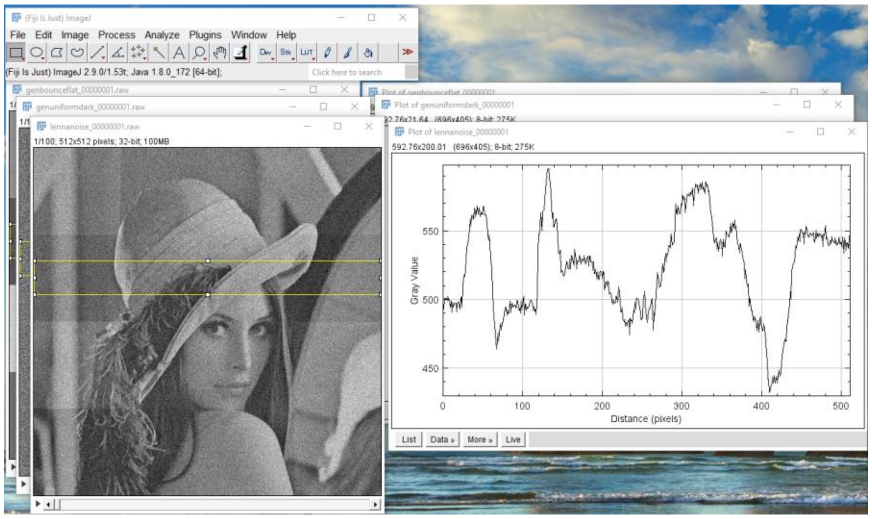

(Fiji Is Just) ImageJ
File Edit Image Process Analyze Plugins Window Help
x=0, y=166, z=0, w=512, h=50
Click here to search
genbounceflat_00000001.raw
1/100; 512x512 pixels; 32-bit; 100MB
Plot of genbounceflat_00000001
592.76x421.69 (696x405); 8-bit; 275K
Gray Value
400
500
600
0
100
200
300
400
500
Distance (pixels)
List
Data »
More »
Live

file: genbounce_flat_00000001.raw



(Fiji Is Just) ImageJ
File Edit Image Process Analyze Plugins Window Help
(Fiji Is Just) ImageJ 2.9.0/1.53t; Java 1.8.0_172 [64-bit];
genuniformdark_00000001.raw
1/100; 512x512 pixels; 32-bit; 100MB
Plot of genuniformdark_00000001
592.76x21.64 (696x405); 8-bit; 275K
Gray Value
Distance (pixels)
List Data » More » Live

file: genuniformdark_00000001.raw



(Fiji Is Just) ImageJ
File Edit Image Process Analyze Plugins Window Help
(Fiji Is Just) ImageJ 2.9.0/1.53t; Java 1.8.0_172 [64-bit];
lennanoise_00000001.raw
1/100; 512x512 pixels; 32-bit; 100MB
Plot of lennanoise_00000001
592.76x200.01 (696x405); 8-bit; 275K
Gray Value
Distance (pixels)
List Data » More » Live

file: lennanoise_00000001.raw

Each sample image is only 100 frames.

A Simple Mock Data Simulator

When the MM-PAD is acquiring data, it saves the data to a fast RAID disk. One discussed approach to getting the data in real time, is to have a separate executable program that reads the files from disk, processes them, and then deletes them from the disk.

It is proposed that a soft IOC running on a Linux machine could serve this function; reading the data files from disk, passing them onto an EPICS interface, and then deleting them, all whilst MM-PAD is streaming new data files to the disk.   For speed gains, the MM-PAD server can be directed to write to a RAM disk.

We have included a Bash script that will create files, much like the MM-PAD server would.

`dataserver.sh` can be used to stream the images to the ramdisk to provide a data source for EPICS. At present, images are only output in units of 100 frames. The image files may be concatenated if longer units are desired, but note that rtsup is presently limited to 1000 frames. This Bash script will repeatedly copy a specified source file to a destination directory with sequentially numbered filenames. The format is

`./data_server.sh <source file> <destination directory> <output base name> <number of times to copy file>`

Rtsup places its output image files into <base directory>/set-<set name>/run-<run name>/frames/<run name>_<8 digit starting frame number, 1-indexed>`, so the script was made to allow a copy to an arbitrary directory. To get 2000 frames of Lenna in a run, a sample command might be

`./data_server.sh /mnt/ramdisk/img_src/lennanoise_00000001.raw /mnt/ramdisk/set-testing/run-noisylenna/frames noisylenna 20`

This will create files `noisylenna_00000001.raw`, `noisylenna_00000101.raw`, … `noisylenna_00001901.raw` in directory `/mnt/ramdisk/set-testing/run-noisylenna/frames` from the `lennanoise_00000001.raw` image file.

Note that if image files are concatenated as described above, the starting frame numbers in the script will be inaccurate, but this may be modified easily.

The intent is that you (the end customer) can write an IOC to read files, and delete them after processing. If you start the IOC first, and then run the Bash script, you will be able to test the timing and speed of such a setup.