areaDetector: A module for EPICS area detector support

Mark Rivers

GSECARS, Advanced Photon Source

University of Chicago

areaDetector Introductory Talk Outline

- Motivation & goals for areaDetector module
- Overview of architecture
- Drivers for detectors & cameras
- Plugins for real-time processing
- Viewers and other clients
- Demo with FLIR USB-3 camera

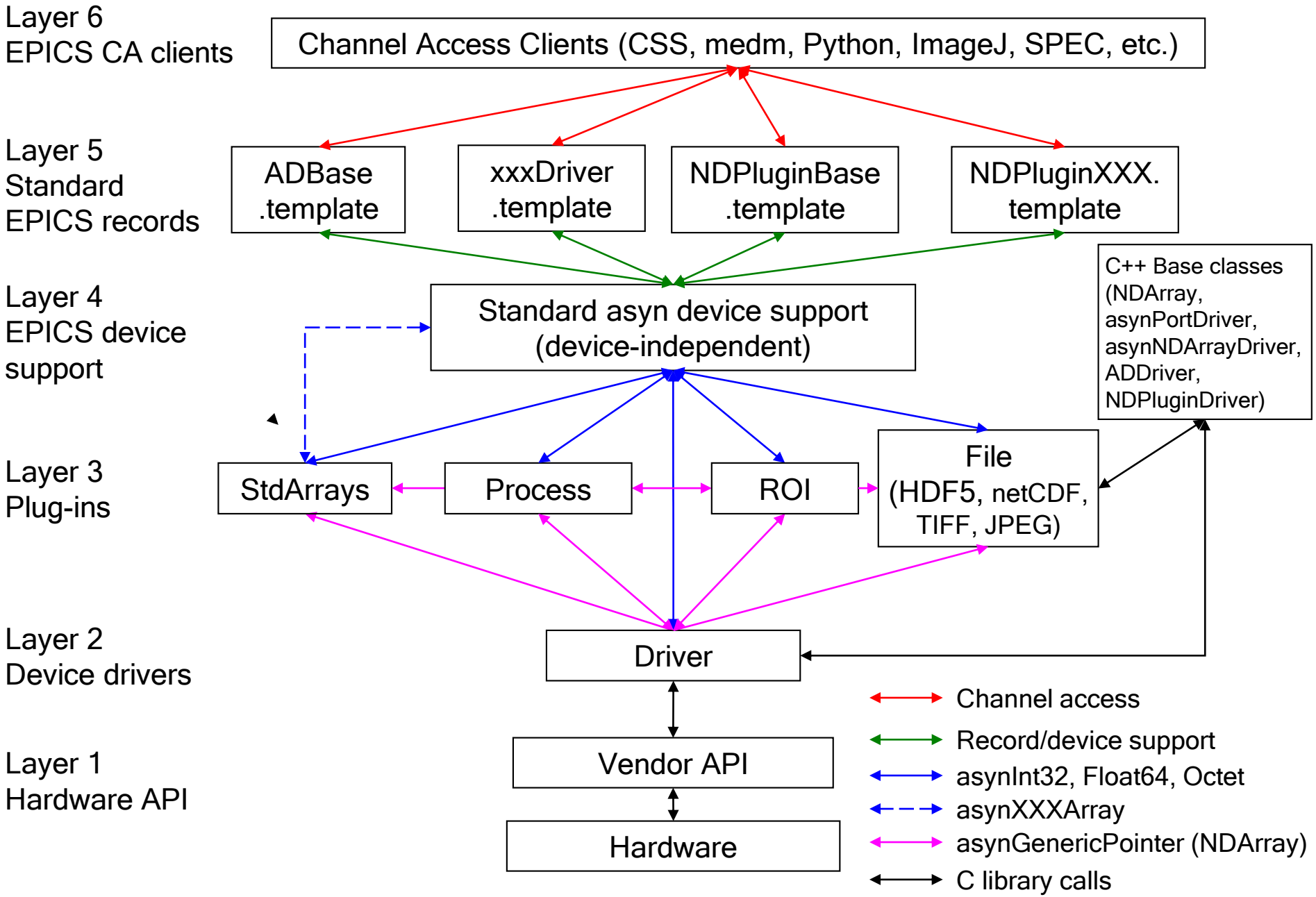
areaDetector - Goals

- Drivers for many detectors, especially those used at synchrotron beamlines
  - Handle detectors ranging from >500 frames/second to <1 frame/second
- Basic parameters for all detectors
  - E.g. exposure time, start acquisition, etc.
  - Allows generic clients to be used for many applications
- Easy to implement new detector
  - Single device-driver C++ file to write. EPICS independent.
- Easy to implement detector-specific features
  - Driver understands additional parameters beyond those in the basic set
- Middle-level plug-ins to add capability like regions-of-interest calculation, file saving, etc.
  - Device independent, work with all drivers
  - Below the EPICS database layer for highest performance

areaDetector – Data structures

- NDArray
  - N-Dimensional array.
    - Everything is done in N-dimensions (up to 10), rather than 2. This is needed even for 2-D detectors to support color.
    - Other types of devices (Xspress3 and xMAP x-ray spectrometers, quad electrometers also use NDArrays and areaDetector plugins.
  - This is what plug-ins callbacks receive from device drivers.
- NDAttribute
  - Each NDArray has a list of associated attributes (metadata) that travel with the array through the processing pileline. Attributes can come from driver parameters, any EPICS PV, or any user-written function.
    - Can store motor positions, temperature, ring current, amplifier gains, etc. with each frame.
    - Written to disk files for TIFF, netCDF, and HDF5 file formats.
- NDArrayPool
  - Allocates NDArray objects from a freelist
  - Plugins access in readonly mode, increment reference count
  - Eliminates need to copy data when sending it to callbacks.

EPICS areaDetector Architecture



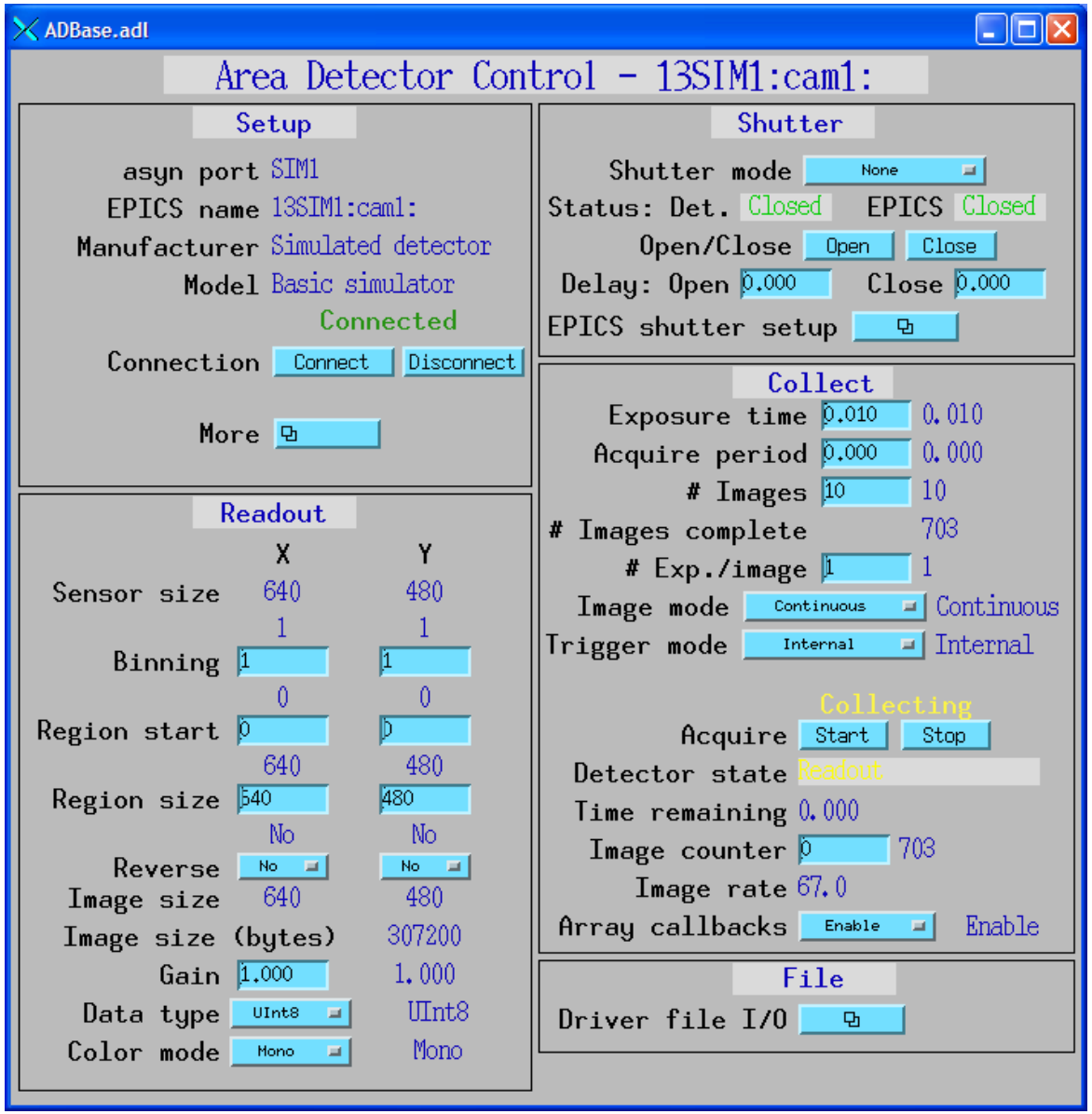
Detector Drivers

Currently ~35 detector drivers covering a wide variety of detectors.

- Simulation driver
- GigE cameras (Prosilica/AVT, Point Grey/FLIR, any GigEVision camera via aravis library)
- FLIR and AVT USB-3.x cameras
- Dectris Pilatus and Eiger pixel array detectors
- Princeton Instruments and Photometrics detectors and spectrometers
- Andor CCD and CMOS cameras
- Perkin Elmer and Dexela flat panel detectors
- Web cameras and Axis video servers
- Many more (Basler, Pixirad, Photonic Sciences, etc.)

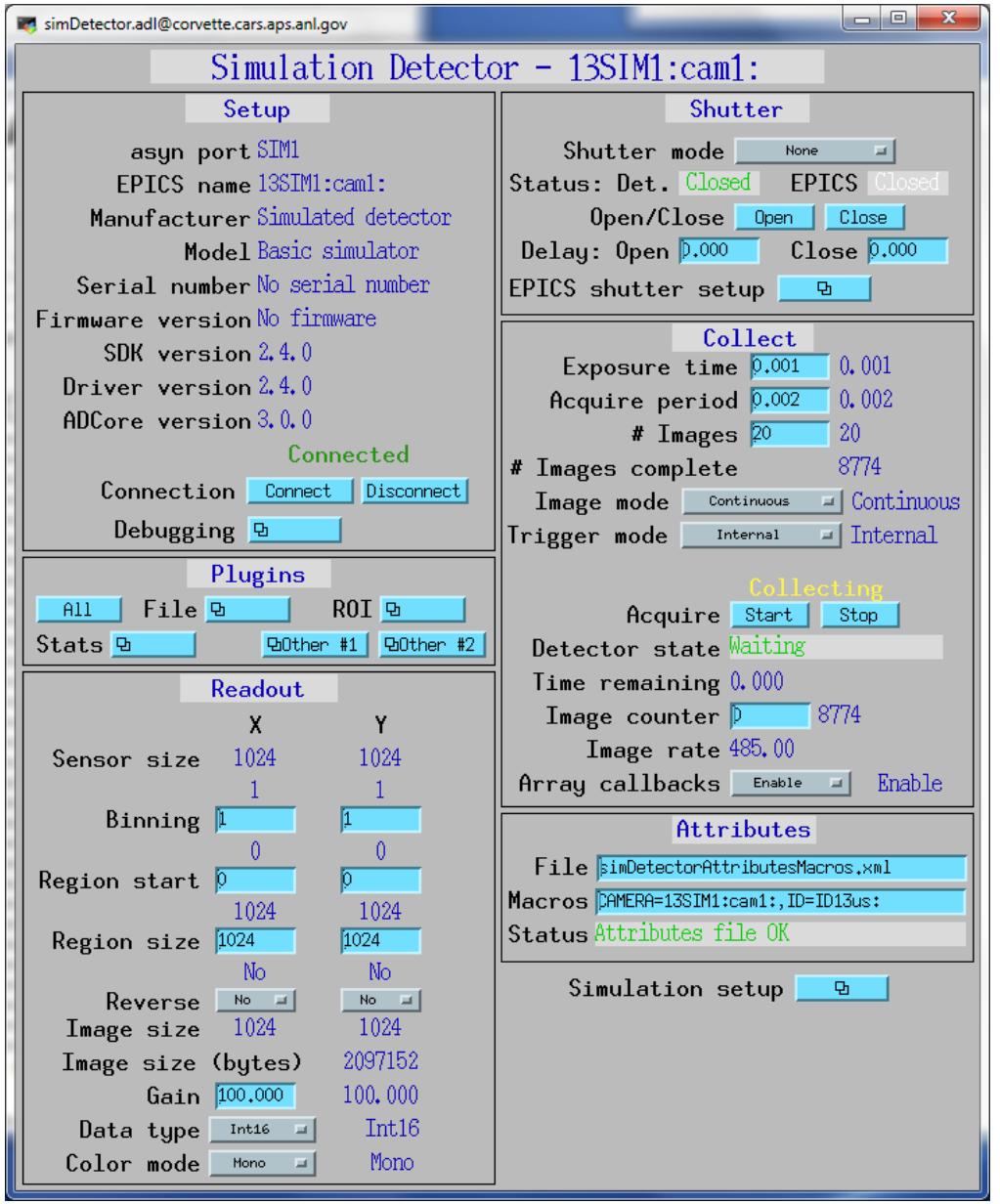
ADBase.adl – Generic control screen

- Works with any detector
- Normally write custom control for each detector type
  - Hide unimplemented features
  - Expose driver-specific features



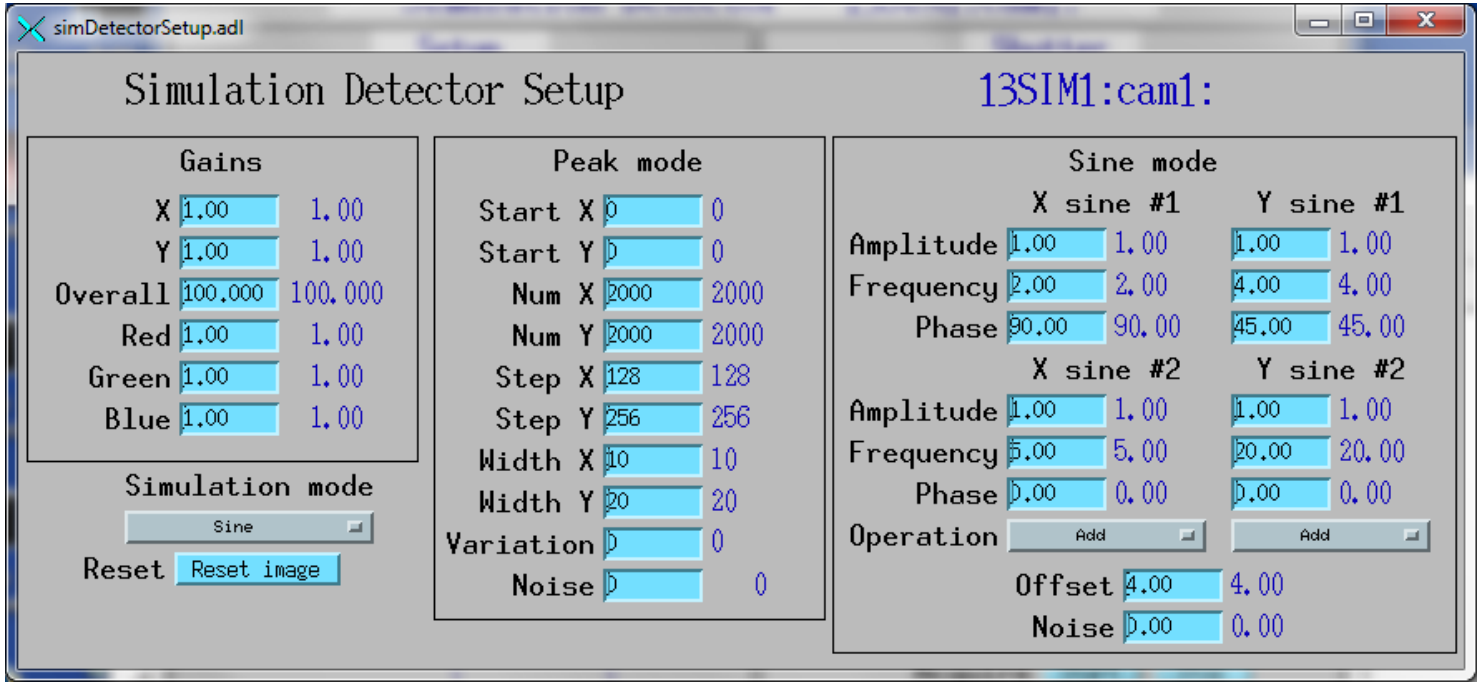
simDetector: Detector-specific screen

- 1024x1024 pixels
- 16-bit integer images
- 485 frames/s
- ~1GB/s

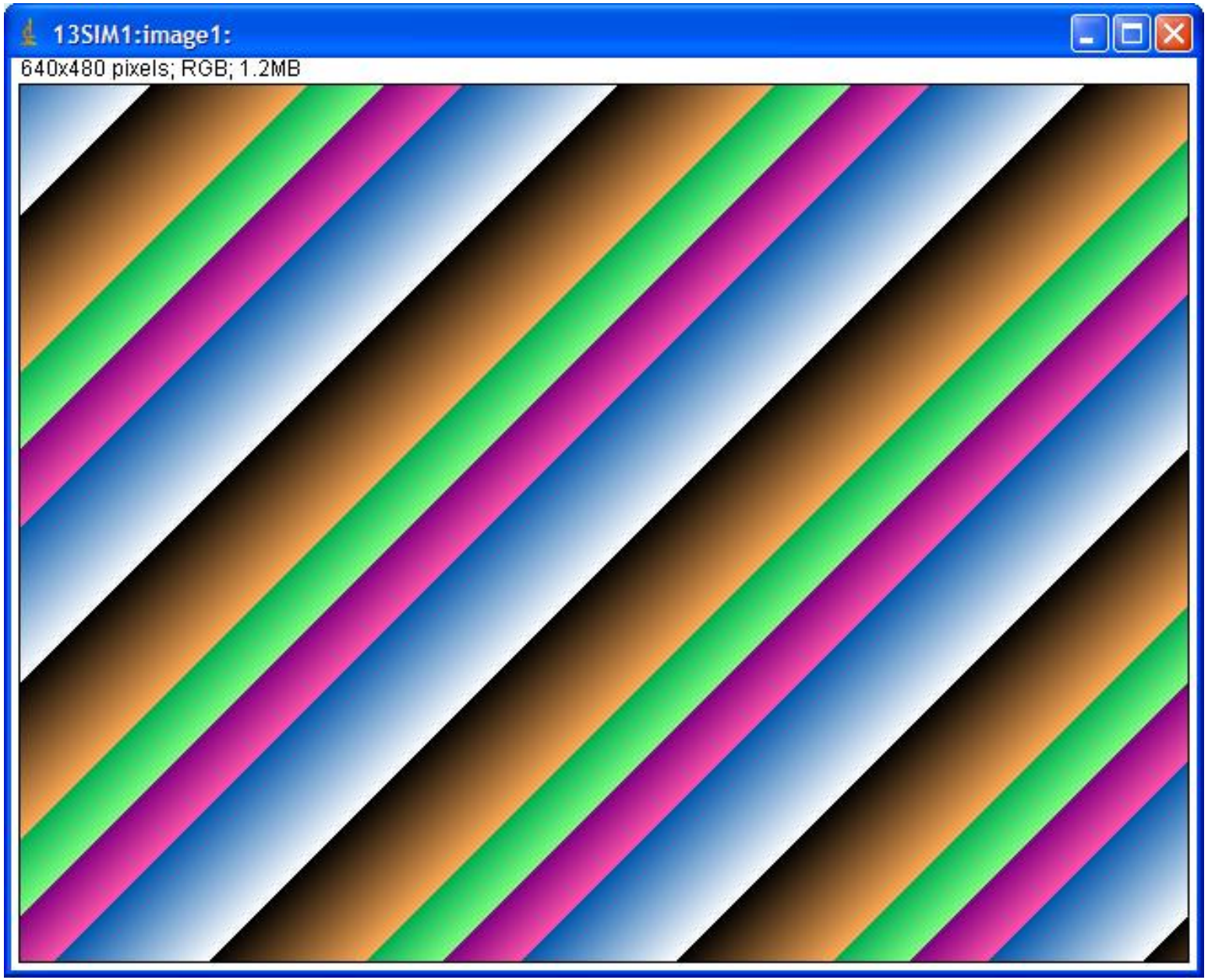


simDetector Driver

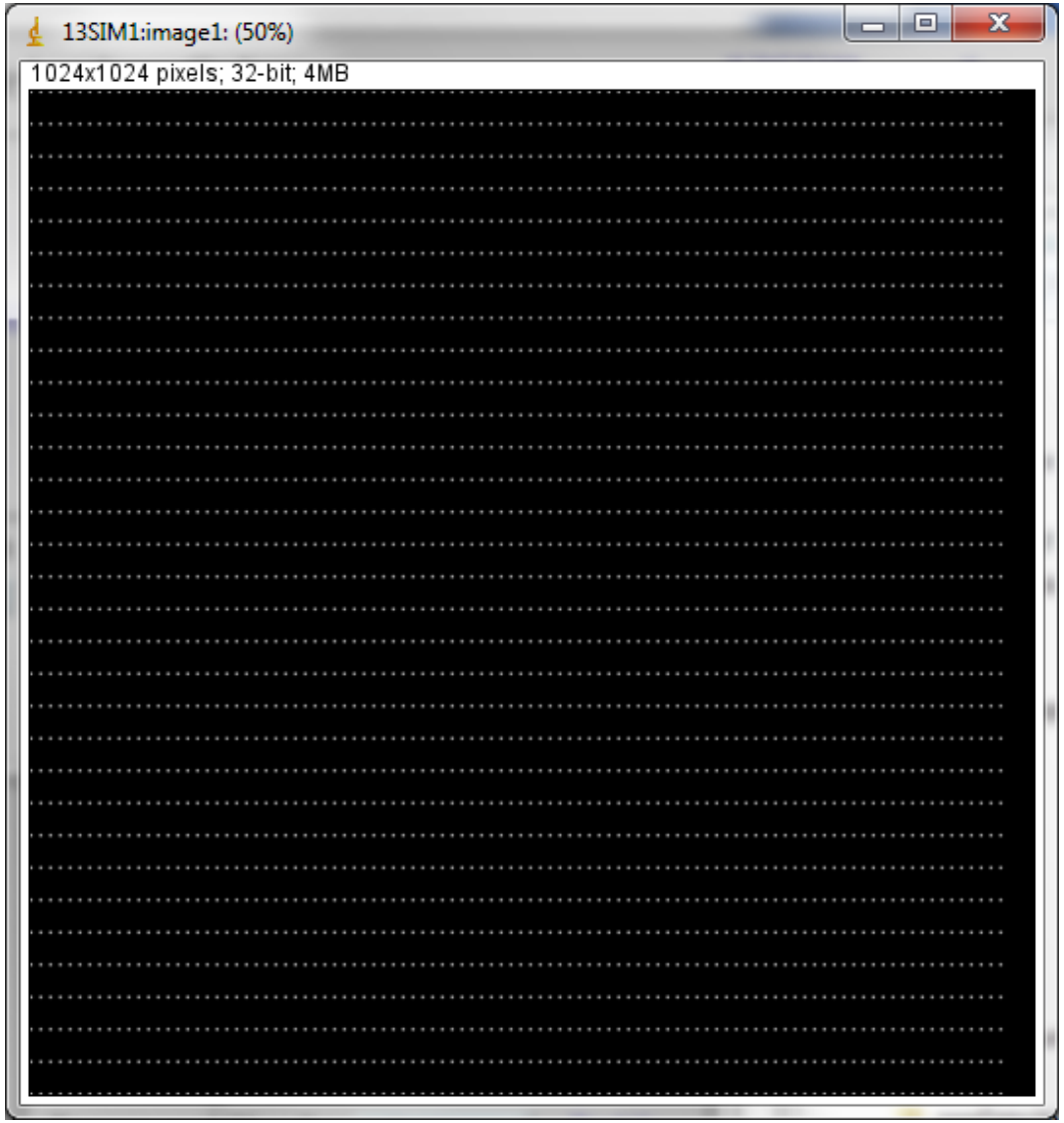
- 3 simulation modes, LinearRamp, Peaks, Sine Waves.
- Independent gains for X, Y, Overall, Red, Green, Blue
- Linear ramp has X and Y linear ramp with array index, each cycle just adds value to each pixel. Very fast.
- Peak mode is array of Gaussian peaks plus noise. Slower.
- Sine mode is 2 sine waves in each of Y and Y, summed or multiplied. Slower.

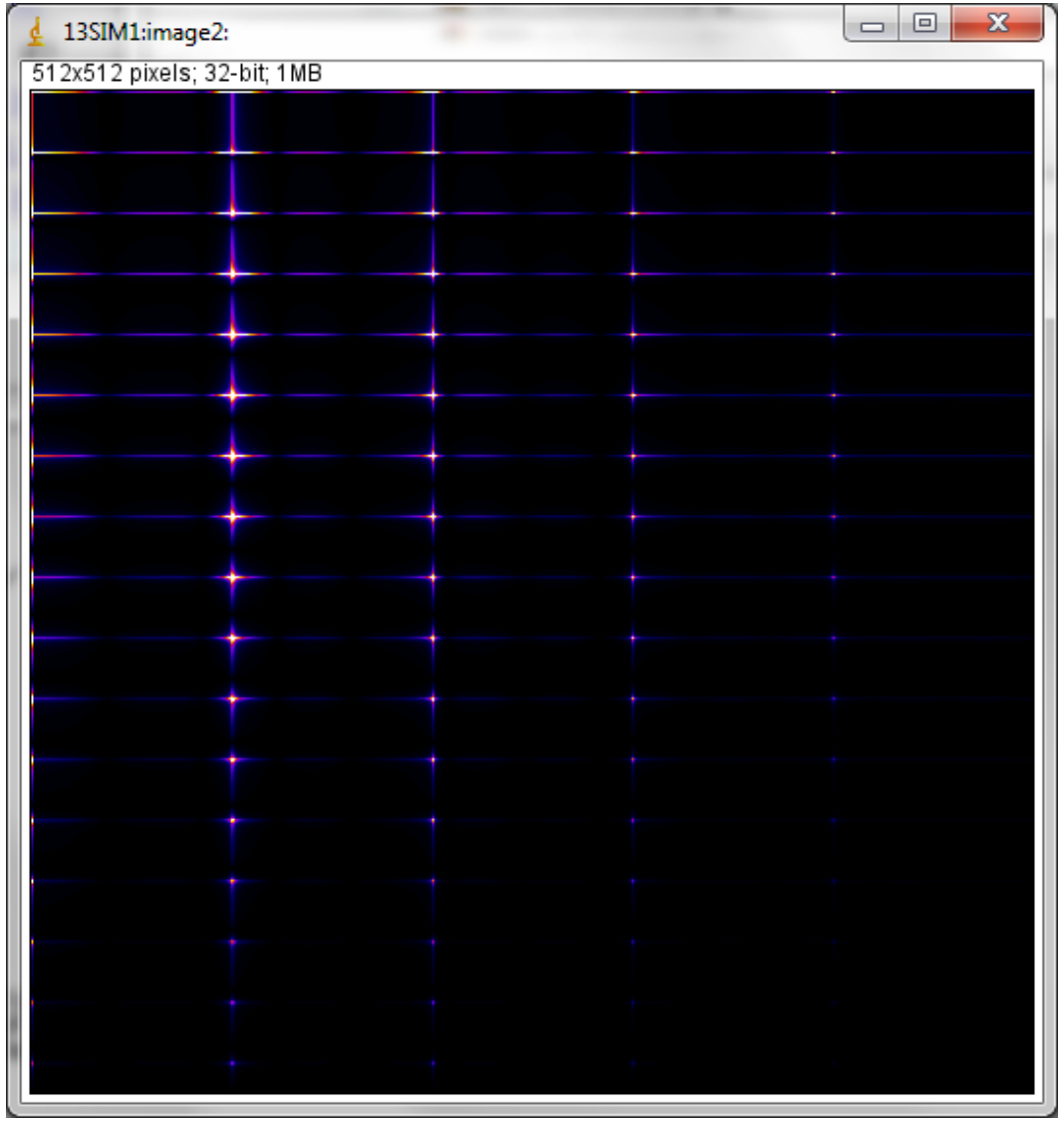


simDetector: Linear Ramp Mode

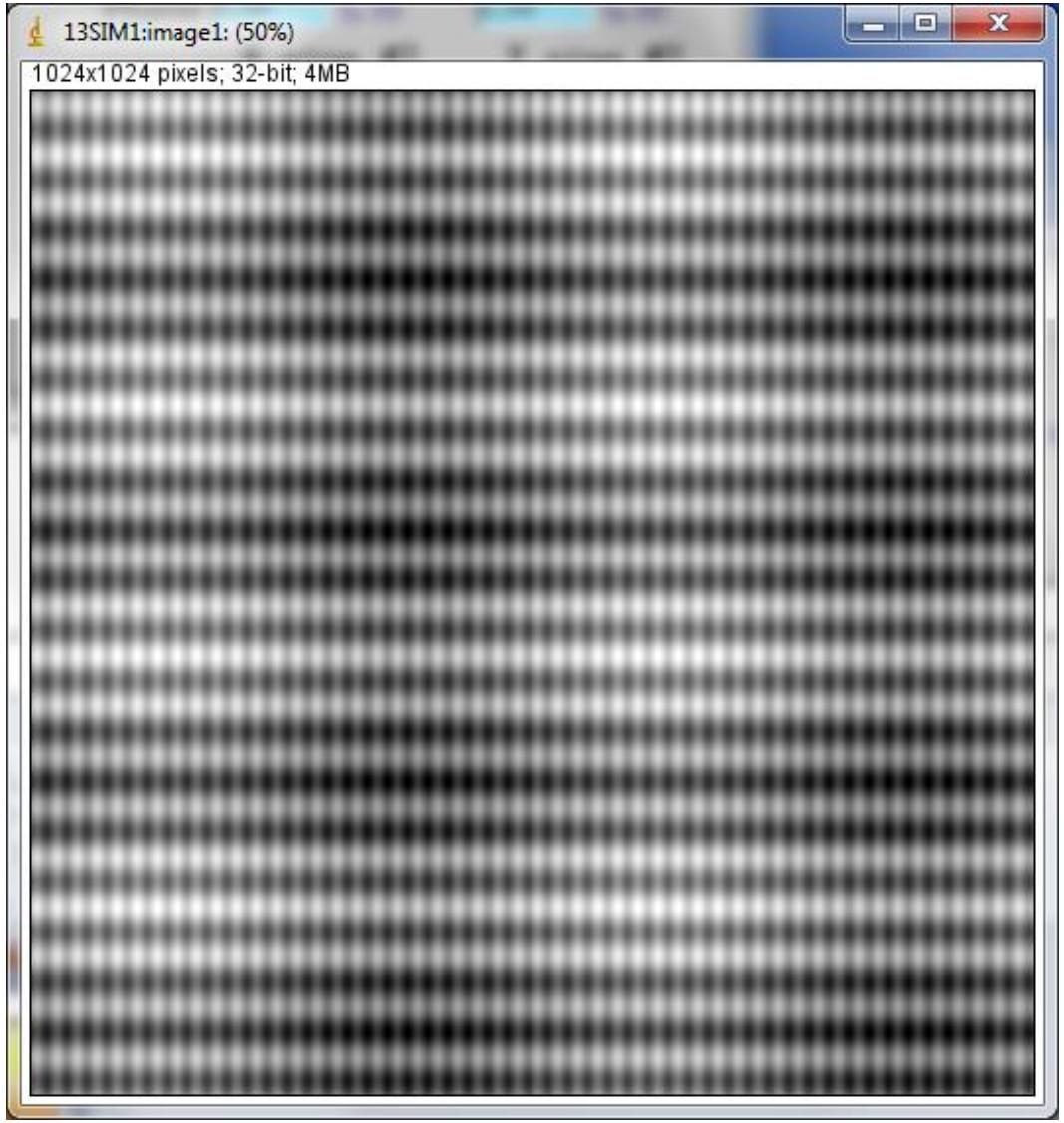


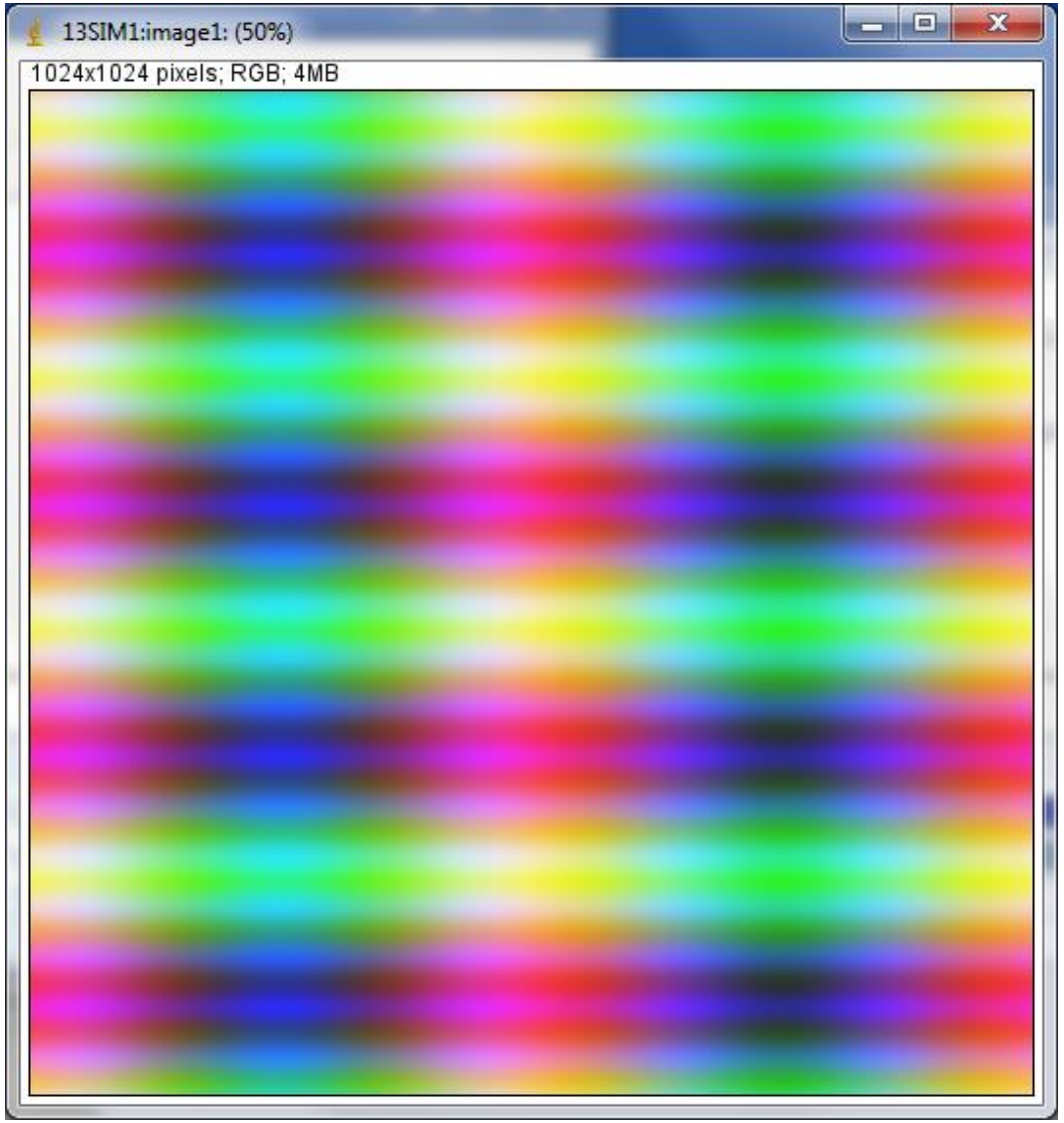
simDetector Peaks mode with FFT



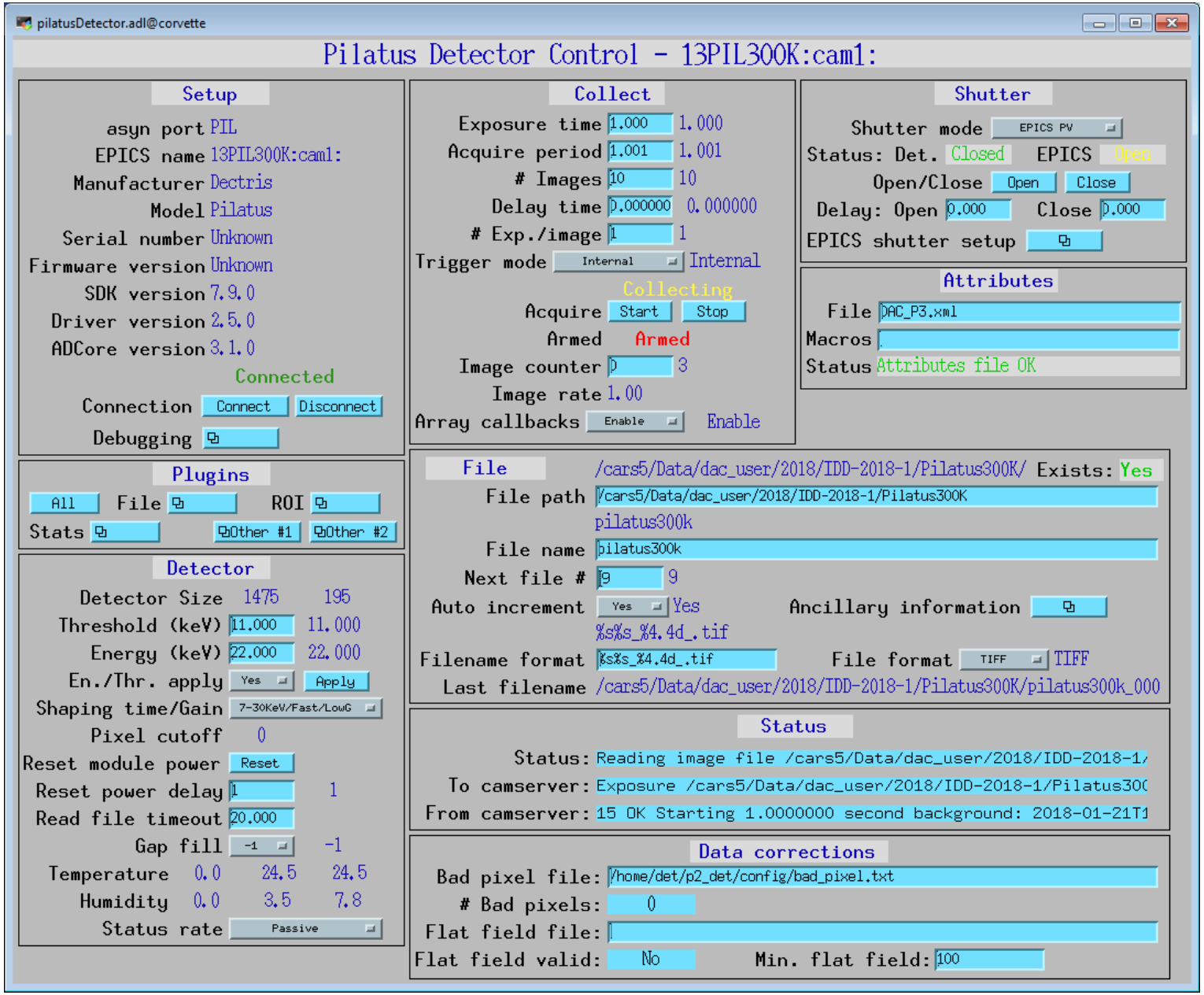


simDetector: Sine mode



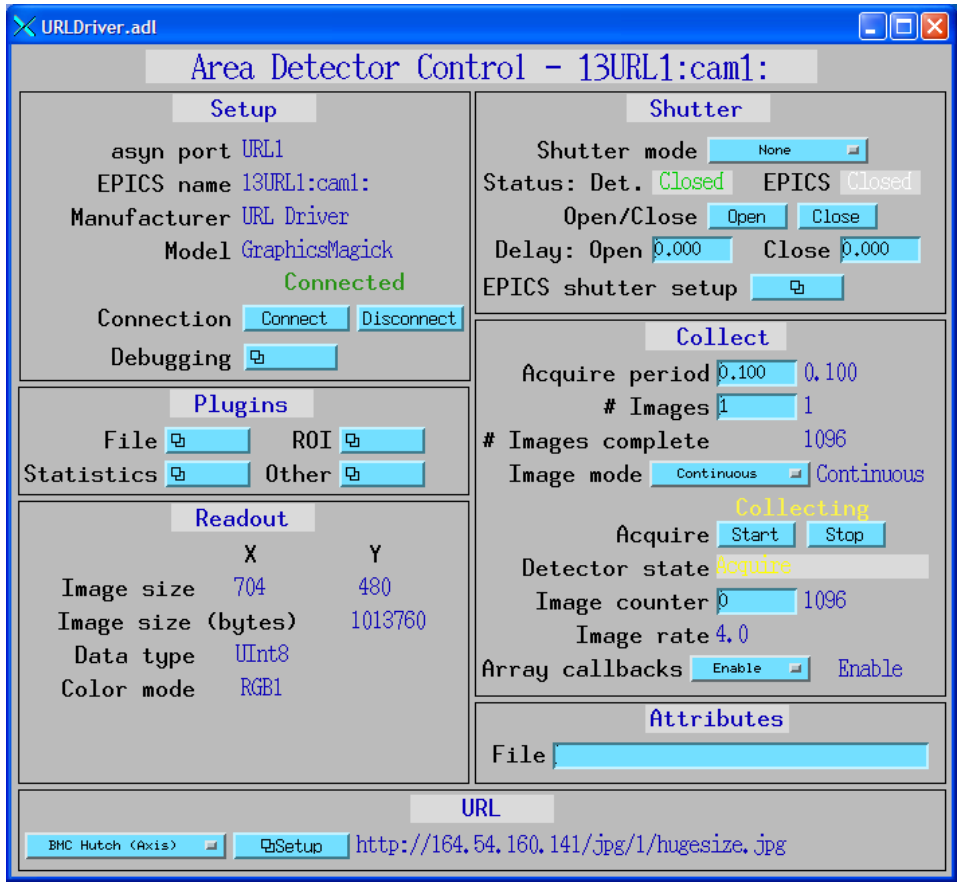


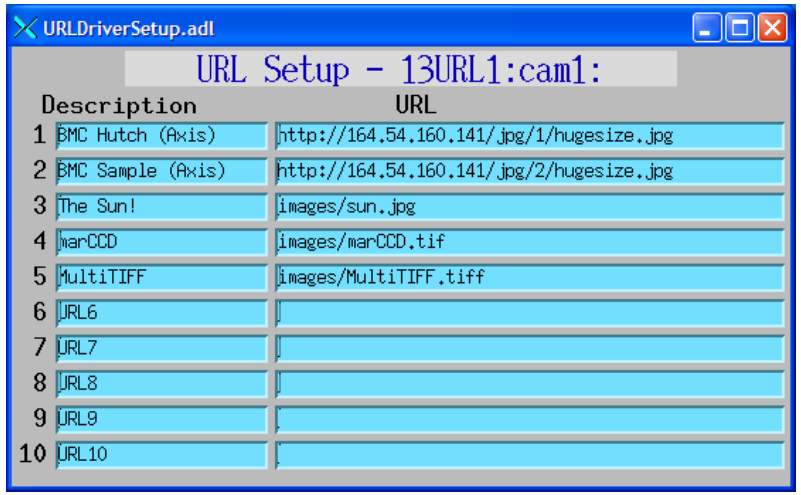
Pilatus specific control screen



URL Driver

- Driver that can read images from any URL.
- Can be used with Web cameras and Axis video servers.
- Uses GraphicsMagick to read the images, and can thus handle a large number of image formats (JPEG, TIFF, PNG, etc.).





GenICam

- Generic Interface for Cameras standard is the base for plug & play handling of cameras and devices.

  “The goal of GenICam™ (Generic Interface for Cameras) is to provide a generic programming interface for all kinds of devices (mainly cameras), no matter what interface technology (GigE Vision, USB3 Vision, CoaXPress, Camera Link HS, Camera Link etc.) they are using or what features they are implementing.

- GenApi
  - Defines the mechanism used to provide the generic API via a self-describing XML file in the device. Part of GenApi is the Schema, which defines the format of the XML file.
- SFNC (Standard Features Naming Convention)
  - Standardizes the name, type, meaning and use of device features, so that devices from different vendors always use the same names for the same functionality.
- GenTL (Transport Layer)
  - Standardizes the transport layer programming interface. It is a low-level API to provide a standard interface to a device regardless of the transport layer It allows enumerating devices, accessing device registers, streaming data and delivering asynchronous events. GenTL also has its own SFNC.

GenICam XML Files

- Every GenICam camera has an XML file inside it that can be accessed to determine the cameras available “features”.
- By reading and parsing this XML file one can automatically generate the EPICS database and OPI screens based on the specific features available in that camera. Small snippet for PixelFormat:

```
<Enumeration Name="PixelFormat" NameSpace="Standard">
  <ToolTip>Format of the pixel data.</ToolTip>
  <Description>Format of the pixel data.</Description>
  <DisplayName>Pixel Format</DisplayName>
  <Visibility> Beginner</Visibility>
  <pIsLocked>TLParamsLocked</pIsLocked>
  <ImposedAccessMode>RW</ImposedAccessMode>
  <EnumEntry Name="Mono8" NameSpace="Standard">
    <ToolTip>Pixel format set to Mono 8.</ToolTip>
    <Description>Pixel format set to Mono 8.</Description>
    <DisplayName>Mono 8</DisplayName>
    <pIsImplemented>Mono8Inq_Reg</pIsImplemented>
    <Value>0x01080001</Value>
  </EnumEntry>
```

GenICam and aravis

- GenICam is not open source
- There is a reference implementation of the GenICam API, but it cannot be released publicly
  - This is a pain, but we need to live with it
- aravis is an open-source toolkit based on reverse-engineering the GenICam specifications and protocols.
  - Based on glib. Until recently was Linux-only, now also supports Windows.
  - Previously supported only GigE, but now also supports USB

ADGenICam

- Vendor APIs for GenICam cameras are very similar, and programming straight to them results in lots of redundant code.
- Instead create a new areaDetector base class to support any GenICam camera
- Much of the generic code is in this layer
  - Implements code to get and set GenICam features
- Creates the areaDetector parameter library dynamically at iocInit from the drvUser fields passed by each record.

ADGenICam

- Derived classes for real drivers
  - Implement the code to read and write features to the device
  - Implement the code to stream the images from the device
- ADAravis
  - Uses aravis library
  - Linux only (most versions, e.g. RHEL 7). Could in principle now make it work with Windows.
- ADSpinnaker
  - Driver for FLIR/Point Grey cameras using their Spinnaker SDK
  - Windows and new versions of Linux (e.g. Ubuntu 18, not RHEL 7)
- ADVimba
  - Driver for AVT/Prosilica cameras using their Vimba SDK
  - Windows and Linux (most versions, e.g. RHEL 7)
- ADVimba
  - Driver for AVT/Prosilica cameras using their Vimba SDK
  - Windows and Linux (most versions, e.g. RHEL 7)

arv-tool (from aravis package)

- Find the cameras

```
corvette: ADAravis>bin/linux-x86_64/arv-tool-0.6
Allied Vision Technologies-02-2142A-06178 (164.54.160.58)
Allied Vision Technologies-02-2604A-07008 (164.54.160.104)
Allied Vision Technologies-50-0503317598 (164.54.160.62)
Allied Vision Technologies-50-0503419258 (164.54.160.21)
PointGrey-13481965 (164.54.160.114)
Prosilica-02-2142A-06110 (164.54.160.57)
```

- Extract the XML file

```
>arv-tool-0.6 -n PointGrey-13481965 genicam > BFly-20E4C.xml
```

- Look at the XML file

```
>more BFly-20E4C.xml
PointGrey-13481965 (164.54.160.114)
<?xml version="1.0" encoding="UTF-8"?>
<!–
<RegisterDescription
xmlns:xi="http://www.w3.org/2003/XInclude"
…
```

- Must run this once on Linux for each new camera type to create XML file.

Python tool to create template file from XML file

- Create the database with makeDb.py

```
>scripts/makeDb.py BFly-20E4C.xml BFly-20E4C.template
```

- Snippet of the template file

```
record(ai, "$(P)$(R)GC_ExposureTime_RBV") {
  field(DTYP, "asynFloat64")
  field(INP,  "@asyn($(PORT),$(ADDR=0),$(TIMEOUT=1))GC_D_ExposureTime")
  field(PREC, "3")
  field(SCAN, "I/O Intr")
  field(DISA, "0")
}
record(ao, "$(P)$(R)GC_ExposureTime") {
  field(DTYP, "asynFloat64")
  field(OUT,  "@asyn($(PORT),$(ADDR=0),$(TIMEOUT=1))GC_D_ExposureTime")
  field(PREC, "3")
  field(DISA, "0")
}
```

- Creates output and readback record for each feature that the camera supports.

Python tool to create medm files from XML file

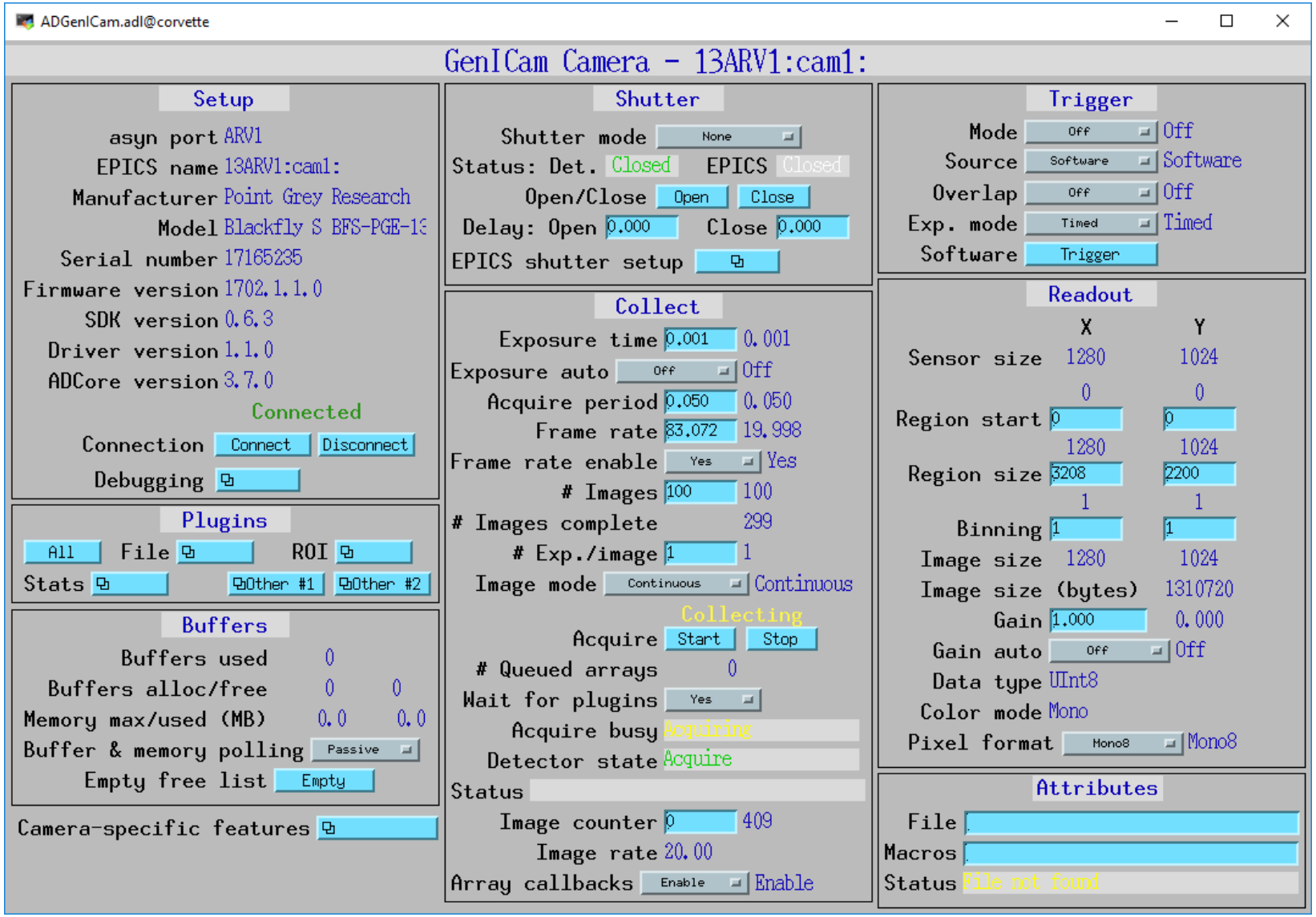
- Create the medm files with makeAdl.py

```
>scripts/makeAdl.py  BFly-20E4C.xml BFly-20E4C
>ls -l *.adl
-rw-rw-r-- 1 epics domain users 54593 Apr 17 16:53 BFly-20E4C-features_1.adl
-rw-rw-r-- 1 epics domain users 48936 Apr 17 16:53 BFly-20E4C-features_2.adl
-rw-rw-r-- 1 epics domain users 21322 Apr 17 16:53 BFly-20E4C-features_3.adl
```

- Snippet of the adl file

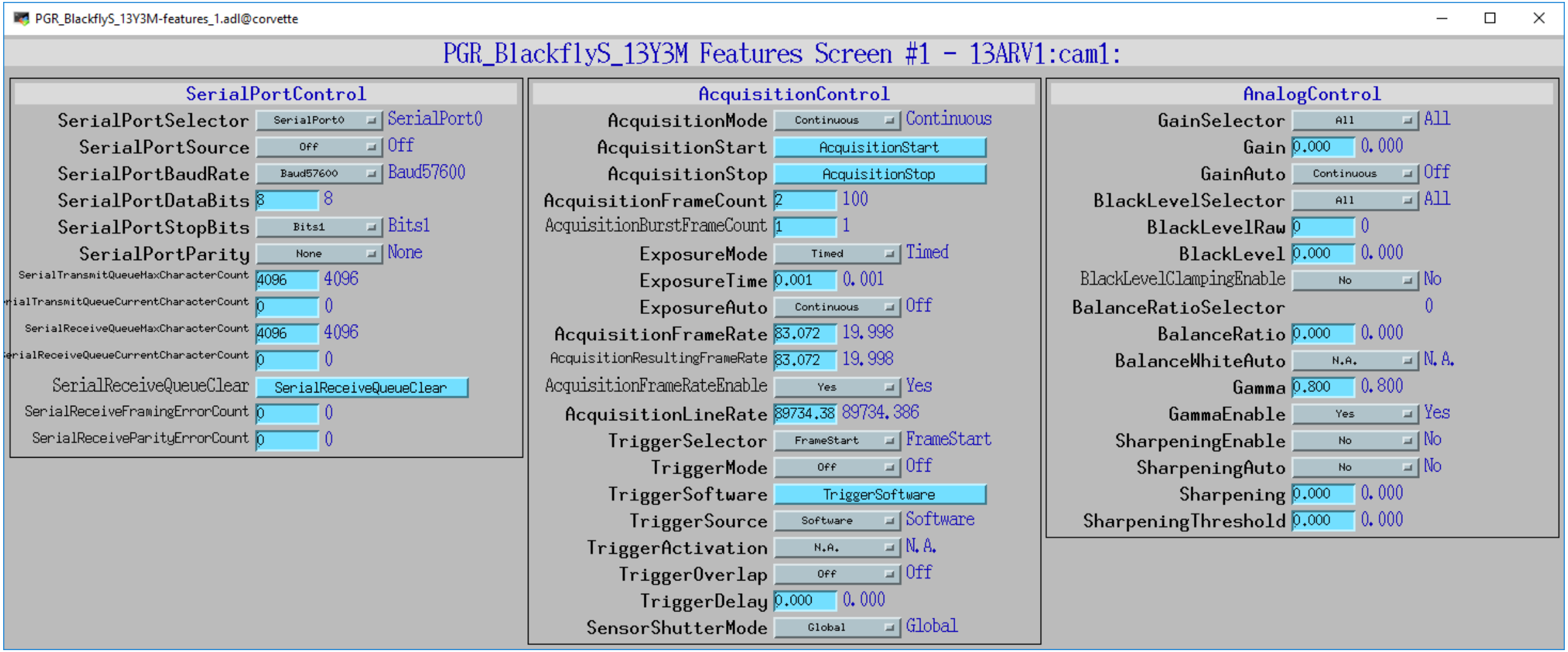
```
# (Menu Button)
 menu {
   object {
     x=235
     y=520
     width=150
     height=20
   }
   control {
     chan="$(P)$(R)GC_SaturationAuto"
     clr=14
     bclr=51
   }
 }
```

Main medm screen for ADGenICam



Identical screens are provided for CSS/BOY, Phoebus, caQtDM and EDM via auto-convert

Auto-generated medm screens for PGR BlackflyS 13Y3M Screen #1



FLIR/Point Grey drivers

- ADSpinnaker driver for all cameras from FLIR/Point Grey using their Spinnaker SDK.
- GigE, USB 3.0, and 10 GigE camera
- High performance, low cost
- I will demonstrate USB-3.0 camera today
- Example models:

Model	Interface	Resolution	Price	Measured Speed (frames/s)	Measured Speed (MB/s)
BlackFlyS BF2-PGE-13Y3M	1-Gbit Ethernet	1280x1024	$415	83 frames/s	104 MB/s
Grasshopper3 GS3-U3-23S6M	USB-3	1920x1200	$1,179	156 frames/s	343 MB/s
Oryx ORX-10G-51S5M	10-Bbit Ethernet	2448x2048	$2,258	163 frames/s	779 MB/s

FLIR/Point Grey GigE Camera
BlackFly PGE-20E4C

- e2v EV76C570 CMOS sensor
- Global shutter
- 29 x 29 x 30 mm
- Power Over Ethernet
- 4.5 micron pixels
- 1600 x 1200 pixels, color or mono
- 50 frames/s
- $525

FLIR/Point Grey USB-3.0 Camera Grasshopper3 GS3-U3-23S6M

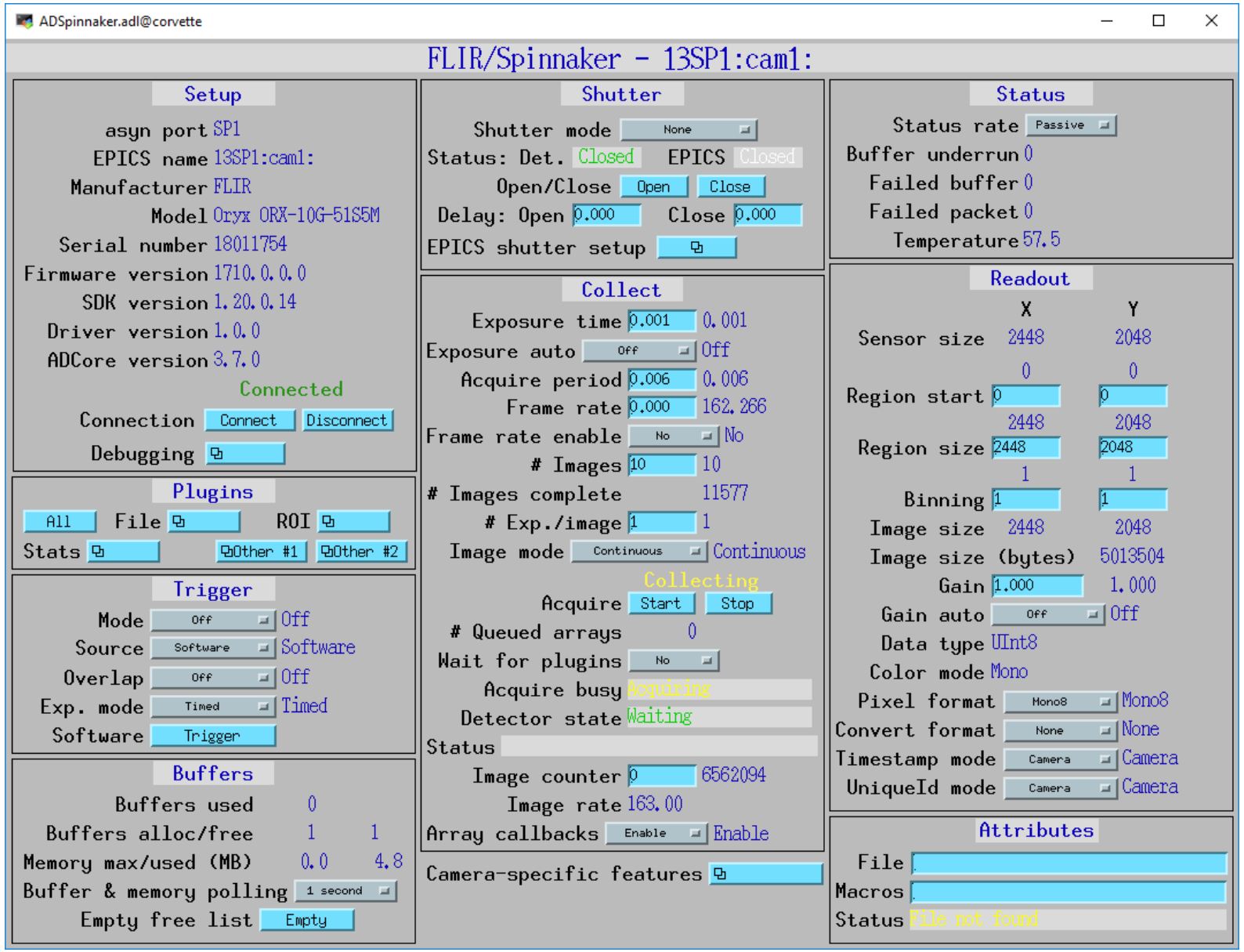
- 1920 x 1200 global shutter CMOS
- Sony IMX174 1/1.2
- Dynamic range of 73 dB
- Peak QE of 76%
- Read noise of 7e-
- 12-bit or 8-bit data
- Max frame rate of 162 fps
  - ~356 MB/S, >3X faster than GigE
- USB 3.0 interface
- Used for tomography at 3 APS beamlines, replaced Andor Neo and PCO Edge
- $1,179

Point Grey 10-Gbit Ethernet Camera
Oryx ORX-10G-310S9M

- 6464 x 4852 global shutter CMOS (31 MP)
- 10-Gbit Ethernet interface
- Sony IMX342 3.45 micron pixels
- Dynamic range of 66 dB
- Peak QE of 65%
- Read noise of 5.2e-
- 12-bit, 10-bit, or 8-bit data
- Max frame rate of 26 fps
  - 778 MB/S, >8X faster than GigE
- Used for tomography on 2 APS beamlines

Model	Resolution	Price	Speed (frames/s)	Speed (MB/s)
ORX-10G-310S9M	6464x4852	$7,089	26 frames/s	778 MB/s
ORX-10G-123S6M	4096x3000	$4,666	68 frames/s	797 MB/s
ORX-10G-51S5M	2448x2048	$2,258	162 frames/s	774 MB/s
ORX-10GS-34S4M	2048x1536	$1,774	216 frames/s	648 MB/s

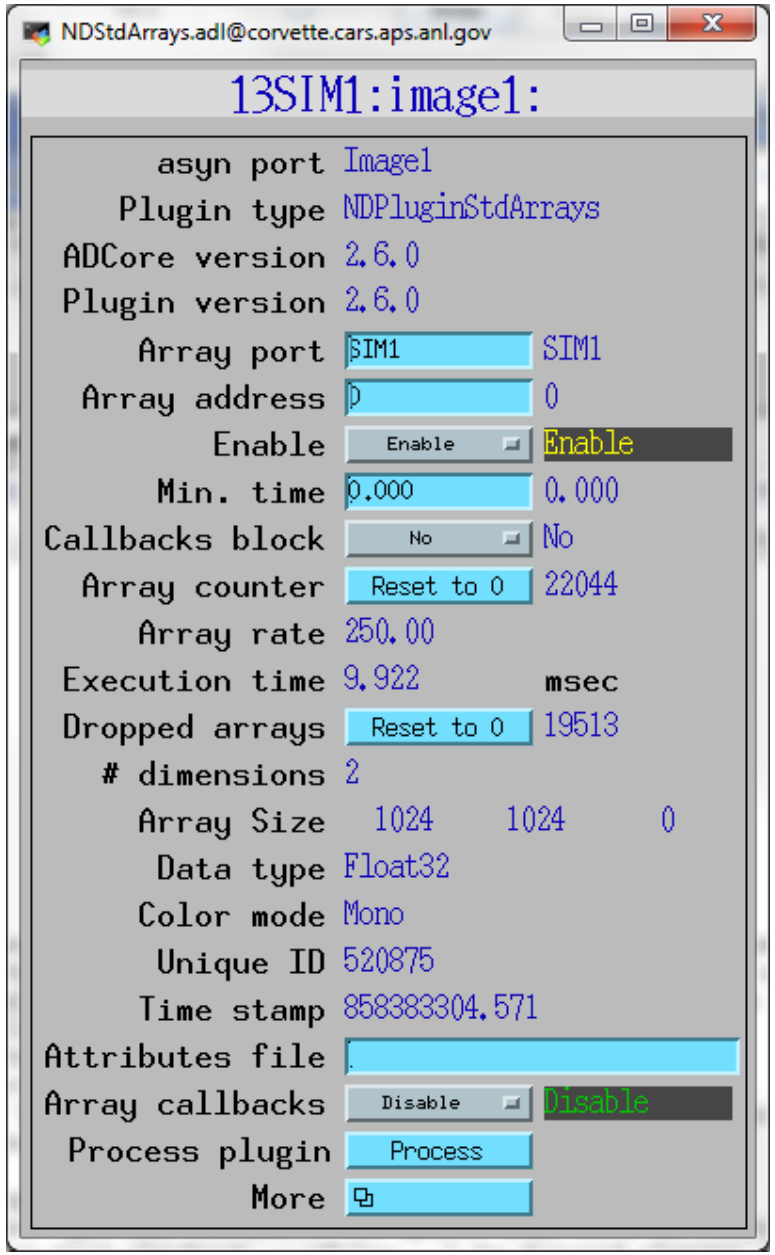
ADSpinnaker Driver

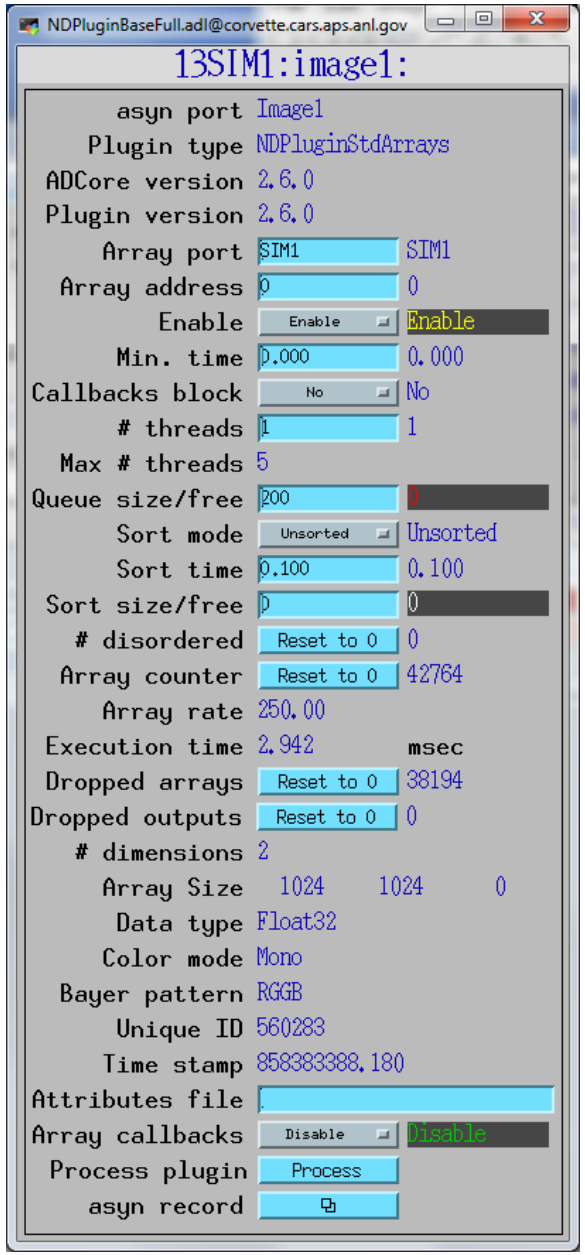


Plugins

- Designed to perform real-time processing of data, running in the EPICS IOC (not over EPICS Channel Access)
- Receive NDArray data over callbacks from drivers or other plugins
- Plug-ins can execute in their own threads (non-blocking) or in callback thread (blocking)
  - If non-blocking then NDArray data is queued
  - If executing in callback thread, no queuing, but slows driver
- Allows
  - Enabling/disabling
  - Throttling rate (no more than 0.5 seconds, etc)
  - Changing data source for NDArray callbacks to another driver or plugin
- Plugins can be *sources* of NDArray callbacks, as well as *consumers*
  - Allows creating a data processing pipeline running at very high speed, each in a different thread, and hence in multiple cores on modern CPUs.

NDPluginDriver medm Screens





Plugins (continued)

- Currently ~20 plugins that perform wide variety of operations
- NDPlugInStdArrays
  - Receives arrays (images) from device drivers, converts to standard arrays, e.g. waveform records.
  - This plugin is what EPICS channel access viewers normally talk to.
- NDPluginPVA
  - Converts NDArrays to EPICS V4 NTNDArrays
  - Exports the NtNDArrays over PVAccess with internal V4 server
  - Can be used to send structured data to EPICS V4 clients
  - When used with the PVAccess driver then areaDetector plugins can be run on different machine from the detector driver
- NDPluginROI
  - Performs region-of-interest calculations
  - Select a subregion. Optionally bin, reverse in either direction, convert data type.
  - Divide the array by a scale factor, which is useful for avoiding overflow when binning.
- NDPluginTransform
  - Performs geometric operations (rotate, mirror in X or Y, etc.)

Plugins (continued)

- NDPluginStats
  - Calculates basic statistics on an array (min, max, sigma)
  - Optionally computes centroid centroid position, width and tilt.
  - Optionally Computes X and Y profiles, including average profiles, profiles at the centroid position, and profiles at a user-defined cursor position.
  - Optionally computes the image histogram and entropy
- NDPluginROIStat
  - Multiple ROIs with simple statistics in a single plugin
  - More efficient when many ROIs are needed, e.g. for peaks in a 1-D energy spectrum
  - Min, max, total, net, mean
  - Time-series of each of these statistics

Plugins (continued)

- NDPluginProcess
  - Does arithmetic processing on arrays
  - Background subtraction.
  - Flat field normalization.
  - Offset and scale.
  - Low and high clipping.
  - Recursive filtering in the time domain.
  - Conversion to a different output data type.
- NDPluginOverlay
  - Adds graphic overlays to an image.
  - Can be used to display ROIs, multiple cursors, user-defined boxes, text, etc.
- ffmpegServer
  - MJPEG server that allows viewing images in a Web browser. From DLS.

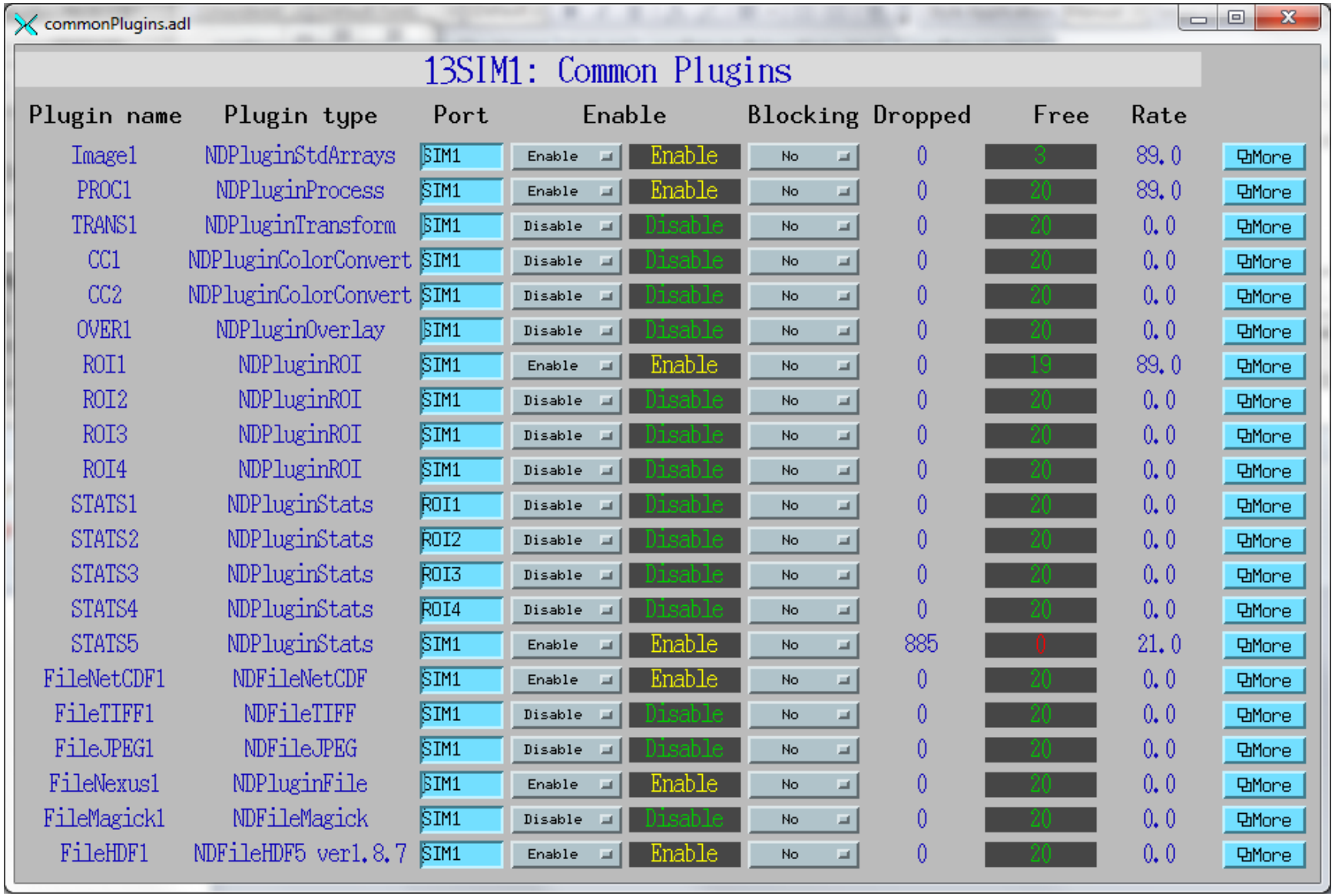
Plugins (continued)

- NDPluginAttribute
  - Extracts NDAttributes from NDArrays and publishes their values as ai records
  - Can collect time-series arrays of the attribute values
- NDPluginCircularBuff
  - Buffers NDArrays in a circular buffer
  - Computes a trigger expression using up to 2 NDAttribute values
  - When trigger condition is met then outputs NDArrays
  - User-specified number of pre-trigger and post-trigger arrays to output
- NDPluginTimeSeries
  - Accepts 1-D NDArrays[NumSignals] or 2-D [NumSignals,NewTimePoints] and appends to time-series buffer
  - Operates in fixed length (stop when full) or circular buffer modes
  - Optional time-averaging of input data

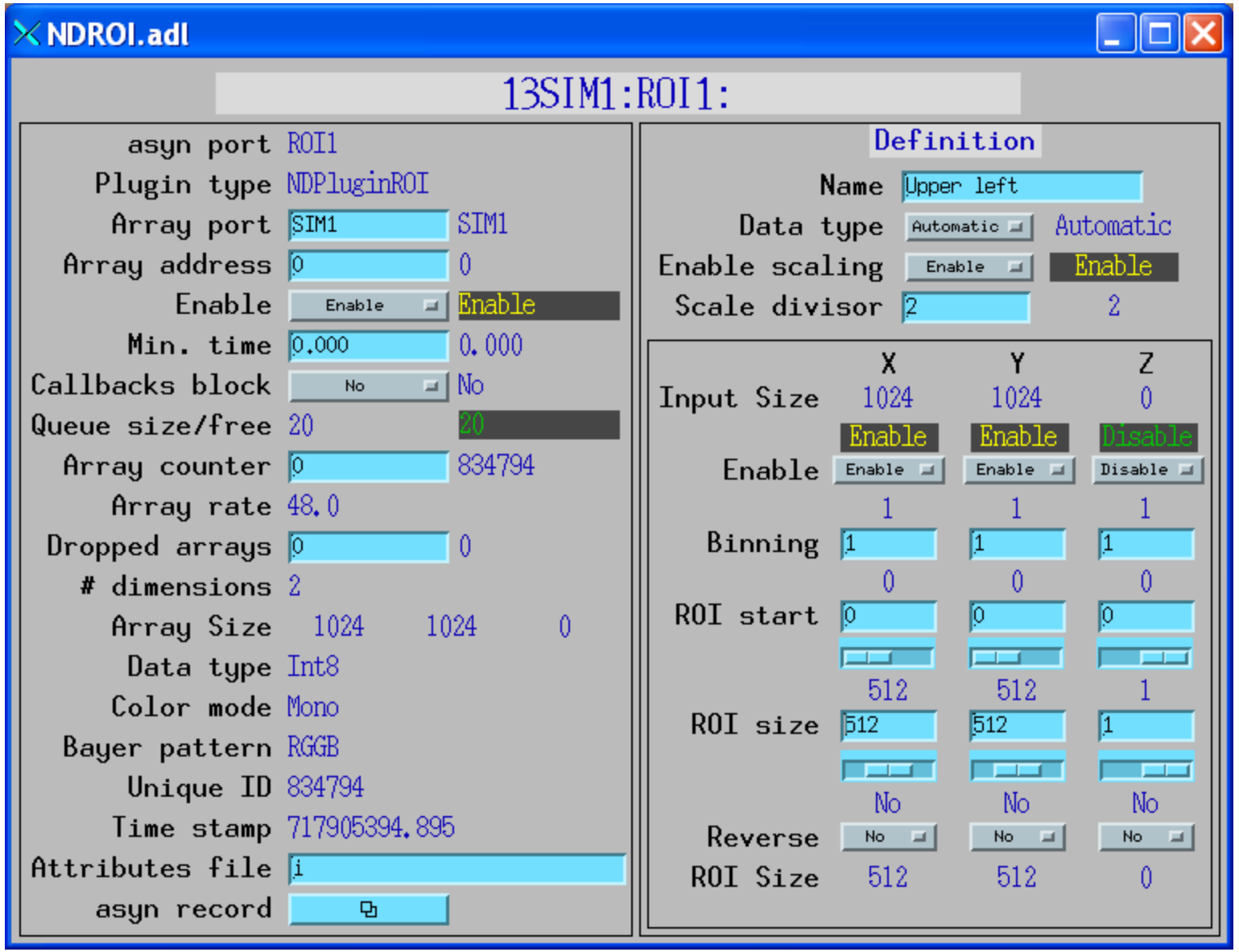
Plugins (continued)

- NDPluginFFT
  - Computes FFT of 1-D or 2-D NDArrays
  - Exports NDArrays containing the absolute value (power spectrum) of the FFT
  - Exports 1-D arrays of the FFT real, imaginary, absolute values, and time and frequency data.
- NDPluginColorConvert
  - Convert from one color model to another (Mono, RGB1 (pixel), RGB2 (row) or RGB3 (planar) interleave)
  - Bayer conversion removed from this plugin, now part of Prosilica and Point Grey drivers.

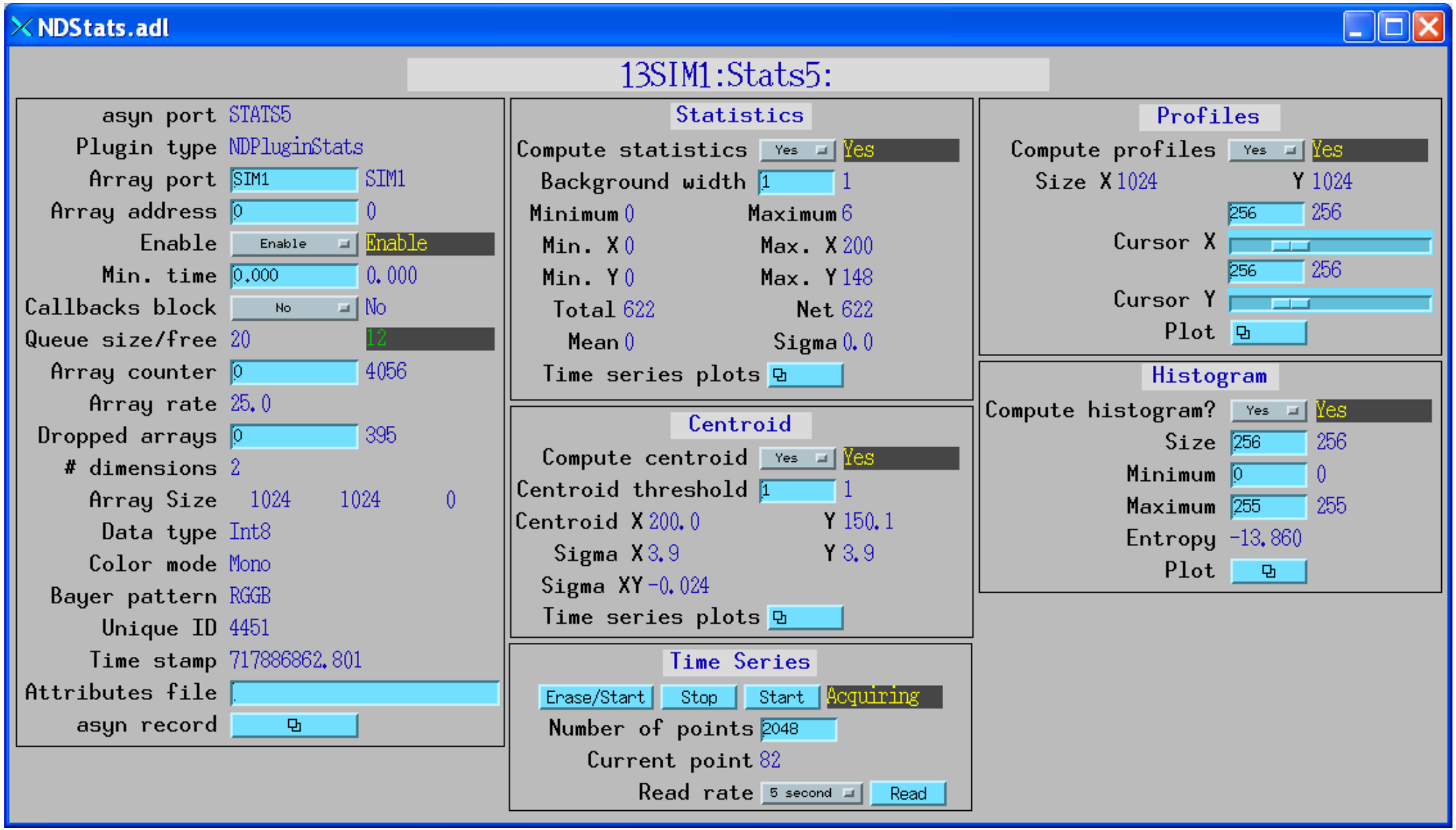
commonPlugins.adl All plugins at a glance



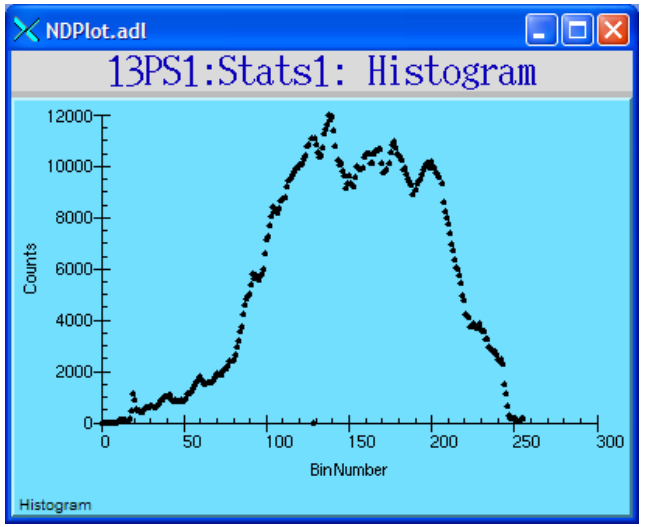
ROI plugin

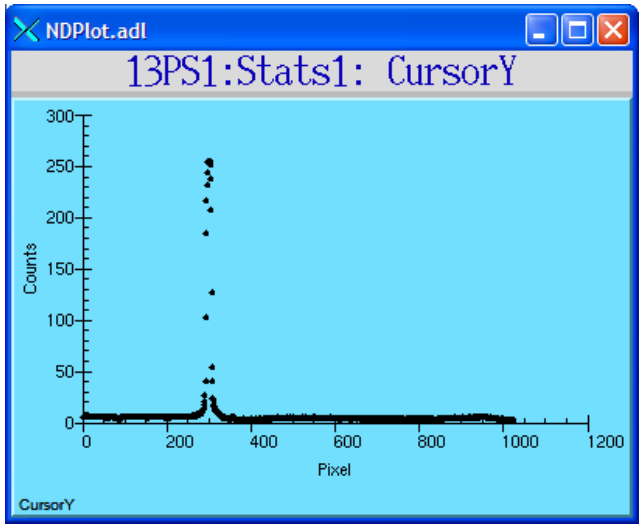

NDROI.adl
13SIM1:ROI1:
asyn port ROI1
Plugin type NDPluginROI
Array port SIM1 SIM1
Array address 0 0
Enable Enable Enable
Min. time 0.000 0.000
Callbacks block No No
Queue size/free 20 20
Array counter 0 834794
Array rate 48.0
Dropped arrays 0 0
dimensions 2
Array Size 1024 1024 0
Data type Int8
Color mode Mono
Bayer pattern RGGB
Unique ID 834794
Time stamp 717905394.895
Attributes file i
asyn record
Definition
Name Upper left
Data type Automatic Automatic
Enable scaling Enable Enable
Scale divisor 2 2
X Y Z
Input Size 1024 1024 0
Enable Enable Enable Disable
Enable Enable Enable Disable
1 1 1
Binning 1 1 1
0 0 0
ROI start 0 0 0
512 512 1
ROI size 512 512 1
No No No
Reverse No No No
ROI Size 512 512 0

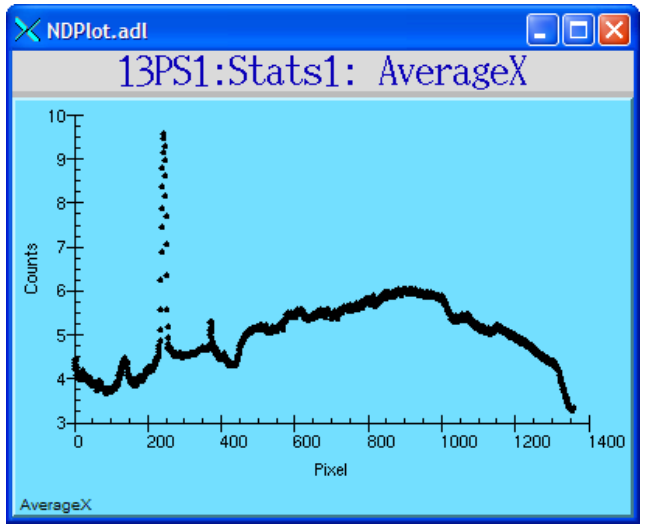
Statistics plugin

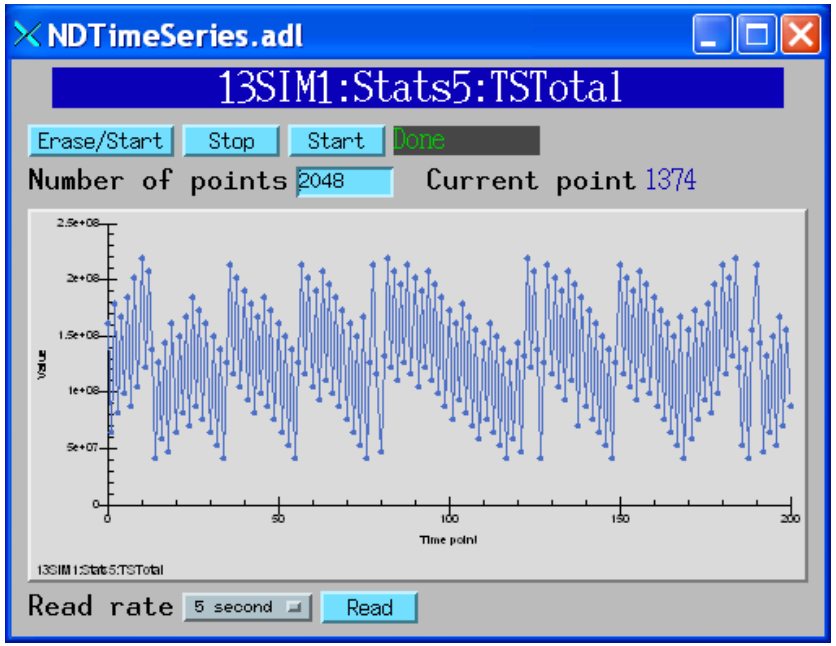


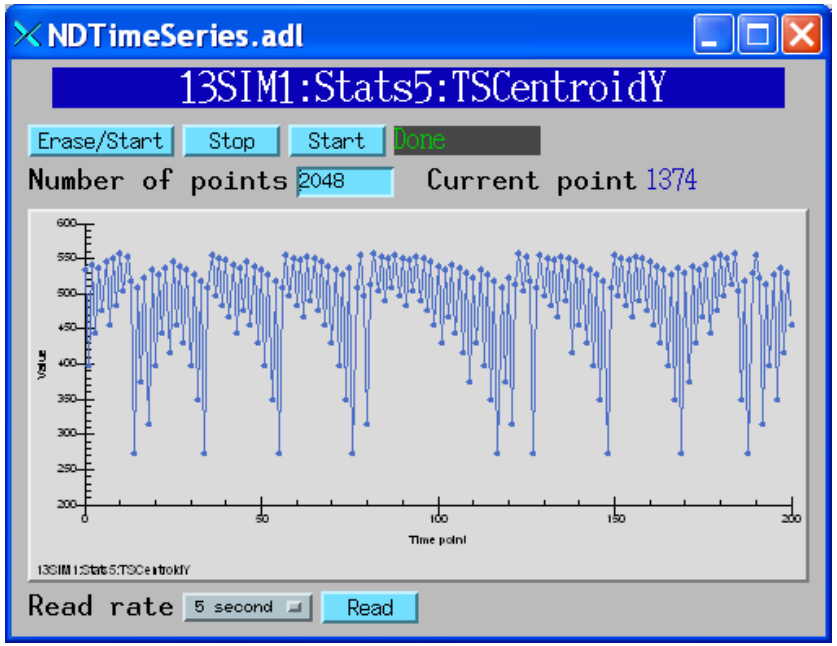
Statistics plugin (continued)



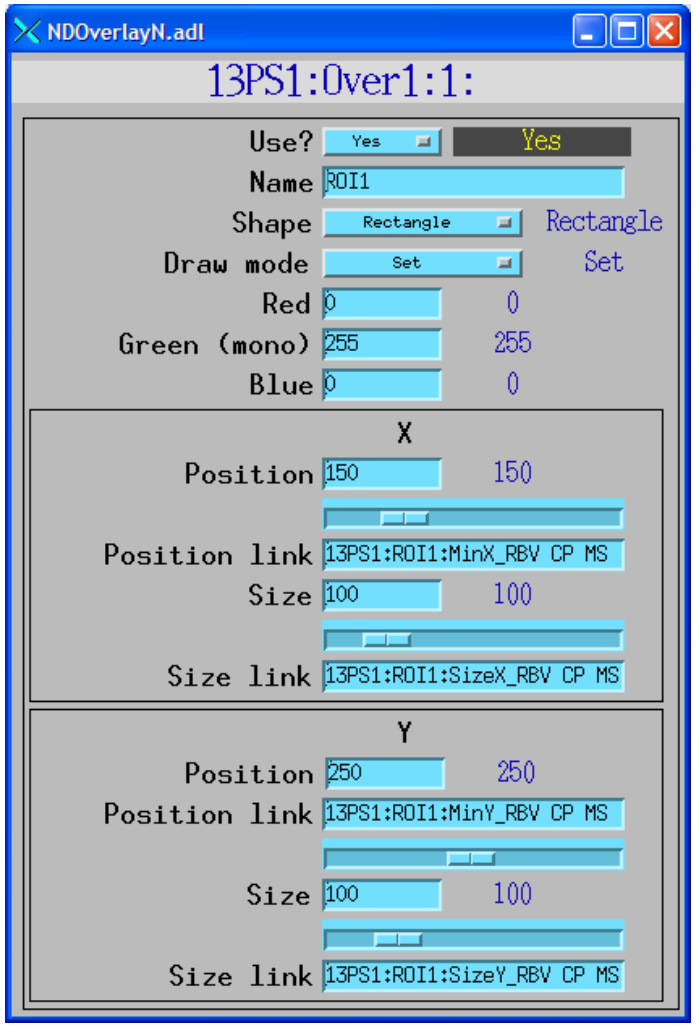


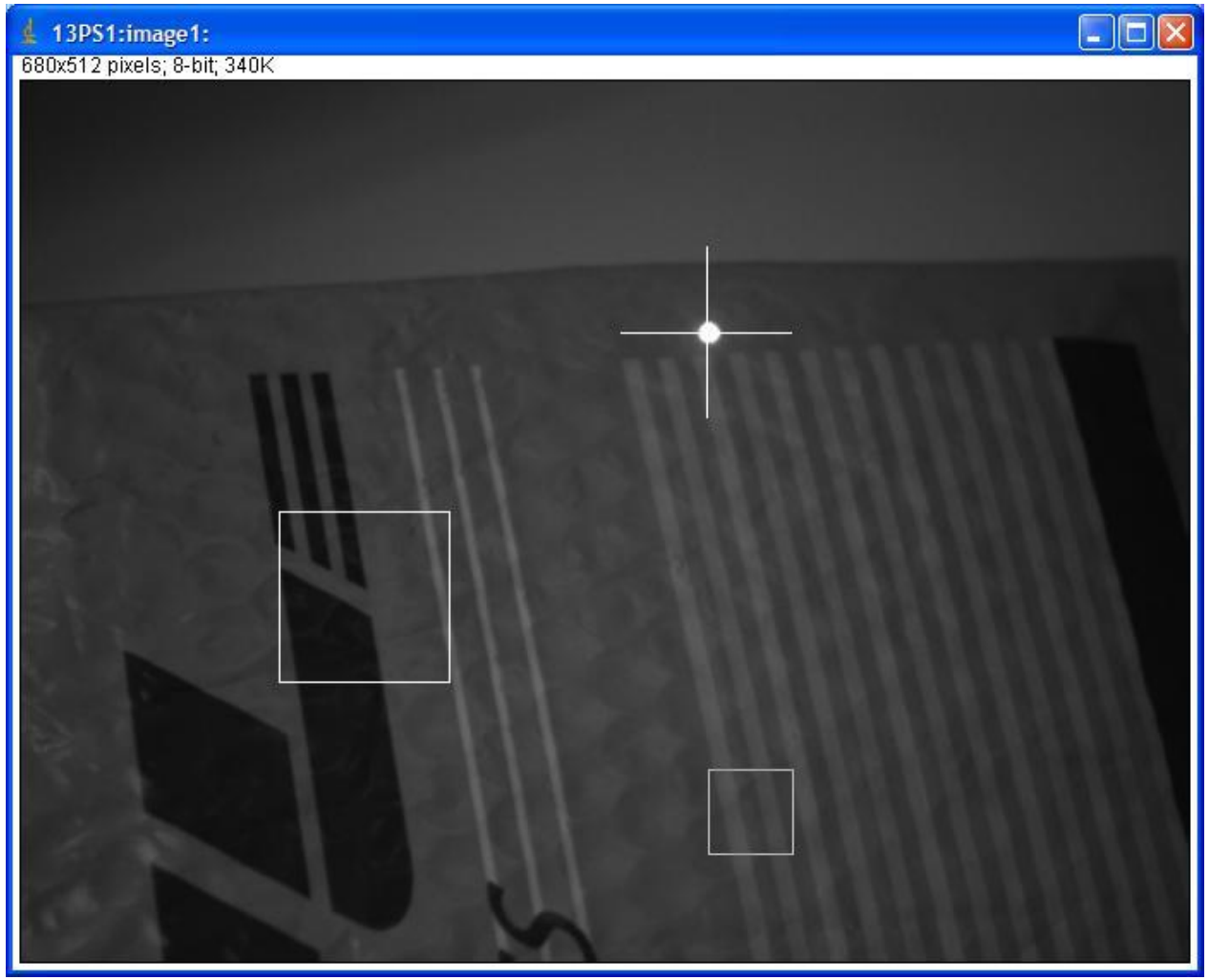






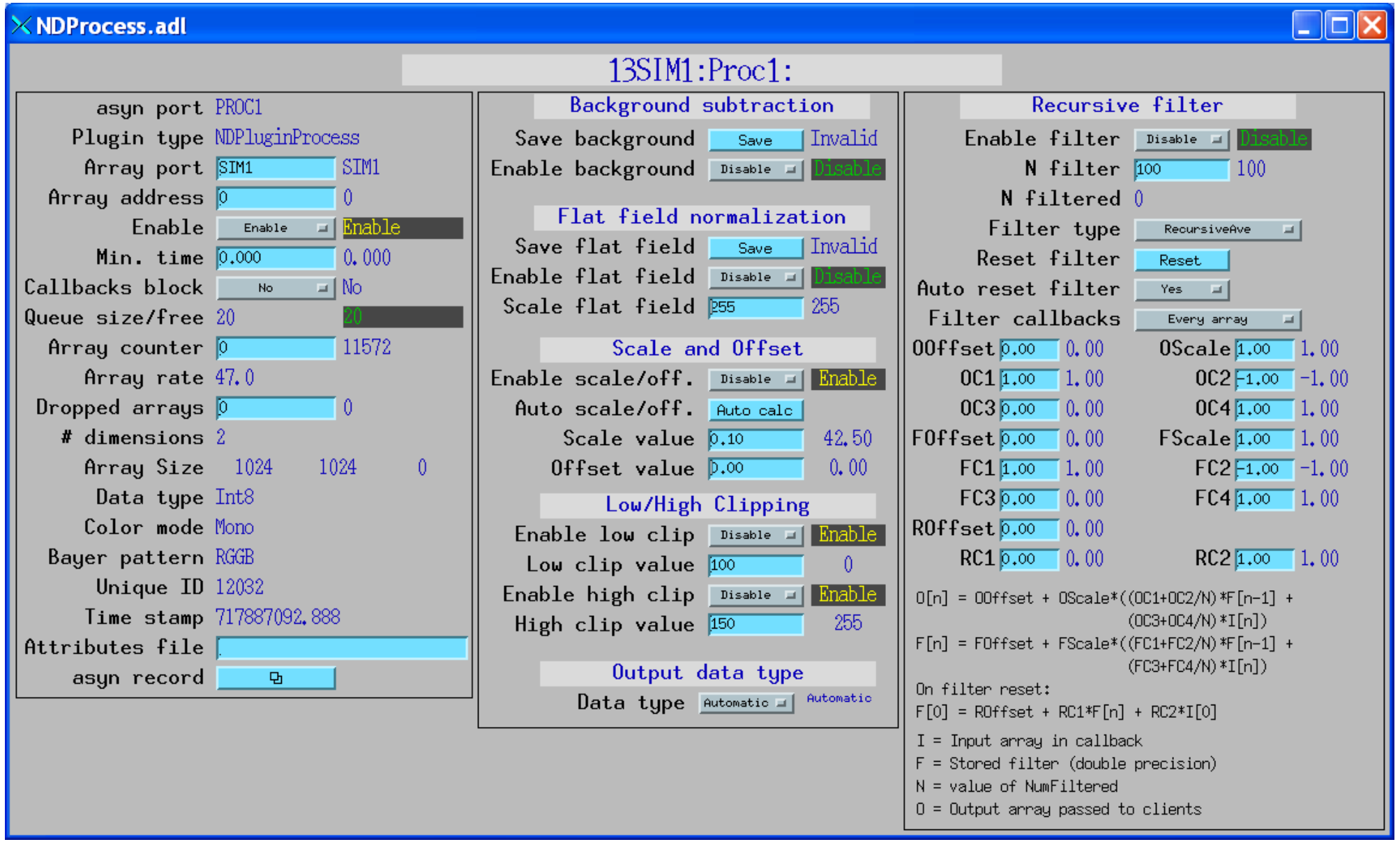
Overlay plugin





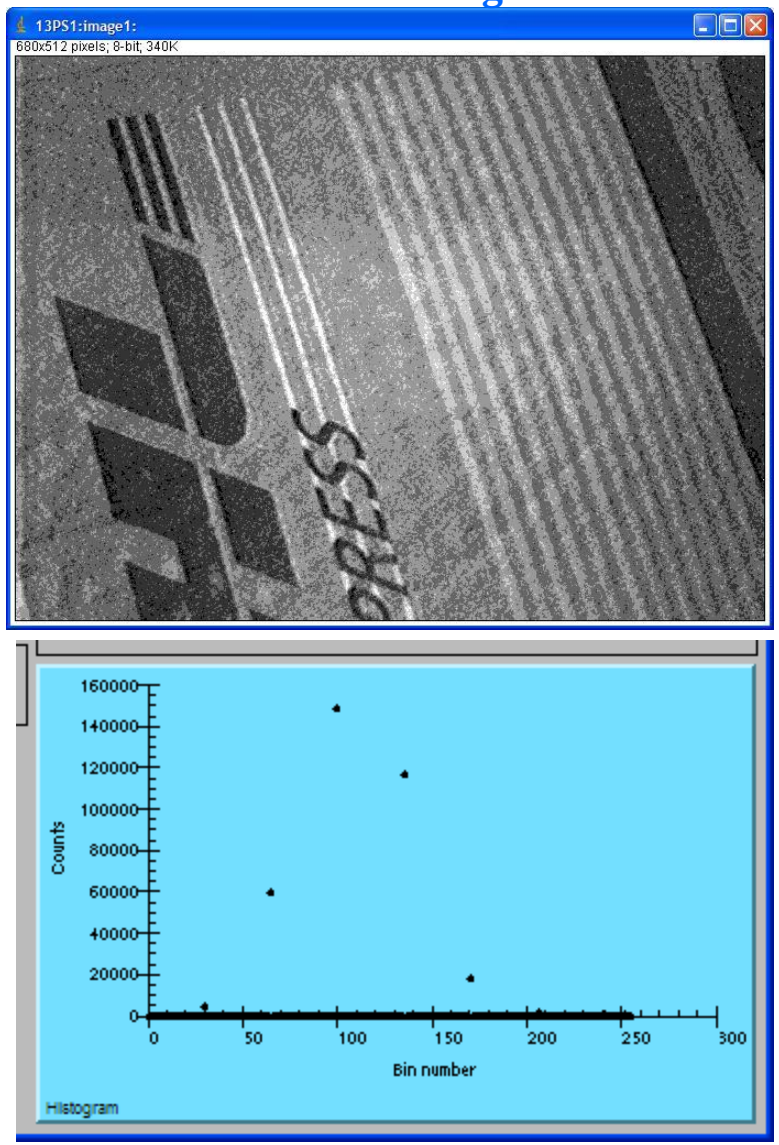
- Centroid of laser pointer calculated by statistics plugin
- Cursor overlay X, Y position linked to centroid

Processing plugin

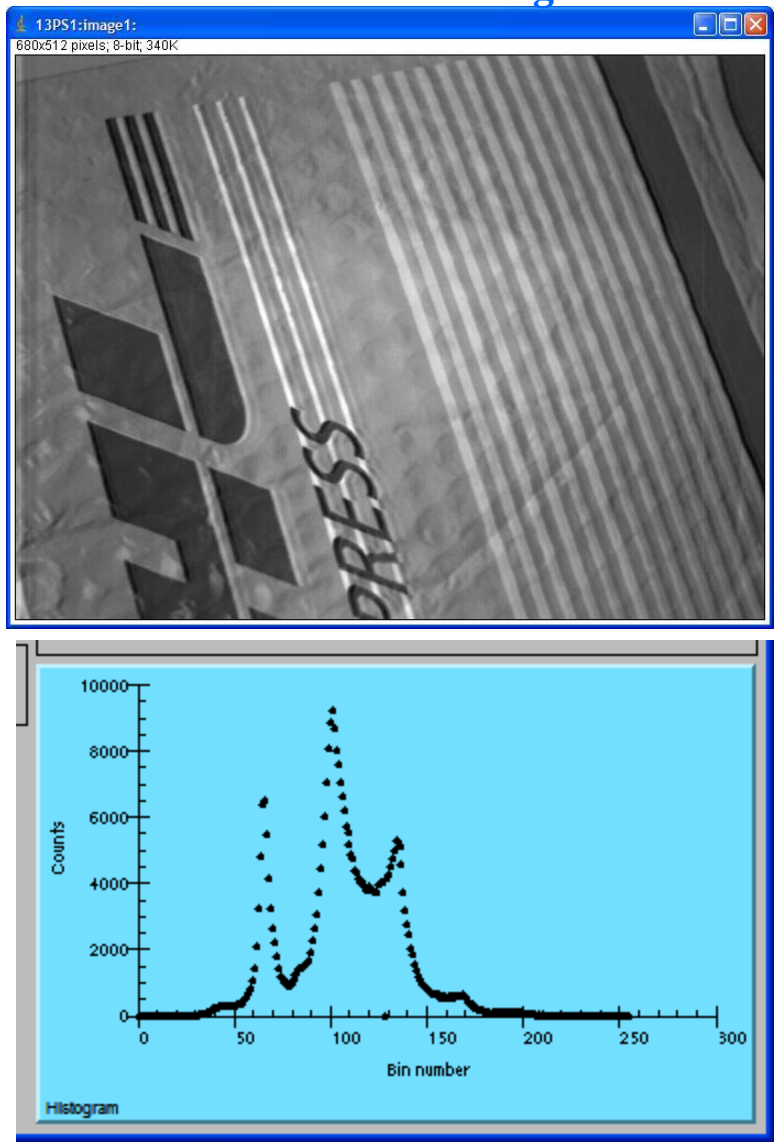


Processing plugin
30 microsec exposure time

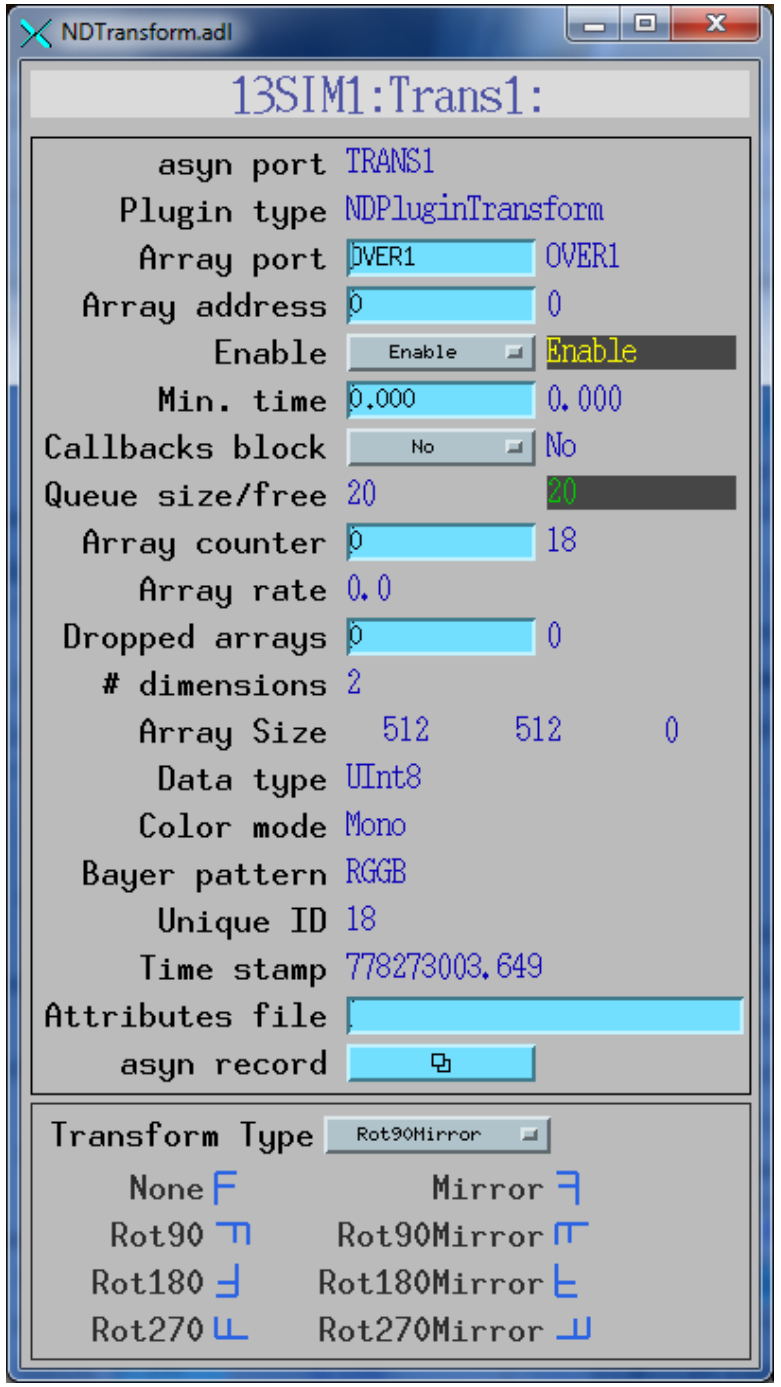
No filtering



N=100 recursive average filter

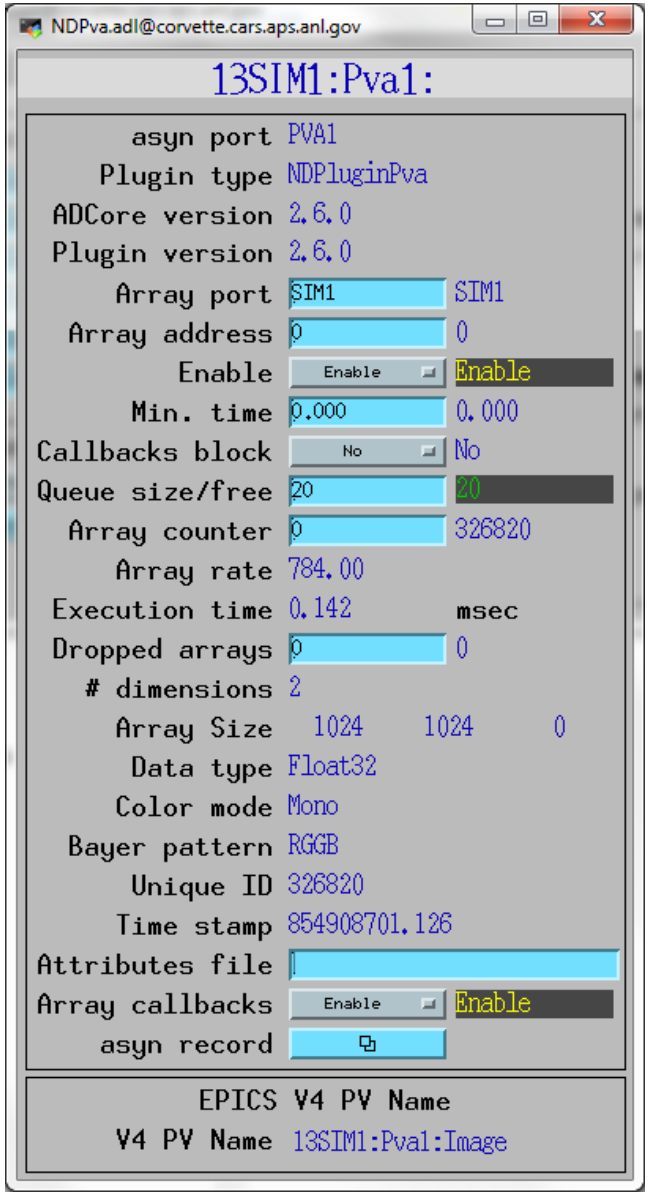


Transform plugin



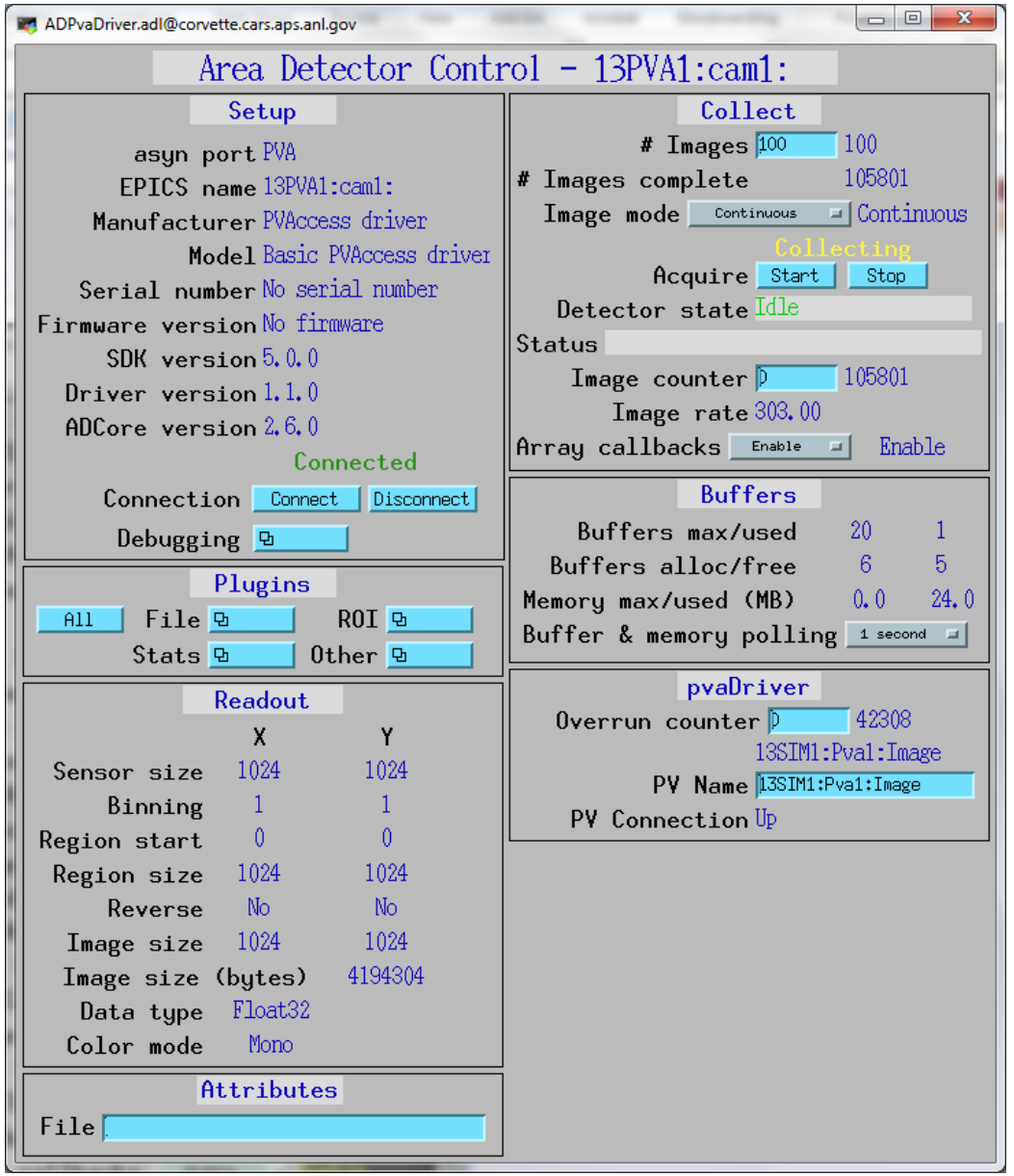
NDPluginPva (EPICS 7)

- Converts NDArrays into the EPICS PVA normative type NTNDArray
- Embedded EPICSv4 server serves the new NTNDArray structure over PVAccess
- High performance, ~3.2GB/s shown here
- Can be received by any EPICS v4 client
  - Java, Python, C++ versions of pvAccess
  - CSS has a widget that can display NTNDArrays
  - ImageJ plugin
  - Can include an NTNDArray receiver in another IOC



pvAccess Driver (EPICS V4)

- Logical inverse of NDPluginPva
- Receives NTNDArrays over the network, converts to NDArrays and calls plugins
- Can be used to run areaDetector IOC and plugins on another machine or in another process
- High performance:
  - ~1.2 GB/s shown here with interprocess communication
  - Saturating 10 Gb Ethernet links has been demonstrated



Plugins: NDPluginFile

- Saves NDArrays to disk
- 3 modes:
  - Single array per disk file
  - Capture N arrays in memory, write to disk either multiple files or as a single large file (for file formats that support this.)
  - Stream arrays to a single large disk file
- For file formats that support it, stores not just NDArray data but also NDAttributes

Plugins: NDPluginFile

- NDFileTIFF
  - Supports any NDArray data type
  - Stores NDAttributes as ASCII user tags
- NDFileJPEG
  - With compression control
  - No NDAttributes
- NDFileNetCDF
  - Popular self-describing binary format, supported by Unidata at UCAR
  - NDAttributes are written
- NDFileHDF5
  - Writes HDF5 files with the native HDF5 API, unlike the NeXus plugin which uses the NeXus API. Supports 3 types of compression.
  - Supports using an XML file to define the layout and placement of NDArrays and NDAttributes in the HDF5 file
  - Supports Single Writer Multiple Reader (SWMR). Only supported on local file systems, GPFS, and Lustre (not NFS or SMB)

Plugins: NDPluginFile

- NDFileNeXus
  - Standard file format for neutron and x-ray communities, based on HDF5, which is another popular self-describing binary format; richer than netCDF
  - May be deprecated in a future release since NeXus files can now be produced with the NDFileHDF5 plugin using an appropriate XML layout file
- NDFileMagick
  - Uses GraphicsMagick to write files, and can write in dozens of file formats, including JPEG, TIFF, PNG, PDF, etc.
- NDFileNull
  - Used only to delete original driver files when no other file plugin is running

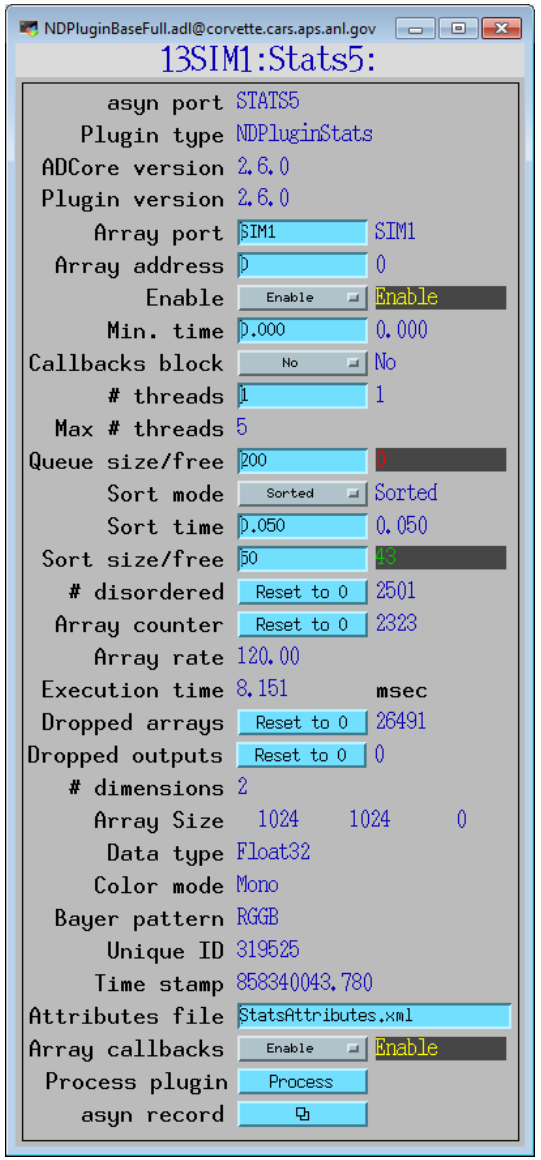
NDFileHDF5
XML file to define file layout

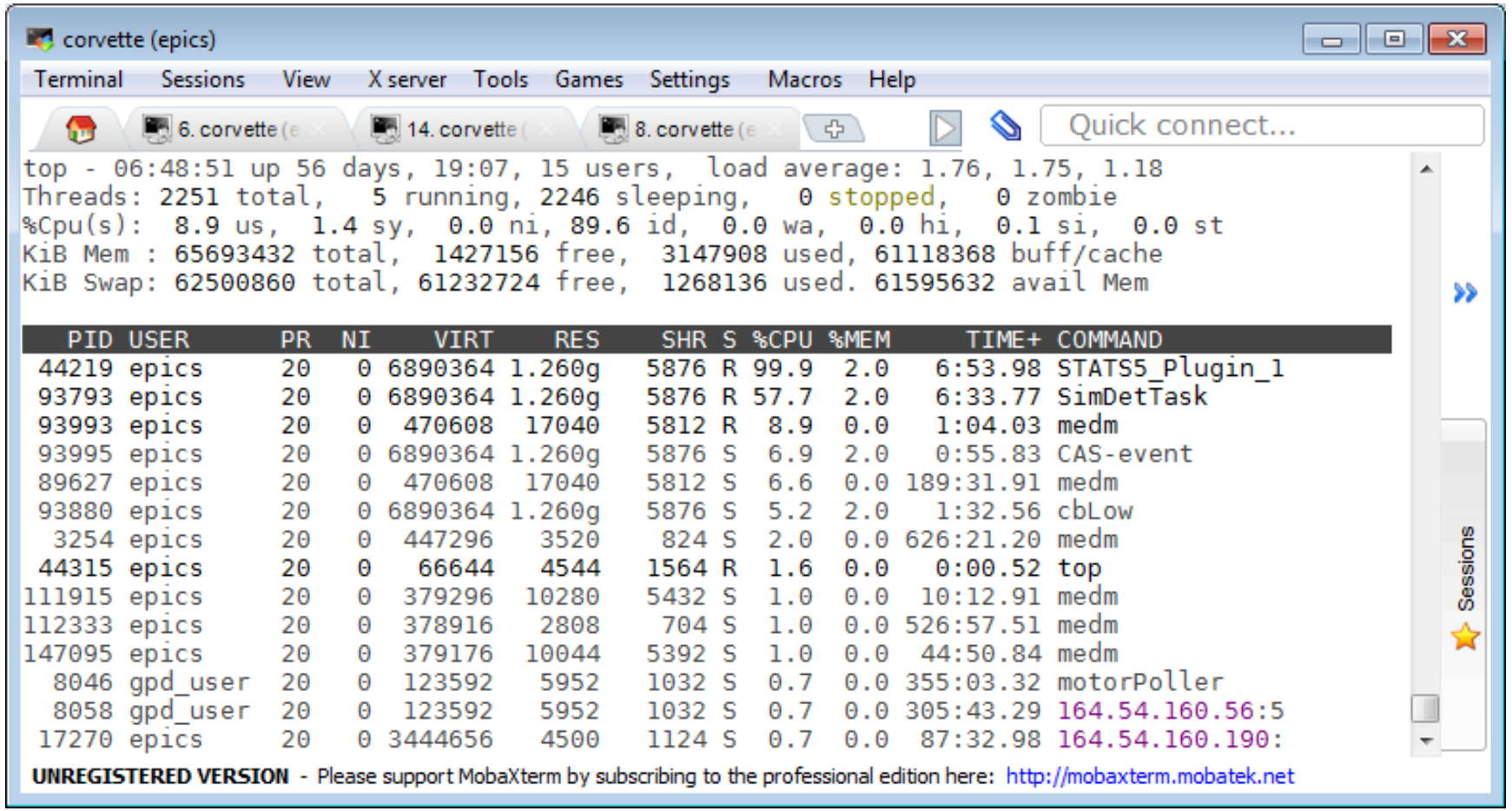
```
<xml>
  <group name="entry">
    <attribute name="NX_class" source="constant" value="NXentry" type="string"></attribute>
    <group name="instrument">
      <attribute name="NX_class" source="constant" value="NXinstrument" type="string"></attribute>
      <group name="detector">
        <attribute name="NX_class" source="constant" value="NXdetector" type="string"></attribute>
        <dataset name="data" source="detector" det_default="true">
          <attribute name="NX_class" source="constant" value="SDS" type="string"></attribute>
          <attribute name="signal" source="constant" value="1" type="int"></attribute>
          <attribute name="target" source="constant" value="/entry/instrument/detector/data"
                     type="string"></attribute>
        </dataset>
        <group name="NDAttributes">
          <attribute name="NX_class" source="constant" value="NXcollection" type="string"></attribute>
          <dataset name="ColorMode" source="ndattribute" ndattribute="ColorMode">
          </dataset>
        </group>          <!-- end group NDAttribute -->
      </group>            <!-- end group detector -->
      <group name="NDAttributes" ndattr_default="true">
        <attribute name="NX_class" source="constant" value="NXcollection" type="string"></attribute>
      </group>            <!-- end group NDAttribute (default) -->
      <group name="performance">
        <dataset name="timestamp" source="ndattribute"></dataset>
      </group>            <!-- end group performance -->
    </group>              <!-- end group instrument -->
    <group name="data">
      <attribute name="NX_class" source="constant" value="NXdata" type="string"></attribute>
      <hardlink name="data" target="/entry/instrument/detector/data"></hardlink>
      <!-- The "target" attribute in /entry/instrument/detector/data is used to
           tell Nexus utilities that this is a hardlink -->
    </group>              <!-- end group data -->
  </group>                <!-- end group entry -->
</xml>
```

Multiple Threads per Plugin

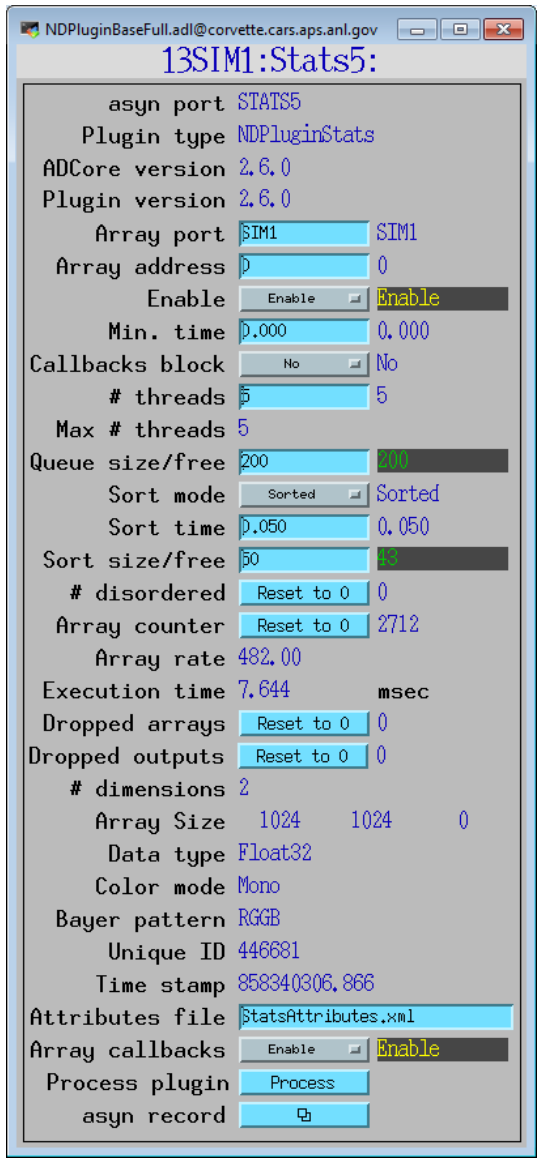
- Support for multiple threads running the processCallbacks() function in a single plugin.
- Can improve the performance of the plugin by a large factor.
- Linear scaling with up to 5 threads (the largest value tested) observed for most of the plugins that now support multiple threads.
- Maximum number of threads that can be used for the plugin is set in constructor and in IOC startup script.
- Actual number of threads to use controlled via an EPICS PV at run time, up to the maximum value.
- Optional sorting of NDArrays by uniqueId to attempt to output them in the correct order.
  - Several new parameters to control this option
- Plugins needed minor modifications to be thread-safe for multiple threads running in a single plugin object.
- Most compute-intensive plugins now support multiple threads.

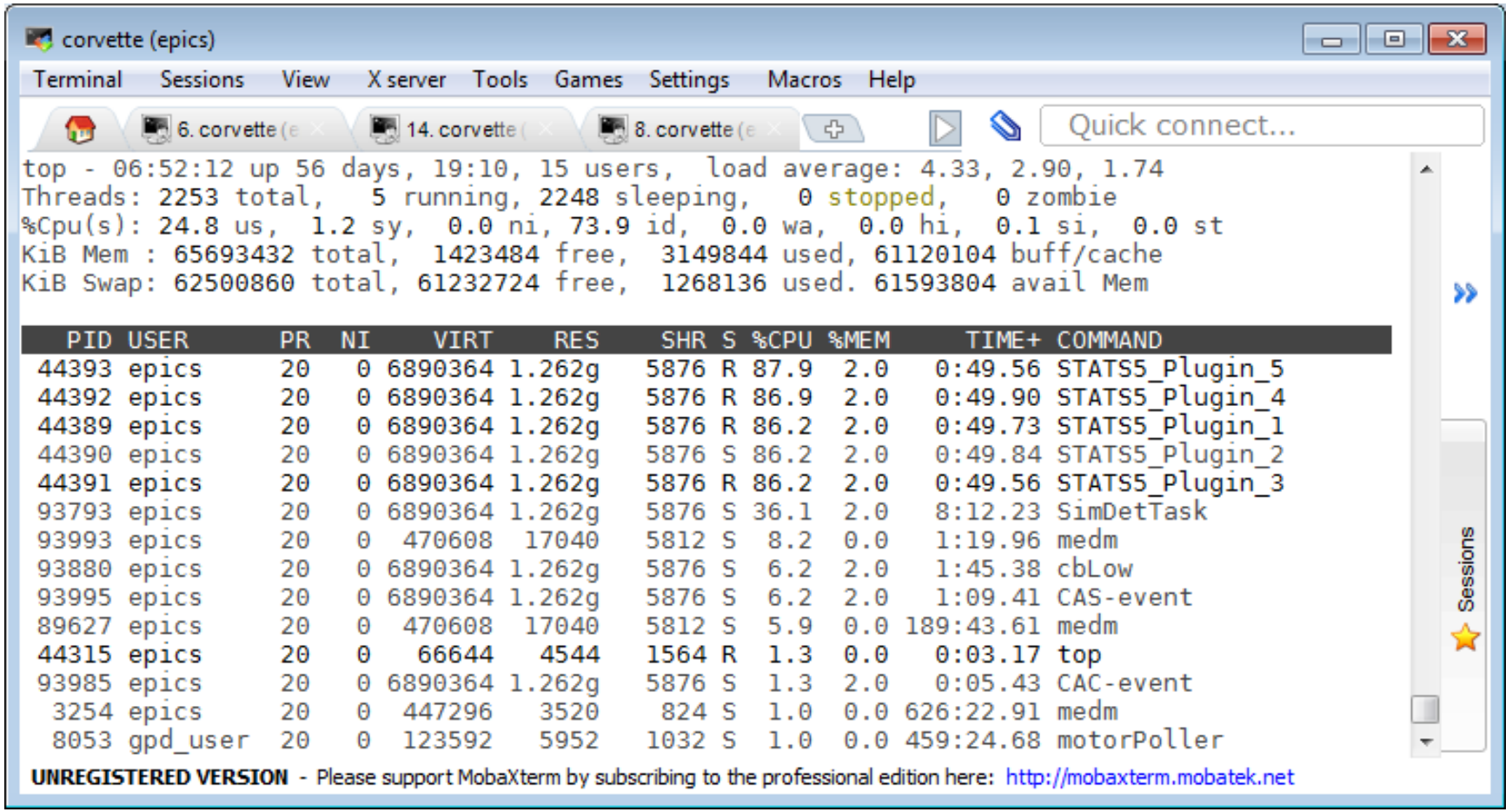
Multiple Threads per Plugin
1 Thread





Multiple Threads per Plugin
5 Threads



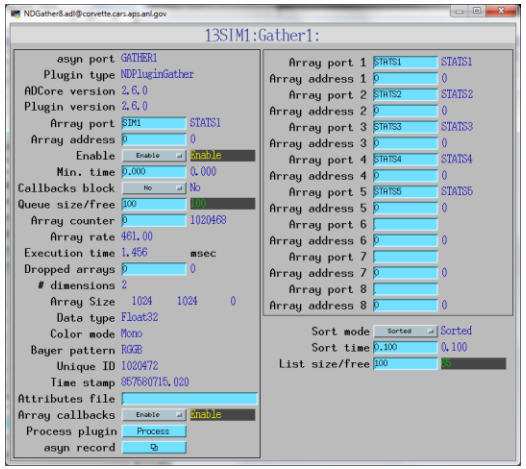


NDPluginScatter

- Used to distribute (scatter) the processing of NDArrays to multiple downstream plugins
  - Allows multiple instances of a plugin to process NDArrays in parallel, utilizing multiple cores to increase throughput.
  - Utilizes modified round-robin for choosing next output plugin
- More complex than multiple threads in a single plugin, but allows the plugins running in parallel to have different configurations or even be different plugins

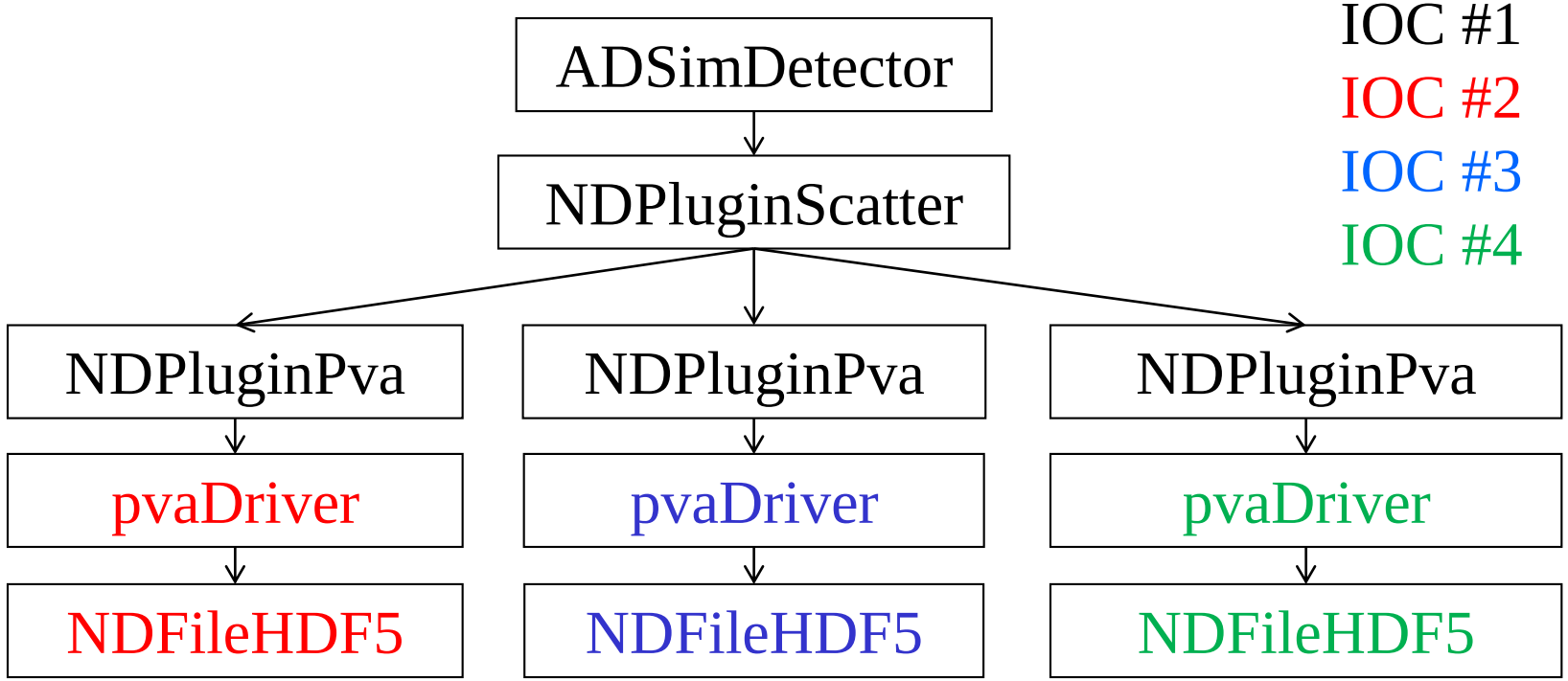
NDPluginGather

- Merges NDArrays from multiple upstream plugins into a single output stream.
- Designed to work with NDPluginScatter
- Optional sorting by uniqueId



Distributed Processing with NDPluginScatter + EPICS V4

Distribute HDF5 file writing to multiple IOCs (4096 x 3078 8-bit)



# IOCs	Files/sec	GB/sec
1	101.0	1.19
2	195.2	2.29
3	217.5	2.55

Viewers

- areaDetector allows generic viewers to be written that receive images as EPICS waveform records over Channel Access
- Current viewers include:
  - ImageJ plugin EPICS_AD_Display. ImageJ is a very popular image analysis program, written in Java, derived from NIH Image.
  - EPICS_NTNDA_Viewer. Same as above but uses pvAccess rather than Channel Access.
  - ffmpegServer allows image display in any Web browser
  - ffmpegViewer high-performance Qt-based viewer for MJPEG stream

NDPluginPva Advantages

- NTNDArray data transmitted "atomically" over the network
  - Channel Access requires separate PVs for the image data and the metadata (image dimensions, color mode, etc.)
- With Channel Access data type of waveform record is fixed at iocInit, cannot be changed at runtime.
  - If the user wants to view both 8-bit images, 16-bit images, and 64-bit double FFT images then waveform record needs to be 64-bit double, adding a factor of 8 network overhead when viewing 8-bit images.
  - pvAccess changes the data type of the NTNDArrays dynamically at run-time, removing this restriction.
- Channel Access requires setting EPICS_CA_MAX_ARRAY_BYTES
  - Source of considerable confusion and frustration for users.
  - pvAccess does not use EPICS_CA_MAX_ARRAY_BYTES and there is no restriction on the size of the NTNDArrays.

Other Drivers that use ADCore

- NDArrays are not limited to 2-D detectors
  - File, ROI, and statistics plugs are useful for other types of detectors
- Used for spectra arrays [NumMCAChannels, NumDetectors, NumPixels] for:
  - Xspress3 from Quantum Detectors
  - xMAP, Mercury and new FalconX from XIA
  - Dante from XGLab
- Used for time-series data [NumTimePoints, NumInputs] for the quadEM quad electrometer software
  - AH401, AH501, TetrAMM from CaenEls
  - Two types of electrometers from BNL Instrumentation group (Peter Siddons)

Conclusions

- Architecture works well, easily extended to new detector drivers, new plugins and new clients
- Base classes, asynPortDriver, asynNDArrayDriver, asynPluginDriver actually are generic, nothing “areaDetector” specific about them.
- They can be used to implement any N-dimension detector, e.g. the XIA xMAP (16 detectors x 2048 channels x 512 points in a scan line)
- Documentation
  - https://areaDetector.github.io
- Can get code from github
  - https://github.com/areaDetector