ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

AREADETECTOR: EPICS SOFTWARE FOR 2-D DETECTORS

M. L. Rivers, University of Chicago, Chicago, USA

Abstract

areaDetector is an EPICS framework to support 2-D detectors. It is modular C++ code that greatly simplifies the task of writing drivers for new detectors. It supports plugins, which receive detector data from the driver and process it in some way. For example, plugins perform statistics calculations, image processing, Region-Of-Interest extraction, file saving, color mode conversion, and export to EPICS Channel Access or pvAccess for image display in clients like ImageJ. Plugins can each run in their own threads, permitting parallel processing on multi-core machines. Drivers have been written for many of the detectors commonly used at synchrotron and neutron sources, including CMOS and CCD imaging and x-ray detectors, pixel array detectors and flat panel detectors.

INTRODUCTION

Most x-ray experiments whether at a synchrotron, FEL, or home laboratory use 2-D detectors. These include x-ray detectors such as direct-detect pixel array detectors and scintillation based detectors with CCD, CMOS, and amorphous silicon sensors. It also includes visible light CCD and CMOS cameras for imaging and optical spectroscopy. These detectors need to be integrated with the control system used for other components of the experiment. Many beamlines around the world use the EPICS control system [1], which is a collaborative open-source control system toolkit. areaDetector is an EPICS framework for controlling detectors [2]. An early (2010) version of areaDetector was described in [3]. This paper will briefly summarize the framework architecture, and then describe in greater detail the features that have been added since 2010.

The goals of the areaDetector module are to:

- Minimize the amount of code that needs to be written to implement a new detector.
- Provide a standard interface defining the functions and parameters that a detector driver should support.
- Provide a set of base EPICS records that will be present for every detector using this module. This allows the use of generic EPICS clients for displaying images and controlling cameras and detectors.
- Allow easy extensibility to take advantage of detector-specific features beyond the standard parameters.
- Have high-performance.
- Provide a mechanism for device-independent real-time data analysis such as regions-of-interest, statistics, image processing.
- Save files in industry standard formats.
- Provide drivers for commonly used detectors in x-ray and imaging applications.

ARCHITECTURE

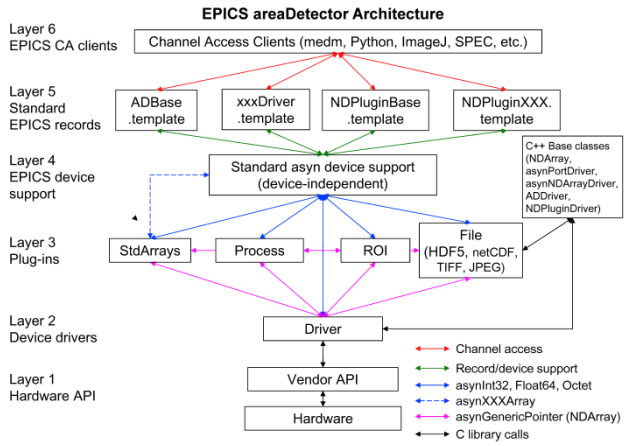

Figure1: areaDetector Architecture.

The architecture of the areaDetector module is shown in Figure 1. From the bottom to the top this architecture consists of the following six layers:

1. This is the layer that allows user written code to communicate with the hardware. It is usually provided by the detector vendor. It may consist of a library, a socket protocol definition, or other type of API.
2. This is the driver that is written for the areaDetector application to control a particular detector. It is written in C++ and inherits from the ADDriver class. It uses the standard asyn interfaces for control and status information. Each time it receives a new data array it passes it as an NDArray object to all Layer 3 clients that have registered for callbacks. This is the only code that needs to be written to implement a new detector.
3. Code running at this level is called a plugin. This code registers with a driver (or another plugin) for a callback whenever there is a new data array.
4. This is standard asyn device support that comes with the EPICS asyn module [4, 5].
5. These are standard EPICS records and EPICS database (template) files that define records to communicate with drivers at Layer 2 and plugins at Layer 3.
6. These are EPICS channel access clients that communicate with the records at Layer 5. areaDetector includes ImageJ client applications that can display images using EPICS Channel Access or pvAccess. It also includes detector and plugin screens for the EPICS display managers medm, edm, CSS BOY, and caQtDM.

IMPLEMENTATION

The areaDetector module depends heavily on asyn. It is the software that is used for inter-thread communication, using the standard asyn interfaces (e.g. asynInt32, asynOctet, etc.), and callbacks. areaDetector is implemented using C++ classes. The base classes, from which

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

drivers and plugins are derived, take care of many of the details of asyn and other common code. The most important classes in the areaDetector software are described here.

NDArray

The NDArray (N-Dimensional array) class is used for passing detector data from drivers to plugins, and from one plugin to another. An NDArray is a general purpose class for handling array data. An NDArray object is self-describing, meaning it contains enough information to describe the data itself. It can optionally contain "attributes" (class NDAttribute) which contain meta-data describing how the data was collected, etc. Each dimension of the array is described by an NDDimension structure, which includes the size, offset from detector origin, binning and a flag indicating if the data are reversed in that dimension from the original detector orientation. NDArrays supported datatypes are signed and unsigned 8, 16, and 32 bit integers, and 32 and 64 bit floats. The NDArrayPool class manages a free list (pool) of NDArray objects. Drivers allocate NDArray objects from the pool, and pass pointers to these objects to plugins. Plugins increase the reference count on the object when they place the object on their queue, and decrease the reference count when they are done processing the array. When the reference count reaches 0 again the NDArray object is placed back on the free list. This mechanism minimizes the copying of array data in plugins.

NDAttribute

NDAttribute is a class for linking metadata to an NDArray. An NDAttribute has a name, description, data type, value, source type and source information. Attributes are identified by their names. There are methods to set and get the information for an attribute. The PVAttribute, paramAttribute, and functAttribute classes are derived from NDAttribute. PVAttribute obtains its value by monitor callbacks from an EPICS Process Variable (PV), and is thus used to associate current the value of any EPICS PV with an NDArray. For example, each image could be tagged with the positions of set of motors and/or the energy of a monochromator. paramAttribute obtains its value from the current value of a driver or plugin parameter. The paramAttribute class is typically used when it is important to have the current value of the parameter and the value of a corresponding PVAttribute might not be current because the EPICS PV has not yet updated. functAttribute gets its value from a user-defined function, and so may be used to obtain its value from any data source.

Class Hierarchy

The class hierarchy for drivers and plugins is shown in Figure 2. To save space this figure does not show all of the drivers or plugins.

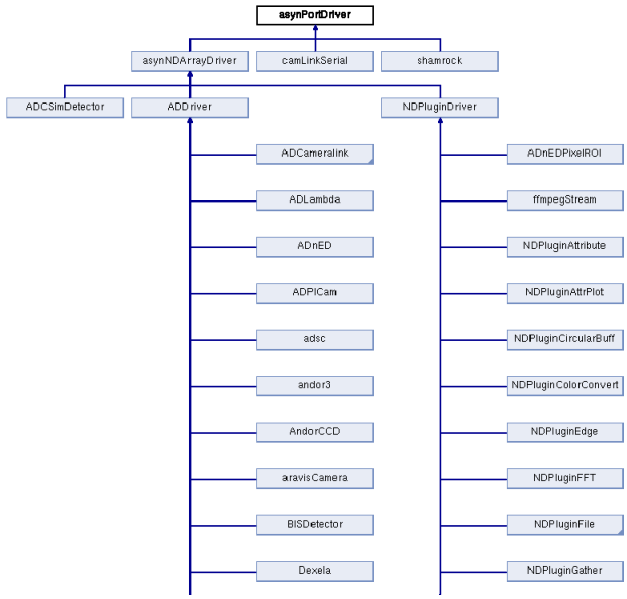

Figure 2: areaDetector class hierarchy.

asynPortDriver

The lowest level class is asynPortDriver, which is part of asyn. It hides the asyn interfaces to EPICS device support, and implements a parameter library and callbacks to device support when parameter values change. This class is used for many EPICS drivers (e.g. A/D, D/A, digital I/O), not just those in areaDetector.

asynNDArrayDriver

asynNDArrayDriver is the class from which both plugins and areaDetector drivers are indirectly derived. It is a general purpose N-dimensional array driver class, and is not limited to 2-D detectors. For example it is used for spectroscopy detector drivers that produce 1-D arrays and for 4-channel electrometer drivers that produce time-series data. It defines the parameters that all NDArray drivers and plugins should implement if possible. These parameters have an associated asyn interface (asynInt32, asynFloat64, etc.), and access (read-only or read-write).

ADDriver

Class ADDriver inherits from asynNDArrayDriver and is the class from which areaDetector drivers are directly derived. It provides parameters and methods that are specific to 2-D area detectors. These are parameters that all areaDetector drivers should implement if possible, and include acquisition time, stop/start acquiring, detector region to read, etc. Although ADDriver is used for 2-D detectors, the NDArrays it generates may be 3-D, because it supports color cameras, with pixel, row or plane interleave formats.

NDPluginDriver

The NDPluginDriver class also inherits from asynNDArrayDriver, and is the class from which all of the plugins are directly derived. It defines the parameters that control the plugin behavior, such as which driver or plugin to get its data from, whether the plugin shown be

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

blocking or non-blocking, the number of threads to use, etc.

DETECTOR DRIVERS

Table 1: Detectors Currently Supported

Repository	Detector/camera type
ADSimDetector	Simulation driver for 2-D images, supports all data types and color modes.
ADCSimDetector	Simulation driver for time-series data from an N-channel ADC.
ADADSC	ADSC CCD
ADAndor	Andor CCD SDK 2.x
ADAndor3	Andor sCMOS SDK 3.x
ADBruker	Bruker CCD with BIS
ADDexela	Perkin Elmer Dexela CMOS flat panel
ADFastCCD	LBNL/ANL/BNL fast CCD
ADFireWireWin	Firewire IIDC/DCAM cameras on Windows
ADLambda	XSpectrum Lamba pixel array
ADLightField	Princeton Instruments CCDs and spectrographs using LightField
ADmar345	mar345 online image plate
ADmarCCD	Rayonix CCD
ADMerlin	Quantum Detectors Merlin Medipix3 pixel array
ADMythen	Dectris Mythen strip pixel array
ADPCO	PCO Dimax and Edge CMOS
ADPerkinElmer	Perkin Elmer amorphous silicon flat panel
ADPhotonII	Bruker PhotonII scintillator/CMOS
ADPICam	Princeton Instruments CCDs using PI CAM library
ADPilatus	Dectris Pilatus pixel array
ADPixirad	Pixirad CdTe pixel array
ADPointGrey	FLIR/Point Grey CCD and CMOS
ADProsilica	AVT/Prosilica CCD and CMOS
ADPSL	Photonic Sciences cameras using PSLViewer socket server
ADPvCam	Photometrics/Princeton Instruments cameras using PvCam library
ADQImaging	QImaging Technology CMOS
ADRoper	Photometrics/Princeton Instruments cameras using WinView
ADURL	Reads images from a URL, such as Web cameras and disk files
aravisGigE	Genicam GigE cameras on Linux
firewireDCAM	Firewire IIDC/DCAM cameras on Linux
NDDriver-StdArrays	Receives NDArrays from EPICS Channel Access clients
pvaDriver	Receives NDArrays from EPICS pvAccess servers

areaDetector contains drivers for many for many commonly used 2-D detectors at synchrotron beamlines. These drivers inherit from ADDriver. Drivers implement as many of the standard areaDetector parameters as possible, and also implement additional parameters that are specific to that detector. If the vendor API supports saving files, then this is implemented in the driver. areaDetector can also save files using plugins described below. For some drivers the only way to get the detector data into areaDetector is to write a disk file then read that back into memory (e.g. Pilatus, marCCD, mar345). For other drivers the data can be obtained directly. Drivers optionally pass the detector data back to any registered plugins as NDArrays as each image is collected. Table 1 lists the detectors that are currently part of the areaDetector project on GitHub. Each detector is in its own repository in the areaDetector project.

PLUGINS

A powerful feature of the areaDetector module is the concept of plugins. Plugins are used to process array data in real time. A plugin registers for callbacks from a driver or another plugin. Each time the driver or other plugin receives a new NDArray it passes a pointer to that NDArray to all plugins that have registered with it for callbacks. Plugins derive from the NDPluginDriver base class, which handles the tasks that are common to all plugins.

Plugin Configuration Parameters

Figure 3 shows the detailed configuration screen that is available for NDPluginDriver. Most plugins have additional configuration parameters beyond those shown in this screen. The following describes the most important plugin configuration parameters shown in this screen, from top to bottom.

NDArrayPort, NDArrayAddress: These control the asyn port name and address of the driver or plugin that will do callbacks to this plugin, i.e. the source of the NDArray data.

EnableCallbacks: Enable or disable this plugin. When enabled it registers for callbacks from the source, when disabled it unregisters.

MinCallbackTime: The minimum time between processing callbacks. Any callbacks occurring before this minimum time has elapsed will be ignored. 0 means no minimum time, i.e. process all callbacks.

BlockingCallbacks: Plugins can execute either in a blocking mode or a non-blocking mode. In the blocking mode the callback is executed by the driver callback thread. In this mode the callback is guaranteed to execute for each NDArray callback. However, it can slow down the driver, and does not utilize the multi-core capability of modern CPUs. In the non-blocking mode the driver callback simply places the NDArray data in a queue that is part of the plugin. The plugin then executes the callback code in its own thread. It removes NDArray data from the queue, processes it, and releases the data back to

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

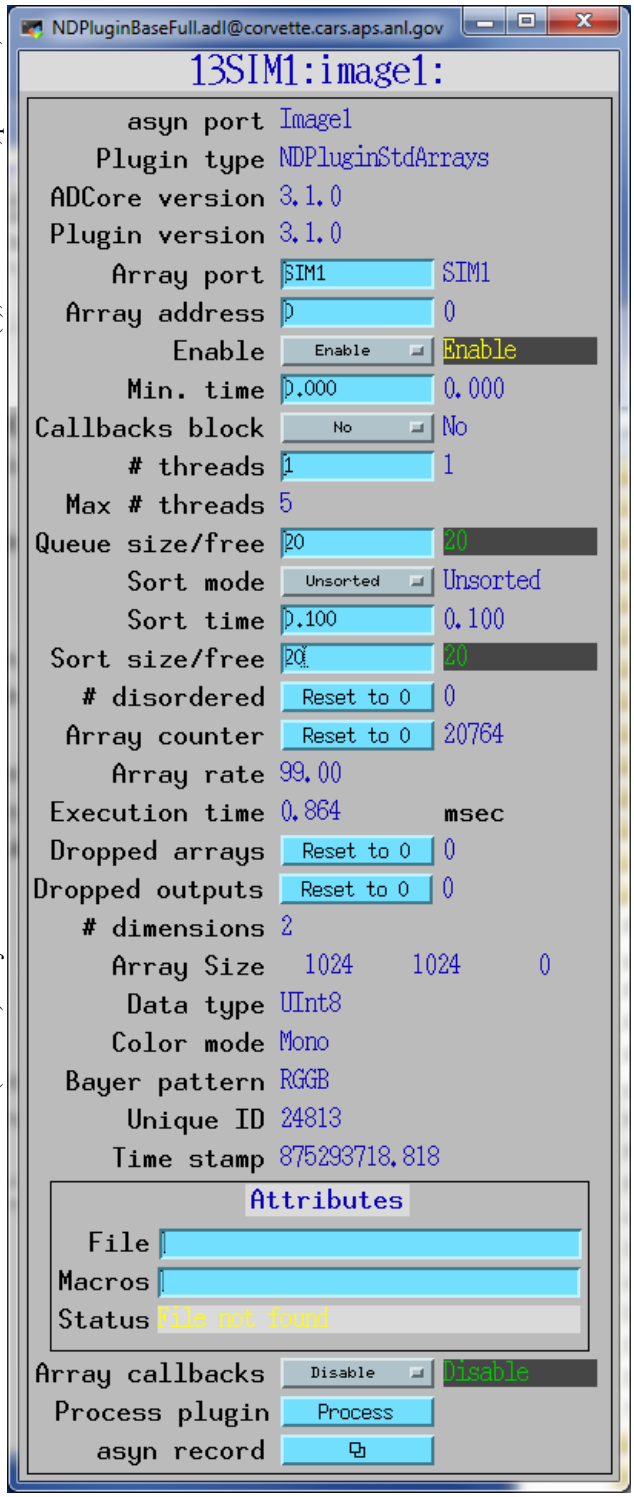

Figure 3: Plugin Detailed Configuration Screen.

the NDArrayPool when it is done. It is possible to drop NDArray data if the queue is full when a callback occurs, i.e. some callback data will not be processed. The non-blocking mode can utilize the multi-core capabilities of modern CPUs because each plugin is executing in its own thread. The operation of the queue and the NDArrayPool class means that data never needs to be copied; each plugin has a pointer to the data which will continue to be valid until the last plugin is done with it.

MaxThreads: The maximum number of threads that this plugin is allowed to use when Blocking-Callbacks=No. This is defined when the plugin is created, and cannot be changed at run-time. Some plugins are not thread-safe for multiple threads running in the same plugin object, and these force MaxThreads=1.

NumThreads: The number of threads to use for this plugin. The value must be between 1 and MaxThreads.

QueueSize: The total queue size for callbacks when BlockingCallbacks=No. This can be changed at run time to increase or decrease the size of the queue and thus the buffering in this plugin. When the queue size is changed the plugin temporarily stops the callbacks from the input driver and waits for all NDArrays currently in the queue to process.

QueueFree: The number of free queue elements. This record goes into minor alarm when the queue is 75% full and major alarm when the queue is 100% full.

SortMode, SortTime, SortSize, SortFree: When the plugin is using multiple threads it is likely that the NDArray output will be slightly out of order, i.e. NDArray.uniqueId fields will not be monotonically increasing. This is because the threads are running asynchronously and at slightly different speeds. As a consequence a plugin downstream of this plugin will receive NDArrays in the wrong order. Plugins have an option to sort the NDArrays by uniqueId to attempt to output them in the correct order. This sorting option is enabled by setting SortMode=Sorted. The arrays to be output are stored in a sorted list of size SortSize, and a thread waits for up to SortTime for the next expected NDArray to arrive before outputting the next element from the list.

DisorderedArrays: The number of NDArrays that have been output in the wrong order.

DroppedArrays: The number of NDArrays that have been dropped because an NDArray callback occured when BlockingCallbacks=No and the plugin driver queue is full, so the callback cannot be processed.

DroppedOutputArrays: The number of NDArrays that have been dropped on output because an NDArray callback occurs when SortMode=Sorted and the sorted list is full (SortFree=0), so the NDArray cannot be added to the list.

NDArray Information: Below DroppedOutputs in Figure 3 are a number of parameters that provide information on the most recent NDArray received (NDimensions, Dimensions, DataType, ColorMode, BayerPattern, UniqueID, and TimeStamp. The Attributes fields allow specifying an XML file that will define additional NDAttributes that this plugin will add to the NDArray before calling any downstream plugins.

ArrayCallbacks: Enable or disable this plugin from doing callbacks to any downstream plugins that have registered for callbacks.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

ProcessPlugin: NDPluginDriver maintains a pointer to the last NDArray that the plugin received. ProcessPlugin causes the plugin to run again using this same NDArray. This can be used to change the plugin parameters and observe the effects on downstream plugins and image viewers without requiring the underlying detector to collect another NDArray.

Available Plugins

Table 2 lists the plugins that are currently part of the areaDetector project. Plugins are not difficult to write, and many sites have written their own for specific applications. To save space in the table the NDPlugin prefix for each plugin type is omitted.

Table 2: Available Plugins

Plugin	Description
AttrPlot	Copies NDAttribute values to waveform records for plotting.
Attribute	Extracts NDArray attributes and publishes their values over Channel Access.
CircularBuff	Buffers NDArrays in a circular buffer until triggered.
ColorConvert	Converts the ColorMode of the NDArray.
FFT	Computes 1-D or 2-D FFTs. Exports NDArrays containing the absolute value of the FFT, and waveforms of the real and imaginary components, and time and frequency axes. Optionally does recursive averaging to increase the signal to noise.
File	Base class for file writing plugins.
Gather	Gathers NDArrays from multiple upstream plugins and merges them into a single stream. Normally used together with NDPluginScatter to allow multiple instances of a plugin to process NDArrays in parallel.
Overlay	Adds graphics overlays to an NDArray image. Used to highlight ROIs or beam location, implement cursors, add text annotation, etc. Provides control of the location, size, line width, color, and drawing mode of each overlay element.
Pos	Attaches positional information to NDArrays as NDAttributes. Designed to store NDArrays in an arbitrary pattern within an HDF5 multi-dimensional dataset.
Process	Performs arithmetic processing on NDArrays. Operations include background subtraction, flat-field normalization, scaling, offset, clipping, recursive digital filter, and data type conversion.
Pva	Converts NDArrays into the EPICS V4 type NTNDArray. An embedded pvAccess server is created to serve the NTNDArray structure on the network.
ROI	Selects a Region Of Interest from an NDArray. It optionally does binning, orientation reversal, scaling, data type conversion, and collapsing (removing) dimensions whose value is 1.
ROIStat	Provides multiple ROIs and simple statistics in a single plugin.
Scatter	Used to distribute (scatter) the processing of NDArrays to multiple downstream plugins, allowing multiple instances of a plugin to process NDArrays in parallel. Normal plugins pass each NDArray to *all* downstream plugins, while this plugin passes each NDArray to *only one* downstream plugin. Normally used with NDPluginGather.
Stats	Computes statistics on the NDArray including min, max, mean, sigma, total, net, centroid, sigma, skewness, kurtosis, profile arrays in X and Y directions, histogram arrays, and entropy.
StdArrays	Converts NDArrays into 1-D arrays for export as waveform records and ancillary PVs with EPICS Channel Access..
TimeSeries	Appends 1-D or 2-D data into time-series arrays. Operates in either Fixed Length or Circular Buffer modes.
Transform	Geometrically transforms NDArrays with the 8 possibilities that involve rotations by multiples of 90 degrees and mirror reflections about the central vertical line of the array.
ffmpegServer	Uses the ffmpeg library to provide compression either into an mjpg stream over http, or to disk in any file format that ffmpeg supports.

File Writing Plugins

NDPluginFile is the base class from which the actual file writing plugins are derived. It supports 3 file saving modes.

1. Single mode. In this mode each NDArray callback results in a separate disk file.
2. Capture mode. In this mode a memory buffer is allocated before saving begins. Callback arrays are placed into this buffer, and when capture stops the file is written to disk. This mode limits the number of frames that can be saved, because they all must fit in a memory buffer. It is the fastest mode, with the least probability of dropping arrays, because no disk I/O is required while capture is in progress.
3. Stream mode. In this mode the data are written to a single disk file for those file formats that support multiple arrays per file (netCDF, NeXus and HDF5). Each frame is appended to the file without closing it. It is intermediate in speed between Single mode and Capture mode, but unlike Capture mode it is not limited

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

by available memory in the number of arrays that can be saved. For file formats that do not support multiple arrays per file (e.g. JPEG, TIFF) this mode is really the same as Single mode, except that one can specify a total number of files to save before stopping.

There are currently 6 file writing plugins.

NDFileJPEG: Writes files in JPEG format. It supports 8- bit images with any color mode and with user-defined quality factor. It cannot store NDAttributes in the file, and can only store 1 image per file.

NDFileTIFF: Writes files in TIFF format. It supports NDArrays of any of the 8 NDArray data types and all color modes. It writes all NDAttributes attached to the NDArray as TIFF ASCII file tags, up to a maximum of 490 tags. These tags start with number 65000. It can only store 1 NDArray per file, and does not support compression.

NDFileNetCDF: Writes files in netCDF Classic format. It supports NDArrays of any of the 8 NDArray data types. It writes all NDAttributes attached to the NDArray as additional variables in the netCDF file, and can store multiple NDArrays per file.

NDFileHDF5: Writes files in HDF5format. It supports NDArrays of any of the 8 NDArray data types. It writes all NDAttributes attached to the NDArray as additional variables in the HDF5 file, and can store multiple NDArrays per file. The layout of the data in the HDF5 file can be defined using an XML file. It supports control of chunking, and multiple compression algorithms. It supports the Single Writer Multiple Reader (SWMR) feature that was added in HDF5 1-10.

NDFileNexus: Writes files in HDF5format, but it uses the NeXus API rather than the native HDF5 API. It supports NDArrays of any of the 8 NDArray data types.

NDFileMagick: Writes file using the GraphicsMagick library. This supports a large number of formats including JPEG, BMP, EPS, FITS, GIF, HTML, JBIG, JB2, PDF, PNG, and TIFF (with compression).

Note that many vendors provide file writing in their API or applications, and this provides another mechanism for writing images to disk.

DISTRIBUTED PROCESSING

An areaDetector IOC runs as a single process. It can effectively use many cores because each plugin can run in one or more threads. It is also possible to scale areaDetector to run in multiple processes and on multiple machines. This is done by using the EPICS V4 pvAccess services [6]. The IOC that is running the detector driver runs the NDPluginPva plugin. This plugin accepts NDArrays from the driver, converts them to the EPICS V4 NTNDArray normative type, and serves them on the network with a pvAccess server. To distribute the load additional IOCs can be run either on the same machine or on other machines on the network. These additional IOCs run the pvaDriver driver code. This driver is a pvAccess client, and subscribes for callbacks from the NDPluginPva plugin. It receives the NTNDArrays and converts them back to NDArrays in this IOC, where plugins can perform additional processing and file I/O. This architecture provides easy scaling when the required processing exceeds that which can be done in a single process on a single machine.

PERFORMANCE

The following tests were done to illustrate what can be gained from using the EPICS V4 mechanism described above to increase performance. In this case the goal was to be able to store 4096x3078 pixel 8-bit images at 190 frames/s as HDF5 files, the maximum frame rate of a new Adimec camera. The tests were done on a single Linux machine with 20 cores and 128 GB of RAM. HDF5 files were written to a 64GB tmpfs RAM disk. The ADSimdetector driver was generating the images, NDPluginScatter sent the images to 3 NDPluginPva plugins. These sent the images to 3 other IOCs, each running the pvaDriver, and the NDFileHDF5 plugin.

Table 3: Performance with pvAccess for Parallel I/O

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

It can be seen that a single IOC was unable to meet the 190 frame/s performance spec, 2 IOCs just met it, and 3 IOCs exceeded it by 10%. With 3 IOCs the performance appears to be limited by memory bandwidth.

VIEWERS

One of the advantages of areaDetector is that it enables the use of generic image display clients that obtain their data via EPICS Channel Access or pvAccess and work with any detector. There are currently three such generic clients provided with the areaDetector distribution. The first two are plugins for the popular ImageJ Java-based image processing program. EPICS_AD_Viewer.java uses EPICS Channel Access, while EPICS_NTNDA-_Viewer.java uses EPICS V4 pvAccess (Figure 4). The third is an IDL-based viewer which can be run without an IDL license under the IDL Virtual Machine. Because ImageJ is free and more widely available and used than IDL, future enhancements are more likely to be done on the ImageJ plugins rather than the IDL viewer. These viewers are contained in the areaDetector distribution in the ADViewers repository.

The V4 EPICS_NTNDA_Viewer has a number of significant advantages compared to the EPICS_AD_Viewer:

- The NTNDArray data is transmitted "atomically" over the network, rather than using separate PVs for the image data and the metadata (image dimensions, color mode, etc.)

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THDPL03

- When using Channel Access the data type of the waveform record is fixed at iocInit, and cannot be changed at runtime. This means, for example, that if the user might want to view both 8-bit images, 16-bit images, and 64-bit double FFT images then the waveform record would need to be 64-bit double, which adds 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, which is a 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.
- The performance using pvAccess is significantly better than using Channel Access. NDPluginPva is 5-10 times faster than NDPluginStdArrays, and ImageJ can display 1.5-2 times more images/s with pvAccess than with Channel Access.

In addition to the ImageJ viewer plugins, there is an ImageJ plugin to graphically define the detector/camera readout region, ROIs, and overlays (EPICS_AD_Controller.java). Another ImageJ plugin provided with areaDetector, GaussianProfiler.java, does real-time line profiles with Gaussian peak fitting (Figure 5).

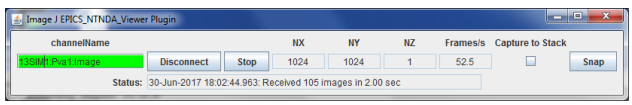

Figure 4: EPICS V4 ImageJ Plugin Control.

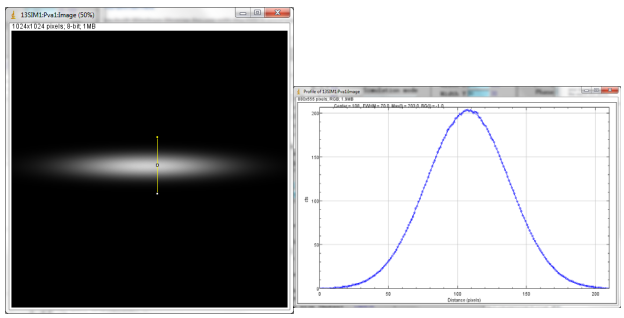

Figure 5: EPICS V4 ImageJ Display and Live Gaussian Profile Fit and Plot.

CODE ORGANIZATION

areaDetector is hosted in the areaDetector project on GitHub. The project currently contains 44 repositories. Most of these contain the drivers for the specific detectors. A few of the most important repositories are described here.

ADCore: Contains the source code, EPICS databases, and OPI screens for the base classes and all of the standard plugins.

ADSupport: Contains the source code for all of the third party libraries that areaDetector can use. These have been modified to use the EPICS build system, so they can be built on most platforms that EPICS supports, including Linux, Windows, Mac OS, and VxWorks. These libraries include JPEG, TIFF, GraphicsMagick, HDF5, blosc, Nexus, netCDF, szip, xml2, and zlib. For each library it is possible to build areaDetector without or without it, and to build with the library in ADSupport or with a system installation of the library.

ADSimDetector: Contains the source code for a simulation detector. Other real drivers are also each in their own repositories.

ACKNOWLEDGMENTS

areaDetector is an active collaboration, and many people have contributed to the software described here. These include Ulrik Pedersen, Tom Cobb, Alan Greer, Peter Heesterman, Giles Knap, Edmund Warrick, Hinko Kocevar, Matt Pearson, Bruno Martins, Stuart Wilkins, John Hammonds, Matthew Moore, Lewis Muir, Tim Mooney, Arthur Glowacki, Tim Madden, Chris Roehrig, Keith Brister, Russell Woods, Brian Tieman, Xiaoqiang Wang, Blaž Kranjc, Phillip Sorensen, Mike Dunning, and Marty Kraimer. Special thanks to Andrew Johnson who has patiently helped me with innumerable questions and problems.

This work was supported by the National Science Foundation-Earth Sciences (EAR-1634415) and Department of Energy-GeoSciences (DE-FG02-94ER14466)

REFERENCES

[1] Experimental Physics and Industrial Control System, http://www.aps.anl.gov/epics

[2] areaDetector: EPICS software for area detectors, http://cars.uchicago.edu/software/epics/areaDetector.html

[2] M. Rivers, "areaDetector: Software for 2-D Detectors in EPICS" in *AIP Conference Proceedings* 1234, 2010, pp. 51-54.

[4] M. R. Kraimer, M. Rivers, and E. Norum, "EPICS: Asynchronous Driver Support," in *Proc. 10th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2005)*, Geneva, Switzerland, Oct. 2005, paper PO2.074-5.

[5] asynDriver: Asynchronous Driver Support http://www.aps.anl.gov/epics/modules/soft/asyn

[6] EPICS V4 Developer's Guide, http://epics-pvdata. sourceforge.net/informative/developerGuide/developerGuide.htm