Flexible and Scalable Data-Acquisition Using the *artdaq* Toolkit

Eric Flumerfelt, Kurt Biery, John Freeman, Gennadiy Luhkanin, Adam Lyon, Ron Rechenmacher, Ryan Rivera, Lorenzo Uplegger, Margaret Votava

IEEE Real-Time 2018 Conference

14 June 2018

Introduction

The *artdaq* Toolkit is a data-acquisition (DAQ) framework for use in high-energy physics experiments. Developed at Fermilab, it is the primary DAQ system for Fermilab's current and next-generation experiments.

- *artdaq* is a single framework used by experiments with radically different DAQ requirements
  - In addition to those shown below, ICARUS, Fermilab Test Beam Facility (See poster!), SENSEI and other CCD experiments, …
- *otsdaq* product is an *artdaq*-based complete DAQ solution

Fermilab Scientific Computing Common Tools

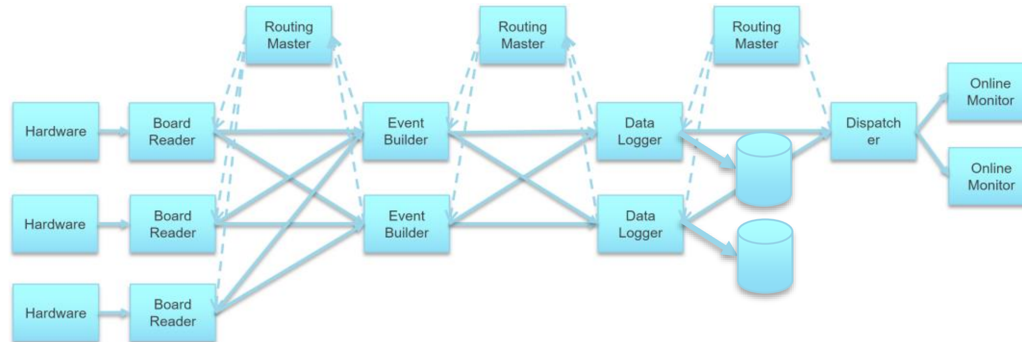
The Scientific Computing Division at Fermilab has adopted a strategy of providing centrally-developed "common tools" to experiments.

- *art* (offline) analysis framework was a very successful implementation of this strategy
  - In-use by all Fermilab experiments for offline analysis
  - Used by *artdaq* for software filtering and high-level triggering
- Frameworks are built on open-source software including Boost, ROOT, and Geant4
  - Run on standard x86_64 Linux machines

Design of *artdaq* – Applications

An artdaq system is comprised of several applications, each forming a "layer" of the system:

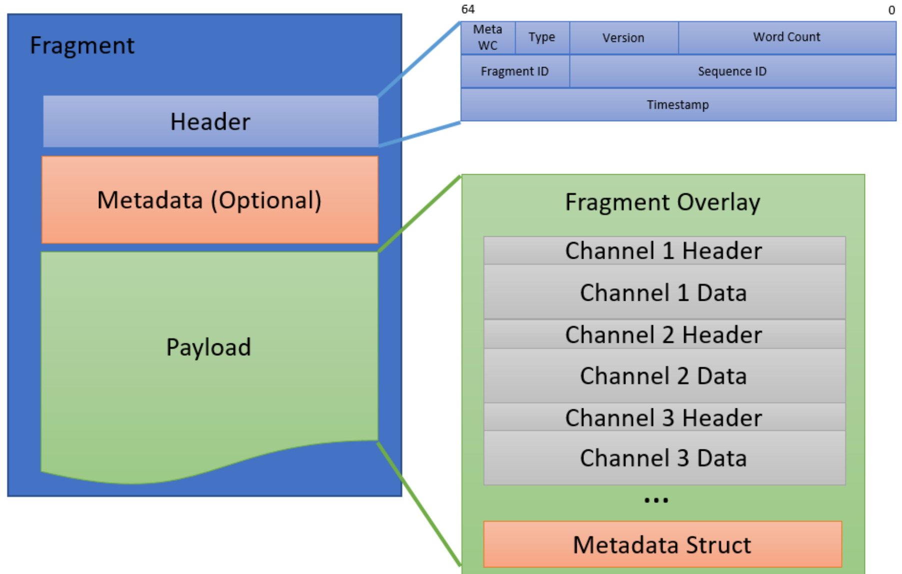
- BoardReaders – Read data from front-end hardware
- EventBuilders – Combine data from BoardReaders into events and perform filtering
- DataLoggers (Optional) – Write data to disk
- Dispatchers (Optional) – Provide data to external online monitoring applications
- RoutingMasters (Optional) – Orchestrate data flow between other artdaq layers



Fermilab

Design of *artdaq* - Fragment

The Fragment class is the basis for *artdaq* data transport. It consists of a header, optional metadata, and a variable-sized payload. The header contains all of the information necessary for transporting the Fragment through the DAQ system.

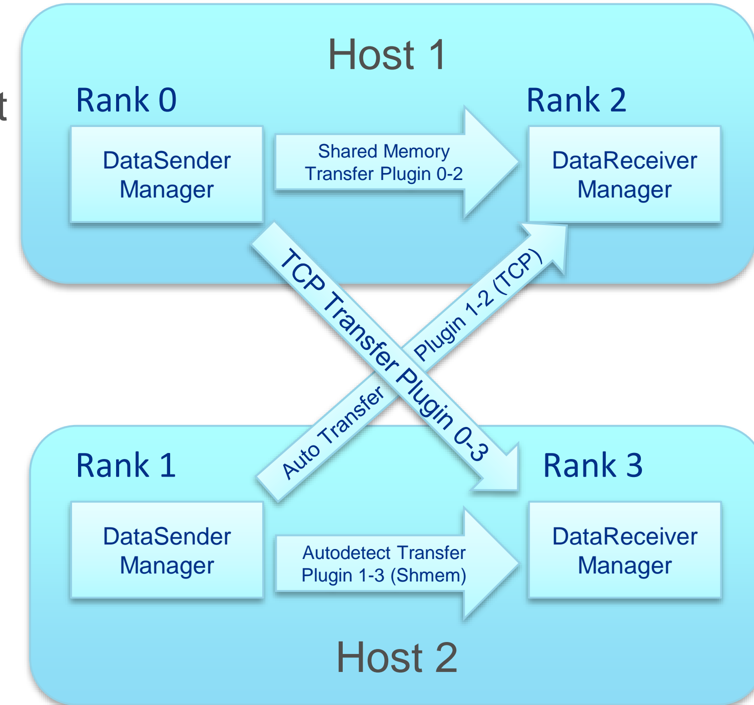


Fermilab

Design of *artdaq* - Transfers

Transfer plugins are used to transport Fragments between *artdaq* layers. *artdaq* provides Shared Memory, TCPSocket and Autodetect transfers

- Each Transfer is a point-to-point link, between two “ranks” in the *artdaq* system
- DataSenderManager gets information from the applicable RoutingMaster to determine the destination for a given Fragment’s Sequence ID
  - If RoutingMasters are not used, Fragments are sent round-robin



Fermilab

Design of *artdaq* - Requests

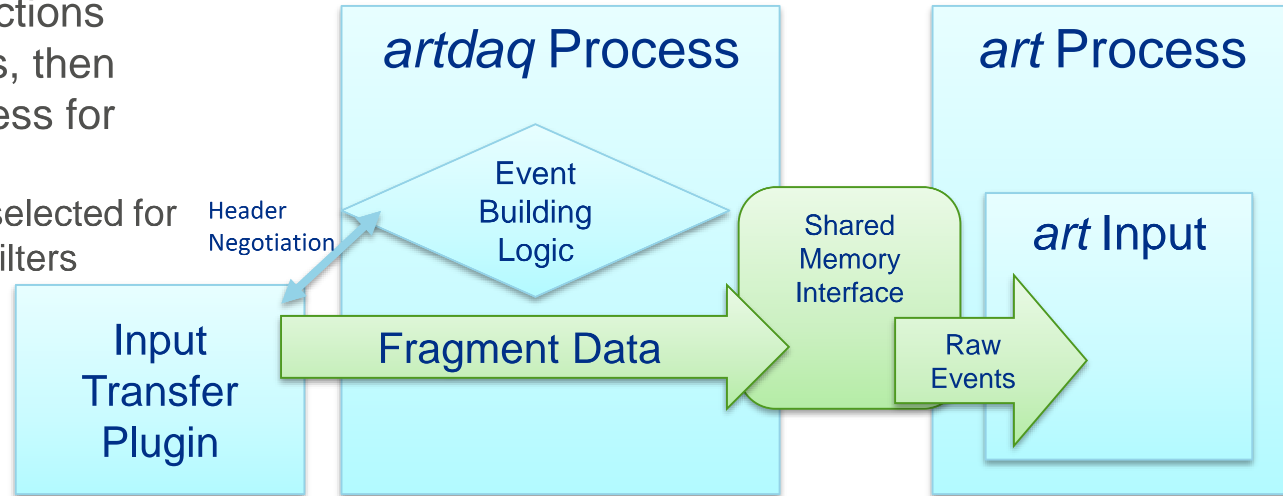
Artdaq supports “Data Requests” to synchronize readouts which may occur at different rates or to propagate hardware triggers through the system.

- Single and Buffer request processing modes are used for rate synchronization
- Window request mode allows for integration of data streams and continuous readout
  - Request uses timestamp field from the first Fragment to reach the EventBuilder
  - Fragment Generator in BoardReader is configured with offset and duration for request window
  - Fragment Generator replies with all data in its buffer matching request

Design of *artdaq* – *art* Process

EventBuilders, DataLoggers, and Dispatchers use child *art* processes for their filtering, analysis and data logging tasks. The *art* processes are fed Fragments through a shared memory interface.

- Event building is done in shared memory; Fragments are received directly into shared memory to avoid copying data
- Dispatchers wait for connections from external *art* processes, then start a companion *art* process for each connection
  - Allows events to be pre-selected for monitoring based on *art* filters



Message Logging and Metrics

artdaq uses TRACE and *art*'s MessageFacility for message logging and a custom metric interface for run-time metric reporting.

- Designed to be a "fire-and-forget" system with minimal impact on operations
- TRACE is a high-speed memory-based messaging interface
  - Messages can be enabled and disabled at run-time
- MessageFacility supports a plugin-based output
  - *artdaq* provides a MessageFacility viewer for displaying messages
- Metric data is aggregated on a dedicated thread
  - Metric Plugins handle sending metrics to various metric reporting services

MessageFacility MsgViewer

ARTDAQ Message Viewer

Total received messages 125

Total displayed messages 125

Message Filter

Set Filters

Categories

AutodetectTransfer
BoardReader_BoardR
DataLogger1_DataRe
DataLogger1_Shared
DataReceiverManage
DataSenderManager
Dispatcher1_DataRe
Dispatcher1_SharedM
EventBuilder1_DataR
EventBuilder1_Share
EventBuilder2_DataR
EventBuilder2_Share

Hosts

ironwork.fnal.gov

Applications

BoardReaderMain (884
BoardReaderMain (884
DataLoggerMain (8843
DispatcherMain (8840)
EventBuilderMain (8837
EventBuilderMain (8839
art (24362)
art (25468)
art (25469)
art (27687)
art (28950)

```
transfer_between_4_and_5_SEND: Constructing ShmemTransfer

Info / DataSenderManager
29-May-2018 08:43:08 CDT
ironwork.fnal.gov (131.225.80.211)
UDPMessage 6 / PID 24362 / DataSenderManager.cc:96
art / DAQ / Booted
enabled_destinations not specified, assuming all destinations enabled.

Info / FastCloning
29-May-2018 08:43:08 CDT
ironwork.fnal.gov (131.225.80.211)
UDPMessage 7 / PID 24362
art / RootOutput:normalOutput@Construction / ModuleConstruction
Initial fast cloning configuration (user-set): false

Error / fileAction
29-May-2018 08:43:08 CDT
ironwork.fnal.gov (131.225.80.211)
UDPMessage 8 / PID 24362
art / PostProcessEvent / run: 148 subRun: 1 event: 2
29-May-2018 08:43:08 CDT  Opened output file with pattern "/home/eflumerf/Desktop/artdaq-

Info / xmlrpc_commander
29-May-2018 08:43:10 CDT
ironwork.fnal.gov (131.225.80.211)
UDPMessage 18 / PID 8840 / xmlrpc_commander.cc:374
DispatcherMain / Early / pre-events
Parameter Set Loaded.

Info / xmlrpc_commander
29-May-2018 08:43:10 CDT
ironwork.fnal.gov (131.225.80.211)
UDPMessage 19 / PID 8840 / xmlrpc_commander.cc:374
DispatcherMain / Early / pre-events
Parameter Set Loaded.

Info / Dispatcher1_SharedMemoryEventManager
29-May-2018 08:43:10 CDT
ironwork.fnal.gov (131.225.80.211)
UDPMessage 20 / PID 8840 / SharedMemoryEventManager.cc:265
DispatcherMain / InRunMap::Running / Sequence ID 41
Starting art process with config file /tmp/partition_0/artConfig_5_1527601390440947.fcl

Info / Dispatcher1_SharedMemoryEventManager
```

All Messages

Error
Warning
Info
Debug

Search

Search / Next

Clear

Pre-filtering

Suppression

Throttling

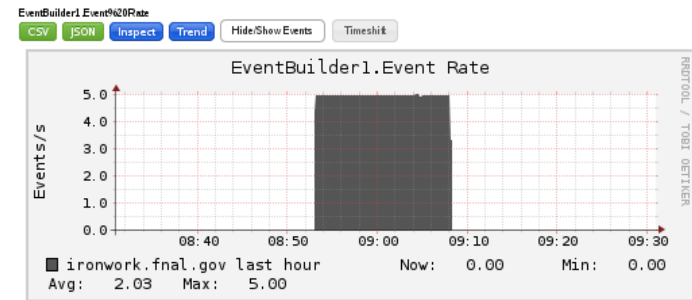
Speedy Mode

Compact View

Pause

Clear

Exit



Fermilab

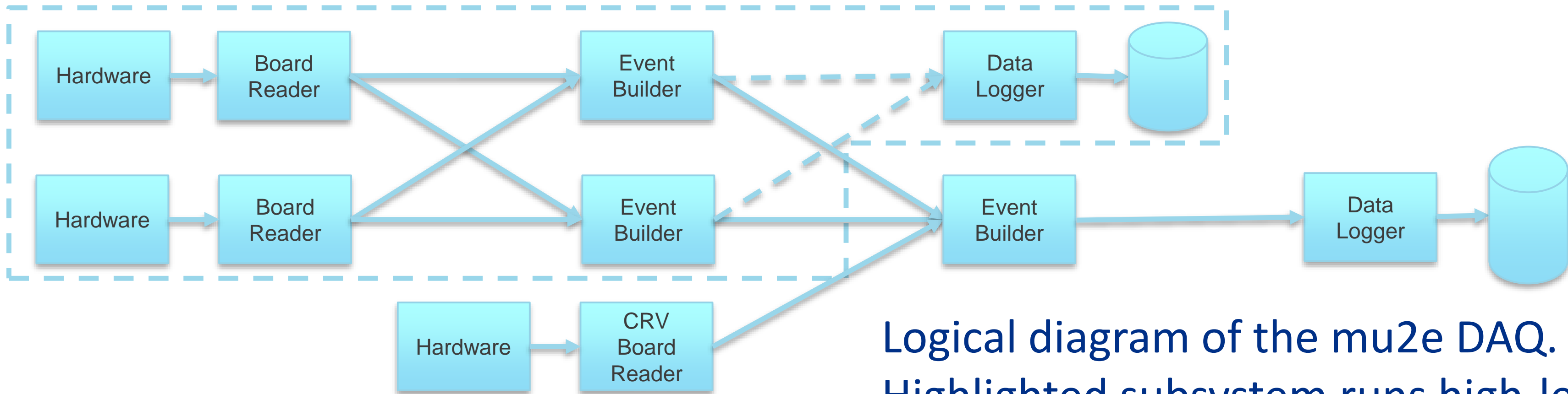
Flexibility of the *artdaq* Framework

- *artdaq* is designed and implemented as a minimal "backbone" with additional functionality provided through various plugins
  - The cost of flexibility is more complex configurations
  - *otsdaq* project provides a "quick-start" environment, with preconfigured plugin selection
- The provided library of components is sufficient for most experiments
  - Experiments provide Fragment "Overlay" classes describing their data format
  - If new plugin implementations are developed by experiments, they can often be integrated into the framework and made available to other experiments

artdaq Framework Scalability

artdaq is designed to be scalable, from simple test systems with a single simulated data source, to complete HEP detector installations such as DUNE.

- Modular design allows for additional components to be added as needed
- System supports "system of systems" architecture to handle any input load



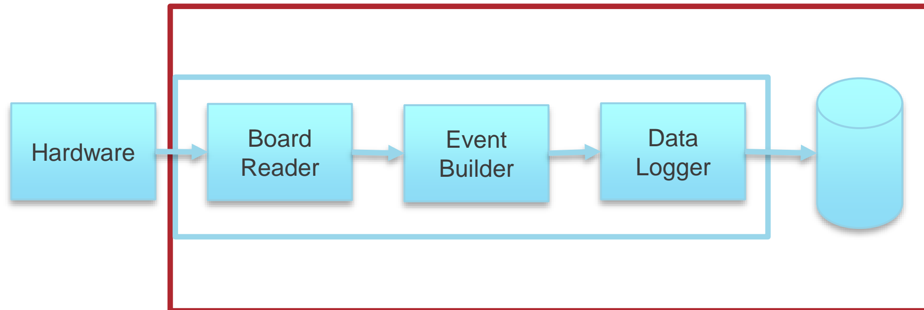
Logical diagram of the mu2e DAQ. Highlighted subsystem runs high-level trigger and requests data from CRV

Fermilab

artdaq Framework Scalability

artdaq is designed to be scalable, from simple test systems with a single simulated data source, to complete HEP detector installations such as DUNE.

- *otsdaq* is distributed in a virtual machine, allowing simple systems to run encapsulated on a Windows host machine
  - Data is written to host's disk and accessible from host OS
  - Hardware communicates using Ethernet protocol forwarded to virtual machine

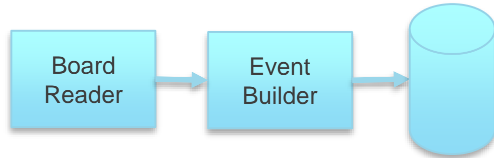


Logical diagram of the CCD DAQ, which started by modifying the *otsdaq* virtual environment

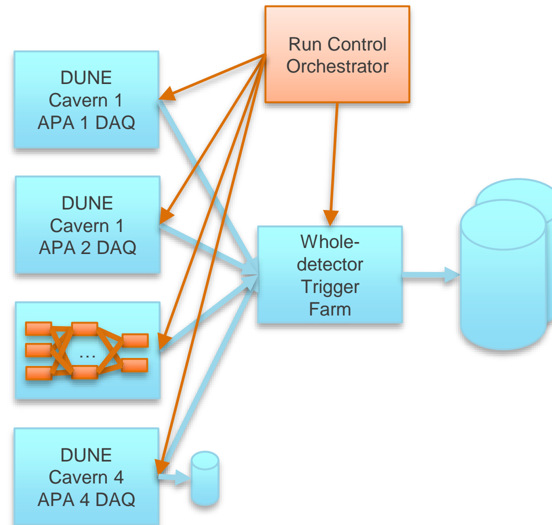
artdaq Framework Scalability

artdaq is designed to be scalable, from simple test systems with a single simulated data source, to complete HEP detector installations such as DUNE.

- In addition to scaling up from test systems to full detector DAQ, *artdaq's* design allows scaling back down to identify and correct issues
  - Modular configuration means that the same configuration can be used for both large and small systems



Simplest possible *artdaq* system, with the BoardReader using simulated Fragments



Conceptual architecture for DUNE DAQ using independent subsystems

Fermilab

Summary

- *artdaq* simplifies SCD's task of supporting multiple experiments' DAQ systems
  - Improvements made in the core framework for one experiment can also benefit other experiments
- By supporting *art* analysis modules in the DAQ, we reduce the number of frameworks necessary for scientists to learn to make quality triggering and monitoring algorithms.
- The *otsdaq* product allows rapid deployment for simple DAQ systems
- *artdaq* supports diverse DAQ needs
  - Mu2e has a complex triggering/filtering algorithm that must run in a very tight time budget
  - protoDUNE (and eventually DUNE) have very high data throughput requirements
- Supporting experiments is paramount
  - Each member of the *artdaq* team also assists the DAQ group for at least one experiment using *artdaq*

Outlook/Future Plans

- HEP Experiments continually challenge the limits of DAQ computing
  - *artdaq* development is driven by meeting and exceeding experiments' requirements
  - "When a physicist asks for a certain rate to disk, make sure the system can handle five times as much, because in a year, they'll be asking for that!"
- Supporting multiple experiments helps keep the framework from becoming too specialized
  - DUNE requires high data rates, but mu2e is interested in high-performance filtering
- *otsdaq* is a productized version of *artdaq* that can be used for any DAQ
  - Available resources constrain the level of support for external users
  - We are working on a comprehensive set of tutorials that should be sufficient to get from 0 to DAQ without expert assistance

Backup

For more information about *artdaq* and *otsdaq,* please see our project web sites:

https://cdcvs.fnal.gov/redmine/projects/artdaq

https://cdcvs.fnal.gov/redmine/projects/artdaq-demo/wiki

https://otsdaq.fnal.gov

Thank You!

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

The DAQ System - BoardReader

Front-End readout hardware is connected to a computer running the first software component in an *artdaq* system: the BoardReader

- A Fragment Generator is an experiment-defined class which reads data from the hardware and packages it in artdaq Fragments
  - Fragment Generators are provided for hardware using the Ethernet protocol provided by the framework's VHDL
- The Fragment defines a header containing all of the information necessary for artdaq to transport the data through the system
- BoardReaders can also be configured to listen for "data requests", allowing for software-defined data flow and continuous readout schemes

The DAQ System - EventBuilder

BoardReaders send Fragments to the EventBuilder, which assembles them into events. It runs an art analysis process, which can perform filtering and initial reconstruction workflows.

- Artdaq's data output is accomplished through art, so some experiments terminate the artdaq data path at the EventBuilder level (EventBuilder/DataLogger functionality combined)
- The artdaq RoutingMaster can be placed between any two layers of an artdaq system
  - Routing Policy takes available buffers on the receiver as input, creates Routing Table linking destination to event number for the senders
  - Round-robin, load-balancing, incremental routing policies provided
  - Plugin type, experiments may customize as needed

The DAQ System - DataLogger

Due to cost, DAQ systems are commonly equipped with dedicated nodes with large disk arrays and network connections that serve as the primary data loggers and host the transfer to offline/long-term storage.

- DataLogger application receives completed events which pass the art filters from EventBuilders and writes them to disk in art/ROOT format
- *art* process in DataLogger only writes to disk and sends data to Dispatcher for online monitoring

The DAQ System - Dispatcher

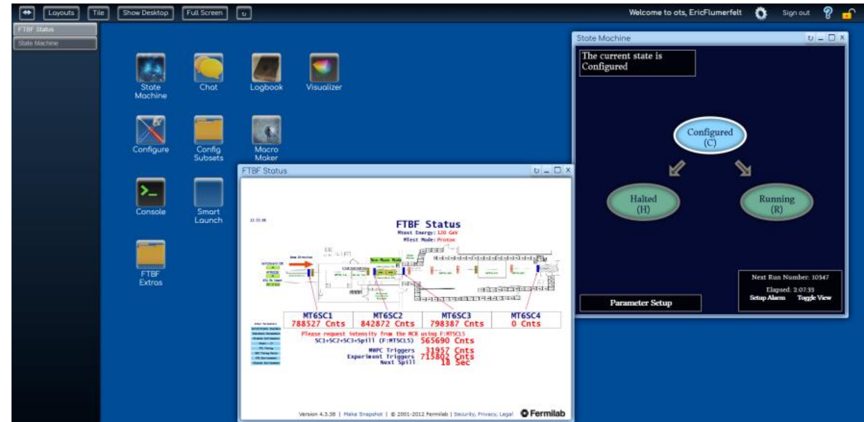
The *artdaq* Dispatcher application receives data from the DataLogger on a "best effort" basis. This disconnect ensures that online monitoring does not affect data-taking.

- *art* Online Monitor processes register with the Dispatcher, which starts a companion *art* process connected to its shared memory segment
  - Companion *art* process can do pre-filtering before sending data to the Online Monitor
  - Transfer plugin is used between Companion and Online Monitor
  - Registration process allows Online Monitors to be connected and disconnected asynchronously, as well as have different versions of the software set up

Run Control

The *artdaq* Framework provides several mechanisms for starting and controlling an *artdaq*-based system. Functionality is highly modularized, and individual pieces can be swapped as needed. State machine commands are received and interpreted by the *artdaq* applications via a “commander” plugin, allowing them to be driven by any desired messaging protocol.

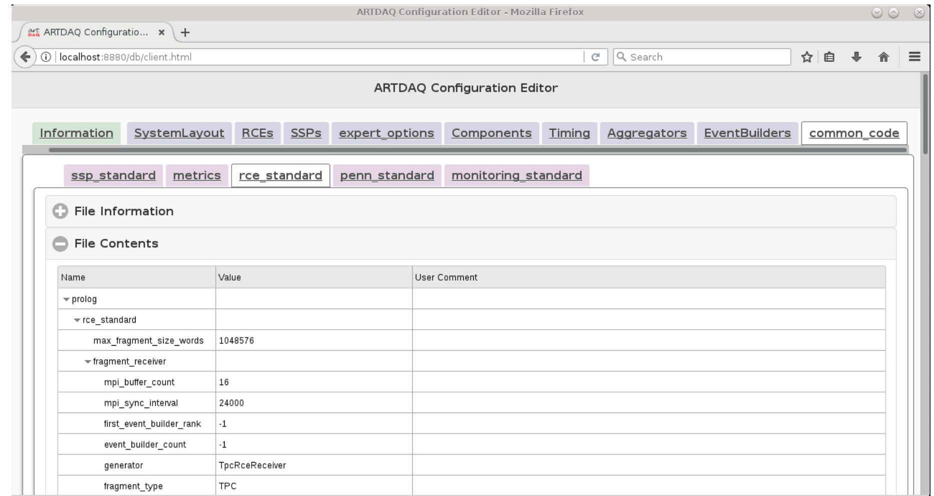
- The DAQInterface package bundled with *artdaq* is a simple python-based console run control, with optional bindings for controlling it from higher-level applications
  - Communication with *artdaq* is through the provided xmlrpc_commander plugin
- *otsdaq* implements XDAQ SOAP messaging and also provides XDAQ-based application management



Configuration

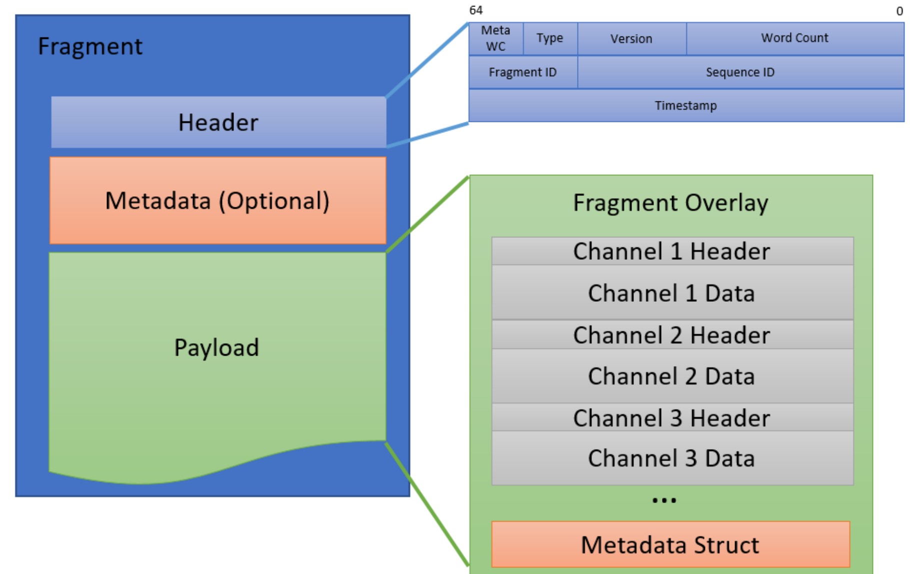
All *artdaq* applications and components are configured using FHiCL (Fermi Hierarchical Configuration Language) documents. FHiCL was developed at Fermilab for use in configuring offline analysis jobs, and has many features which are useful in *artdaq*. A MongoDB-based database stores FHiCL documents converted into a form of JSON, and a GUI configuration editor is also provided.

- Configuration documents not only specify system parameters, but also which plugins to load
- Applications can define allowable configurations and perform validation of required parameters



Design of *artdaq* - Fragment

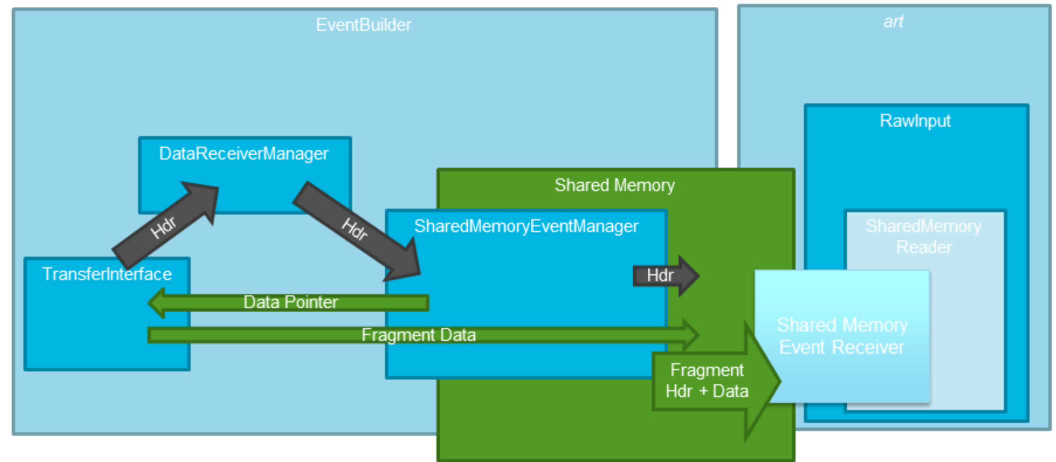
- Header includes Sequence ID (which is used to create event numbers in *art*), Timestamp, Type, Fragment ID (identifying the source component of the Fragment), and size
- Metadata includes information about the BoardReader which created the Fragment
- Payload is experiment-defined and can be accessed via a “Fragment Overlay”, a helper class which defines a Fragment Type and access methods for the data
  – Overlays are used in reconstruction and analysis jobs



Design of *artdaq* – *art* Process

EventBuilders, DataLoggers, and Dispatchers use child *art* processes for their filtering, analysis and data logging tasks. The *art* processes are fed Fragments through a shared memory interface.

- Event building is done in shared memory; Fragments are received directly into shared memory to avoid copying data
- Dispatchers wait for connections from external *art* processes, then start a companion *art* process for each connection
  - Allows events to be pre-selected for monitoring based on *art* filters



Fermilab