

Notes from *artdaq* chat

02/09/2016

How to log into ICARUS machine:

- (1) You need a CERN computing account, which means you've registered with CERN.
- (2) `ssh username@lxplus.cern.ch`
- (3) `ssh icadaq@PC03-warp`
 - (a) Password is the username
 - (b) This is a machine in use, so please don't just start communications with hardware willy-nilly.
Unless you're communicating directly with someone, probably OK to look around, but don't do anything.

Where things are on that machine:

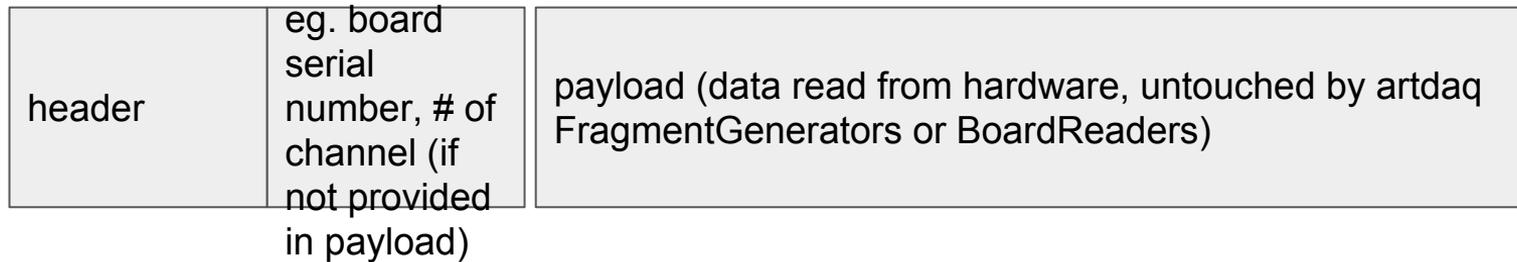
- All the artdaq-demo stuff is in /home/icadaq/artdaq-demo-base
- I've started an artdaq area for icarus (*icartdaq* :)): /home/icartdaq
- Repositories, until we get something more official (soon):
 - <https://github.com/wesketchum/icartdaq>
 - <https://github.com/wesketchum/icartdaq-core>
 - Mostly up to date
-

ARTDAQ Fragment Reference

https://cdcvns.fnal.gov/redmine/projects/artdaq-demo/wiki/Fragments_and_FragmentGenerators_w_Toy_Fragments_as_Example
s

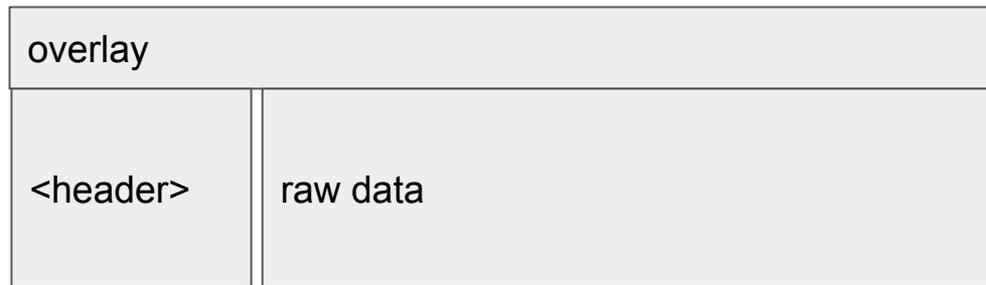
ARTDAQ Fragment Intro

artdaq::Fragment

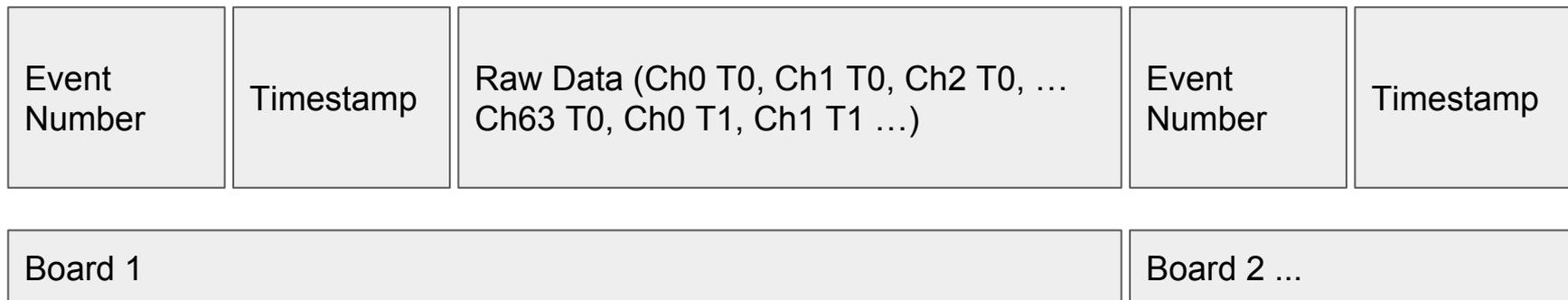


experiment-specific

overlay



ICARUS fragment structure



<https://github.com/wesketchum/icartdaq-core/blob/master/icartdaq-core/Overlays/CAEN2795Fragment.hh>

DarkSide example: <https://cdcvs.fnal.gov/redmine/projects/darksidecore/repository/revisions/develop/entry/darksidecore/Data/V172xFragment.hh>

Board Block

(CAEN2795 Fragment payload is vector<BoardBlock>)

```
struct BoardBlock {  
  
    uint32_t evNum : 24;  
  
    uint32_t unused : 8;  
  
    uint32_t timestamp;  
  
    uint32_t data[NSamples][NChannels];  
  
}
```

The order of the indicies of the data array should be chosen so that they correctly

CommandableFragmentGenerator

Abstract base class with the following:

- start()
- getNext()
- pause()
- resume()
- stop()
- report()

The one I made for ICARUS: https://github.com/wesketchum/icartdaq/blob/master/icartdaq/Generators/CAEN2795_generator.cc

Metric Reporting

<https://cdcv.s.fnal.gov/redmine/projects/artdaq-utilities/wiki/Artdaq-ganglia-plugin>

https://cdcv.s.fnal.gov/redmine/projects/artdaq-utilities/wiki/Sending_metric_data_from_User_Code

http://mu2eddaq01.fnal.gov/ganglia/?c=mu2e%20DAQ&h=mu2eddaq01-data.fnal.gov&m=load_one&r=hour&s=by%20name&hc=4&mc=2

https://cdcv.s.fnal.gov/redmine/projects/ds50daq/wiki/Enabling_Ganglia_monitoring_on_the_WH14N_E_teststand

<https://github.com/ganglia/monitor-core/wiki/Ganglia-Quick-Start> <= Ubuntu instructions

Notes from the whiteboard, part 1

To-do

1. Change the ToyFragment classes to make header/metadata/payload more clear.
2. Check artdaq-demo documentation for places where the artdaq/artdaq-core breakdown is not correctly represented.
3. Create git repos for ICARUS

CommandableFragmentGenerator Questions/Comments:

1. Do we want to make more of the protected methods in CommandableFragmentGenerator virtual? [start, stop, report?] [also see notes on slide 8]
2. The samples that we provide in the demo should have all of the methods that we want users to consider included, even if they are empty. We should also include comments that talk about what can be done in each of them, etc. Also include relative path to CommandableFragGen so users can easily look at it.

Notes from the whiteboard, part 2

Proposed best practice:

1. Treat the data read out of the hardware as immutable.

Additional to-do items or things to be considered:

1. Modify the metadata interface in `artdaq::Fragment` to make it easier/more robust for users to update the metadata contents after the `Fragment` is constructed.
2. Add the use of MRB to the demo.
3. Create an `artdaq-skeleton` (and probably `artdaq-core-skeleton`) package that users can use to get started.

Meeting summary

We first talked about the layout of data in `artdaq::Fragments` and experiment-specific payloads. Also, the role of `Overlay` classes in interpreting the experiment-specific payloads. Wes pointed out that the existing `ToyFragment` handling in the demo, unfortunately, made this harder to understand rather than easier. We pointed out that the definition of the `Metadata` structs in the `Overlay` class headers is simply a convenience place to do that; the `metadata` struct is not used in the `Overlay` class methods.

Next, we talked about current contents of Wes' ICARUS fragment overlay class(es) and generators. (Links on page 6.)

Then we talked a bit about metrics and reporting.

(I've glossed over a couple of details - please see the Whiteboard Notes on the previous two pages for additional information; KB)