General Introduction

The *artdaq* data acquisition toolkit currently provides functionality for data transfer, event building, event reconstruction and analysis (using the *art* analysis framework), process management, system and process state behavior, control messaging, local message logging (status and error messages), DAQ process and *art* module configuration, and the writing of event data to disk in ROOT format. Functionality that will be available soon includes centralized message logging and display, and DAQ monitoring (the health and performance of the DAQ system).

Future additions to *artdaq* are expected to include a graphical interface to the run control and state management facilities, management and archiving of configurations, and infrastructure for remote control and monitoring. In addition, all future enhancements to the *art* framework and its utilities will naturally be available to users of *artdaq* since *art* is a core component of the toolkit.

The toolkit is designed to provide many of the core functions that are needed in a DAQ system and allow experimenters to focus on the components that are specific to their experiment. These experiment-specific components are

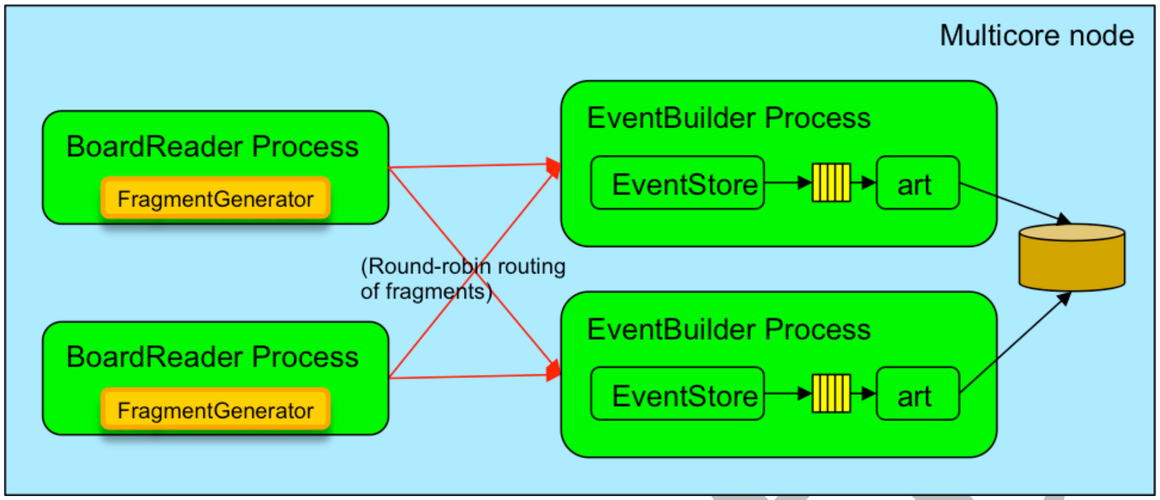
- the software that configures and reads the data from the experiment-specific hardware,
- the analysis modules that run inside the *art* framework to reconstruct, compress, and/or filter the event data,
- the data quality monitoring algorithms that verify the quality of the data.

The artdaq-demo package demonstrates the features that are currently available in *artdaq* using a small test system that includes simulated generation of data fragments and a Huffman compression module.

Introduction to the Demo System

The demo system includes scripts to start, configure, and run a small 2-by-2 system on a single Linux host. As shown in the figure below, this sample system has two BoardReader processes and two EventBuilder processes. (The demo system can be extended to demonstrate any number of BoardReader and EventBuilder processes, and to demonstrate a system that is distributed across multiple Linux hosts.)

In a system with real hardware modules, BoardReader processes are responsible for configuring the modules and reading data from them, and in the current design the model is to have one BoardReader process per front-end card. (BoardReaders are also responsible for sending the data fragments that are read from the hardware modules to the EventBuilder processes.) BoardReader processes make use of FragmentGenerator instances to interact with the hardware. FragmentGenerator instances are instantiations of C++ classes that are experiment-specific and implement the artdaq::FragmentGenerator interface. These can either communicate with real hardware or provide simulated data. In the demo system, the FragmentGenerator instances generate simulated CAEN V1720 digitizer data.



The EventBuilder processes build complete events using instances of the EventStore class and pass them to the *art* framework. In an experiment DAQ, the *art* framework can be configured to run analysis modules that are written by members of the experiment or are part of the core *art* suite. In the demo, several sample modules are provided, and the standard example uses a module that compresses the digitizer data.

Description of the Sample System

The instructions for running the sample demo system (available on the artdaq-demo Redmine Wiki) include the following steps:

- start the MPI program – This uses the Process Management Tool that is part of *artdaq* and the mpirun program that is part of MPI.
- initialize (configure) the system – This step sends configuration strings in FHICL format to the DAQ processes using XMLRPC.
- start a run – This step sends the start command and the run number to the DAQ processes using XMLRPC. The commands are sent in a well-defined order so that downstream processes are ready when the data flow is started.
- stop a run – This step sends the stop command to the DAQ processes using XMLRPC. In this case, upstream processes are notified first so that the flow of data is stopped at the front-end and all events are drained from the system.
- shutdown the system – The shutdown command is sent using XMLRPC, and the MPI program is stopped.
- examine the events in one or both of the disk files that are written (one per EventBuilder) – An *art* module is used to display data from the events.

The goal of the sample system is to demonstrate the features of *art* and *artdaq*, including the following:

- data transfer using MPI – The demo system software is built against an Ethernet version of the MPI library so that tests on multiple nodes can be run without needing specialized hardware (such as an Infiniband network).
- assembly of the data fragments from the two BoardReader processes into full events – The EventBuilders write complete events to disk. When the data on disk is examined with the “event dump” module, both fragments are shown for each event.
- compression of the digitizer data by a module in the *art* framework – As described in the instructions for running the demo, the selection of whether the compression algorithm is run can be made at configuration time. The event dump utilities, which are used to examine the data on disk, de-compress the data, if needed, before displaying information about the data in each event.
- startup and control of the MPI program (the set of BoardReader and EventBuilder processes) using the *artdaq* Process Management Tool
- state behavior – A subset of the state transitions that are supported by *artdaq* are shown in the sample system demo. These are initialization (configuration), start (begin run), stop (end run), and shutdown.
- control of the DAQ processes using XMLRPC control messages. The control commands are sent to the DAQ processes by scripts that were developed as part of *artdaq*. These scripts use the xmlrpc client program.
- configuration of the DAQ processes and the *art* framework using FHICL configuration strings – In the sample system, the configuration strings are generated automatically by the scripts which help to run the system. Interested users can see the contents of these strings in the console output when they are echo-ed by the applications that receive them.
- writing of event data to disk files in ROOT format

Next steps

There are several useful exercises that interested users of the demo system may want to try. These include the following:

- create one or more new types of simulated fragments and include those in the sample system
- create one or more additional *art* modules and run them in the sample system
- run any number of BoardReader processes to create fragments and run any number of EventBuilders (within the resource limits of the single PC).
- run the BoardReader and EventBuilder processes on multiple PCs (that all have equivalent access to the necessary binaries and libraries on disk)

Instructions for doing these exercises will be added to the Redmine Wiki soon.

Also, additional documentation will be provided for the demo, including the following:

- a description of the configuration options that are available for the DAQ processes and the *art* modules
- a description of the FragmentGenerator interface and instructions for creating C++ classes that implement it
- other material as needed

Reference Material

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

artdaq talks and paper
- available from the Redmine Wiki page

art Redmine Wiki
- https://cdcvs.fnal.gov/redmine/projects/art/wiki