ROOT usage by the *art* framework and its users

Kyle J. Knoepfel

ROOT Users' Workshop in Sarajevo, Bosnia and Herzegovina

13 September 2018

art framework

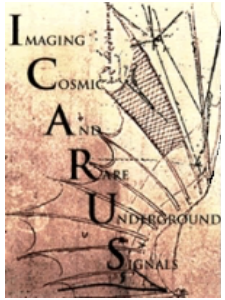
- We support the offline (and some online) processing for 11 projects/experiments.
  - Intended to be used for small-scale jobs and large-scale production campaigns.
- Hierarchical data-processing levels ($run \supset subrun \supset event$)
  - Support concurrent execution of events
- **Experiments decide** what the event represents.
  - Some experiments make different choices for different stages of processing.
- Data-processing workflows are assembled by a configuration file

```
art -c config.fcl -s in.root -o out.root …
```

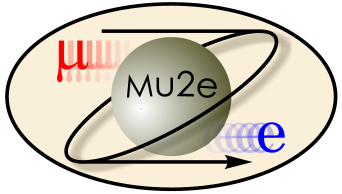
  - All processing elements can be user/experiment-defined (most are).
- Experiment-specific workflows are executed via their configuration files, **not** via different executables.

art provides the framework needs for ~2k physicists

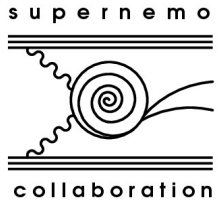
artdaq project



LArIAT experiment

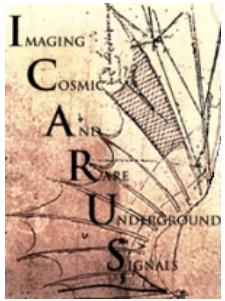


Previous and potential users

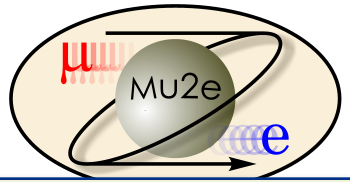


art provides the framework needs for ~2k physicists

artdaq project



LArIAT experiment



art development guided by stakeholders, who meet weekly at a dedicated meeting:

- Discussion of upcoming changes and issues with stakeholders
- Sharing among experiments
- Same governance model from the beginning

Distinctive aspects of the *art* user community

- Many *art* users are doing development for experiments that are not yet running:
  - Reconstruction algorithms not yet finalized
  - Workflows are under development
- They are involved in software development including event-generation, material simulation, processing raw data, reconstruction, to analyzing quantities of physical interesting.
- They are defining experiment-specific data models.
- They exercise aspects of ROOT that most end-users of frameworks would not:
  - Exploring compression algorithms/levels, implementing I/O rules, etc.
- They are generally willing to (drastically) rethink any stage of the physics workflow:
  - Event representation, exploring other I/O libraries, etc.

art's efforts

- **Development is forward-looking**. We aim to (e.g.):
  - run *art* efficiently on HPC machines
  - deliver cutting-edge software tools and incorporate best-of-class C++ libraries
  - enable experiments to benefit from modern language features
- **Experiment support**
  - We actively contribute to the code bases of *art*-using experiments/projects
  - Design guidance and code reviews at the request of experiments
  - Small-scale profiling efforts at the request of experiments
- **User support**
  - Configuration description and validation suite
  - We support open-source Clang builds on Linux and macOS
  - Profiling tools, data-dependency graph generator, etc.

art's relationship with ROOT

- Current *art* developers have:
  - Greatly benefited from "next-door" advice from one of the ROOT experts at Fermilab
  - Contributed to the development of ROOT 6
  - Participated in ROOT's planning meetings

- Since the beginning (2009) *art* has used ROOT
  - It's what *art* users depend upon for I/O and analysis work
  - It has provided type-introspection facilities that the framework has relied upon

ROOT I/O in *art* and its experiments

- *art*'s I/O system is file-format-agnostic
  - To choose a specific file format, specify the appropriate input source/output module, which is loaded at run-time based on the configuration
- *art* provides its own input sources/output modules (e.g. `RootInput`/`RootOutput`), but others are available:
  - Many different kinds of experiment-defined input sources
  - Really only one useful output module right now (`RootOutput`)
- All user-facing interactions with input/output files happen under the covers:

```
void produce(Event& e) {
  auto const h = e.getValidHandle<std::vector<Hit>>(tag_); // May read from input source
  auto const nHits = h->size();
  e.put(std::make_unique<std::size_t>(nHits));             // Will write to output file
}
```

art's `TFileService`

- Allows user-created ROOT constructs to be organized into a ROOT file based on the module in which they were created.

```
art::ServiceHandle<TFileService> tfs{};
auto h1f = tfs->make<TH1F>("h1f", "Histo 1", 100, 0., 200.);
auto tree = tfs->make<TTree>("tree", "My analysis tree");
auto g1 = tfs->makeAndRegister<TGraph>("graph", "A graph", 10);
```

- The histograms are attached to a file, whose name is given at the command line:

```
art -c config.fcl -T myFile.root -o out.root
```

- This is how *art* modules should interact with ROOT. This allows *art* to:
  - Open/close the ROOT file without user interaction
  - Manage ROOT's global state to avoid collisions with *art*/ROOT files

Further ROOT use

- For `TFileService` and the `RootOutput` module, *art* is able to switch to new ROOT files based on some condition being satisfied:

Number of processed events	Number of processed input files
Number of processed subruns	File size
Number of processed runs	File age

  - Self-registering of ROOT objects makes this difficult to do seamlessly.

- We use SQLite databases for storing *art* metadata
  - We have written a SQLite VFS that allows writing a database to a TKey in a TFile
  - Metadata stored alongside event data, which are represented by TTrees

Dictionary generation

- *art* provides (but does not require) a CMake-based build system, which includes facilities for building dictionaries

```
# my/dir/with/classes{.h,_def.xml}
include(ArtDictionary)
art_dictionary(my_lib1 my_lib2)
```

- The user creates the requisite `classes.h` and `classes_def.xml` files, and *art*'s build system creates the dictionary at build-time.

- *art* also verifies the consistency of checksums (thanks to CMS for the code)

```
INFO: class 'critic::test::Data' has a different checksum for ClassVersion 10.
Incrementing ClassVersion to 11 and assigning it to checksum 914314239
WARNING: classes_def.xml files have been updated: rebuild dictionaries.
```

art framework and *art* ROOT I/O

- *art* framework concepts are independent of an I/O mechanism
  - Experiments may be ready for new framework features, independent of the ROOT version
  - Experiments may be ready for different versions of ROOT at different times

art framework and *art* ROOT I/O

- *art* framework concepts are independent of an I/O mechanism
  - Experiments may be ready for new framework features, independent of the ROOT version
  - Experiments may be ready for different versions of ROOT at different times
- As of August 2018, ROOT support is introduced through a separate package:

- Now possible for experiments to setup *art* and ROOT independent of each other.
- New versions of ROOT do not necessitate new versions of *art* (and vice versa).

Desires for ROOT

- Fewer side effects
  - Remove as much global state as possible
  - Ownership semantics are non-obvious (e.g. `TH1::AddDirectory`)

- Some cleanup
  - As few global-namespace entities as possible.
  - Fewer member functions; more free functions
  - Remove utilities/types that are provided by the C++ standard
  - Updated interface (e.g. can we get rid of '`char const*`' function arguments?)

- Proposals for ROOT 7 fulfill many of these desires.
  - We welcome those changes!

Documenting ROOT's phase space

- ROOT's guarantees
  - What are they?
    - (e.g.) when can forward/backward compatibility be broken?
  - Who informs what these guarantees should be and how?
  - How often are they re-evaluated?

Documenting ROOT's phase space

- ROOT's guarantees
  - What are they?
    - (e.g.) when can forward/backward compatibility be broken?
  - Who informs what these guarantees should be and how?
  - How often are they re-evaluated?
- ROOT's boundaries
  - When does it make sense to use ROOT and when does it not?
  - Tell us what these boundaries are—it builds confidence.
  - These boundaries can change over time!

Documenting ROOT's phase space

- ROOT's guarantees
  - What are they?
    - (e.g.) when can forward/backward compatibility be broken?
  - Who informs what these guarantees should be and how?
  - How often are they re-evaluated?
- ROOT's boundaries
  - When does it make sense to use ROOT and when does it not?
  - Tell us what these boundaries are—it builds confidence.
  - These boundaries can change over time!
- *Having these aspects clearly spelled out would help everyone!*

Thank you.