Toolkit for data reduction to tuples for the ATLAS experiment

Scott Snyder (BNL) <snyder@bnl.gov>
Attila Krasznahorkay (NYU) <Attila.Krasznahorkay@cern.ch>
(For the ATLAS Collaboration)

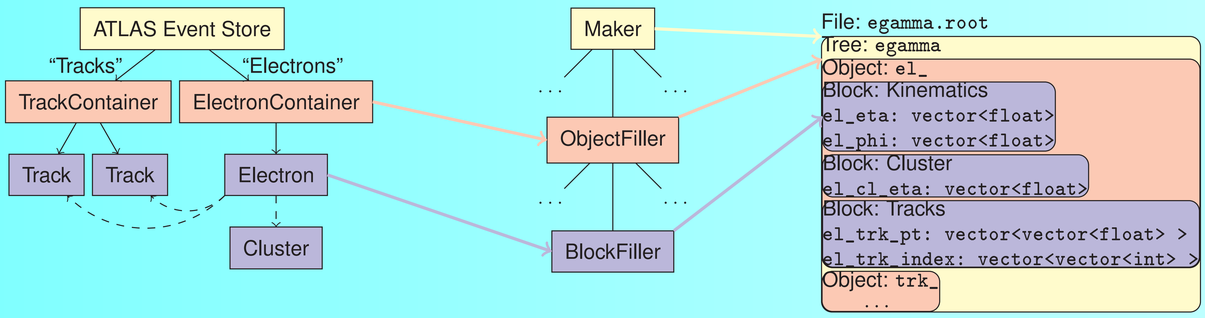
Introduction

The final stage of a physics analysis is most often performed from a tuple-based data format using tools such as ROOT [1]. ATLAS grew to have many such tuples, each produced by a different group with different conventions.
One size doesn't fit all, but ATLAS provided a modular toolkit for producing such tuples. This allows sharing implementations, and more importantly, ensures that all tuples made this way share common naming and conventions.

Design goals

- Tuple making should just copy from the event store (a "blackboard" data store for passing event data between reconstruction algorithms) to the tuple.
- Emphasize flat tuples that can be read with no extra runtime support, but don't preclude also using objects.
- Contents of the tuple can be customized at a relatively fine level of granularity.
- Tuple should be extensible by user code.
- Be independent of the format used to store the tuple.

Overview



Block filler tools

Should take a single object and copy data from it to the tuple; looping over containers of objects is handled by the caller.

```
// Example block filler tool.
struct FourMomFillerTool
 : public BlockFillerTool<FourMom>
{
  // ... Boilerplate omitted.
  // Variables being filled.
  // Class types may also be used.
  float *m_pt, *m_eta, *m_phi;

  // Called once to declare variables to fill.
  virtual StatusCode book() {
    CHECK( addVariable ("pt",  m_pt) );
    CHECK( addVariable ("eta", m_eta) );
    CHECK( addVariable ("phi", m_phi) );
    return StatusCode::SUCCESS;
  }

  // Called for each object.  Framework will
  // set the pointers appropriately.
  virtual StatusCode fill (const FourMom& p) {
    *m_pt  = p.pt();
    *m_eta = p.eta();
    *m_phi = p.phi();
    return StatusCode::SUCCESS;
  }
};
```

Generic components

Block filler and association tools depend on the types of objects being manipulated. However, the core tools for retrieving objects from the event data store, looping over collections, calling block filler tools, and formatting the data into the tuple are mostly independent of the types being manipulated. The standard C++ RTTI is augmented with information about class inheritance relations; this allows generic pointers to be properly converted.

Alternate tuple formats

While all physics analyses so far use ROOT tuples, a prototype exists for writing HDF5 [2]. No changes are needed beyond selecting the alternate backend implementation.

References

[1] R. Brun and F. Rademakers, *ROOT — An Object Oriented Data Analysis Framework*, in Phys. Res. A 389 (1997) 81–86. http://root.cern.ch

[2] The HDF Group, *Hierarchical data format version 5*, 2000–2010. http://www.hdfgroup.org/HDF5

Associations

Tools can associate either from one object to another (single association) or from one object to a set of objects (multiple association). Associations can be represented in the tuple either by adding an index to another object in the tuple ("el_trk_index" above) or by directly "containing" the target object within the source object ("el_cl_eta"). Multiple contained associations can use either a nested vector ("el_trk_pt") or enter additional tuple rows.

Configuration

Assembling a tuple from predefined objects (physics groups will do this to define their tuples):

```
tuple = MakerAlg ('egamma', file = 'egamma.root')
tuple += ElectronTupleObject(3) #level of detail=3
tuple += TrackTupleObject(1)
# An electron with a different name.
tuple += ElectronTupleObject (1, sgkey = 'myEles')
```

Defining an object (usually done by groups responsible for physics objects):

```
ElectronTupleObject = make_SGDataVector_TupleObject \
 # Type name, name in data store, prefix in tuple
 ('ElectronContainer', 'Electrons', 'el_')
ElectronTupleObject.defineBlock \
  # Level-of-detail, name, block filler tool
  (0, 'Kinematics', FourMomFillerTool)

# Add cluster info by associating to cluster object.
ElClusterAssoc = SimpleAssociation \
  (ElectronClusterAssocTool, prefix = 'cl_')
ElClusterAssoc.defineBlock \
  (1, 'ClusterKin', FourMomFillerTool)

# Associate to set of tracks.  Add both track momenta
# directly and indices into track list in the tuple.
TrkClusterAssoc = ContainedVectorMultiAssociation \
 (ElectronTrackAssocTool, prefix='trk_')
TrkClusterAssoc.defineBlock \
  (1, 'TrackKin', FourMomFillerTool)
TrkClusterAssoc.defineBlock \
  (1, 'TrackIndex', IndexFillerTool, Target='trk_')
```

Reading tuples

Tuples in ROOT format can be read directly with ROOT. There are also packages in C++ and Python that allow efficiently reading a tuple while providing an object structure on top of it.

Acknowledgment

This work is supported in part by the U.S. Department of Energy under contract DE-AC02-98CH10886 with BNL.