DUNE Analysis Data in RNTuple

Amit Bashyal, Barnali Chowdhury, Peter van Gemmeren
Argonne National Lab

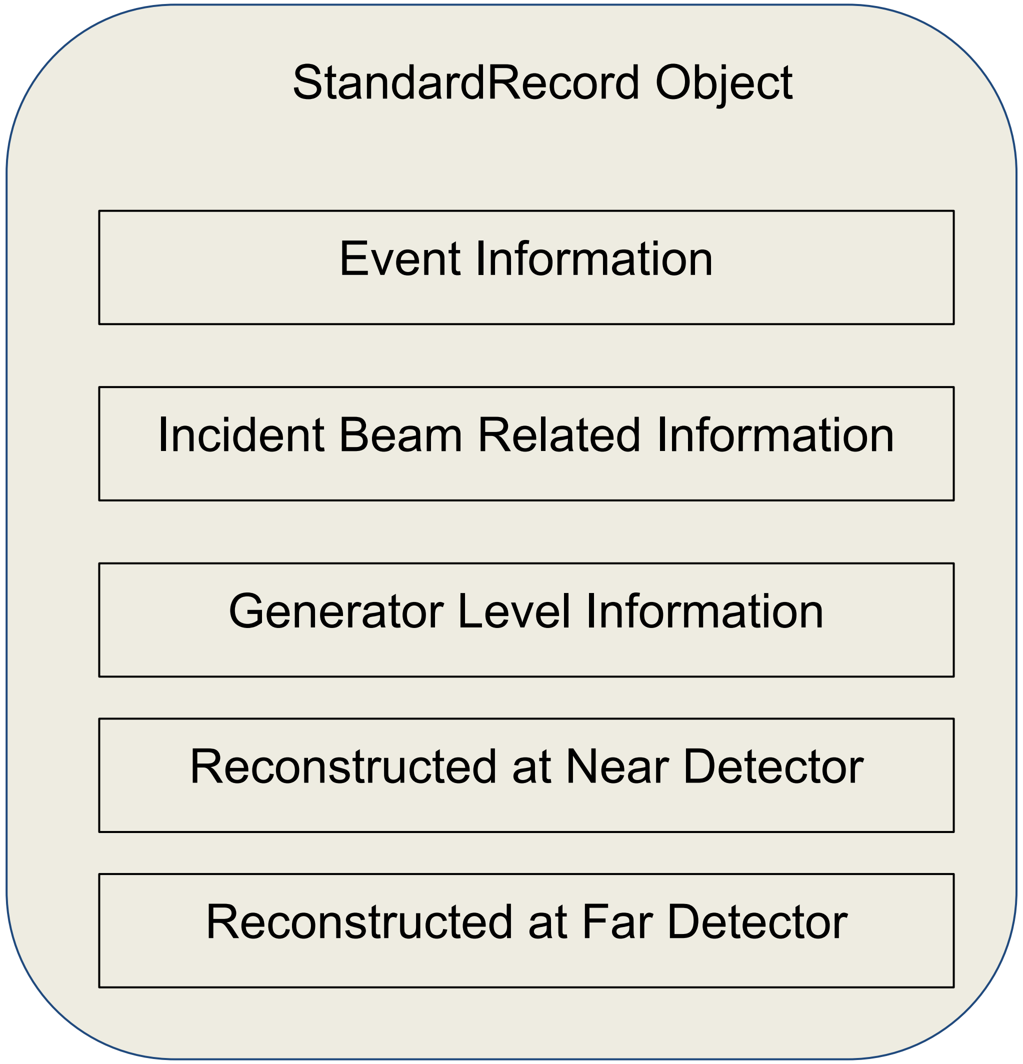
April 03, 2024

DUNE Analysis Data Format

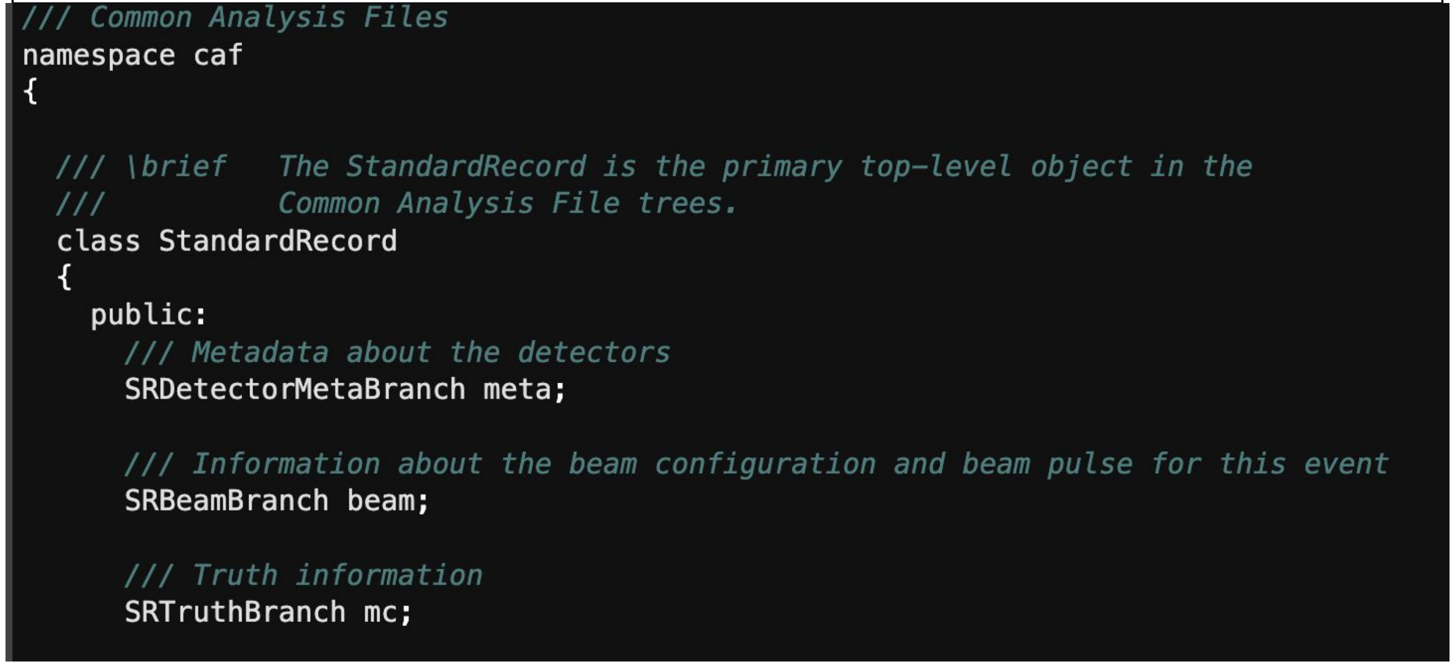
- CAF Data Model
  - Commonly written in ROOT::TTree and HDF5
  - Simpler object oriented with multiple level of hierarchies and segmentation
  - Discard hit by hit (detector level) information with intricate structure
  - Higher-level reconstructed variables from hits are saved for further analysis

- CAF Data Model in RNTuple
  - Following slides show the bare minimum requirements to read/write CAF objects using RNTuple
  - Further implementations based on experiment specific requirements

CAF Data Model and Persistence in RNTuple


StandardRecord Object
Event Information
Incident Beam Related Information
Generator Level Information
Reconstructed at Near Detector
Reconstructed at Far Detector

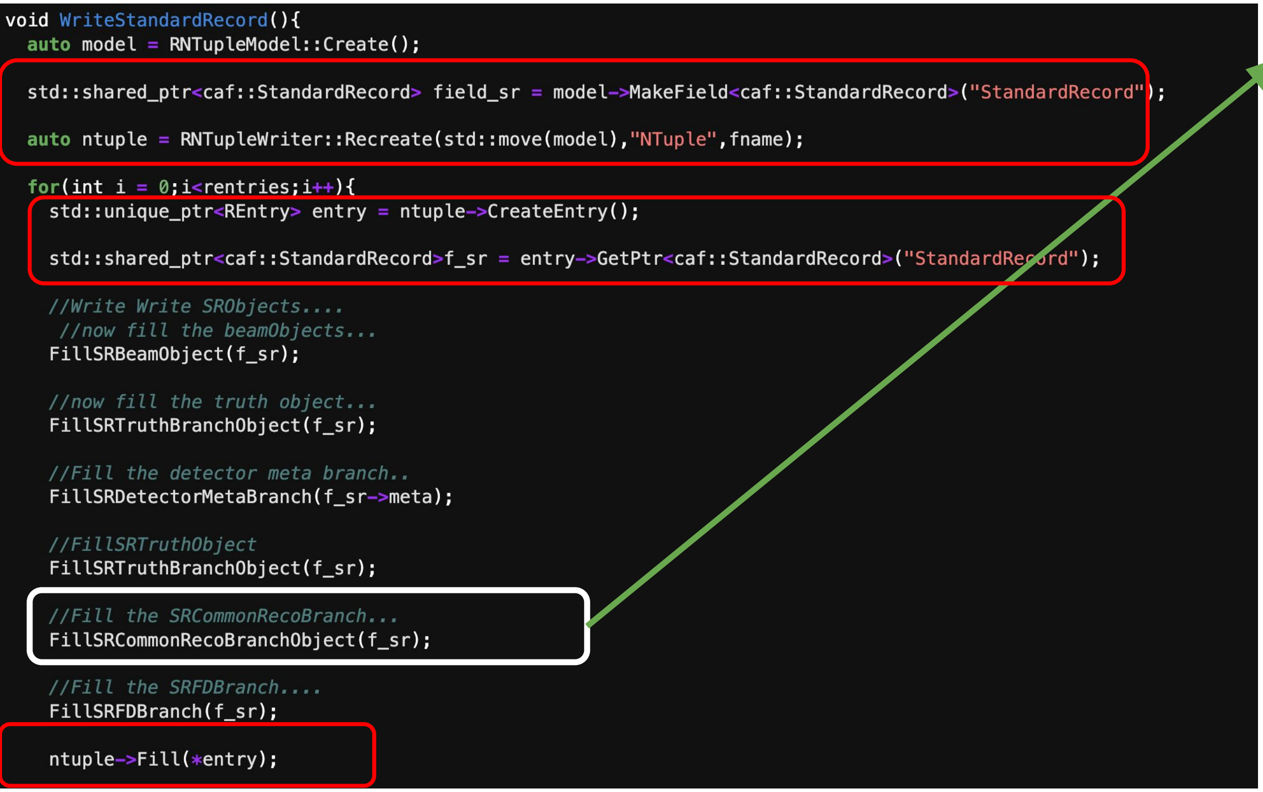
- **StandardRecord (SR)**: Top level CAF object
- Summary of neutrino event
- Information related to neutrino event as SR member objects
- At the basic level made of C++ fundamental types, std::vectors, ROOT::TVector3D and ROOT::TLorentzVector.



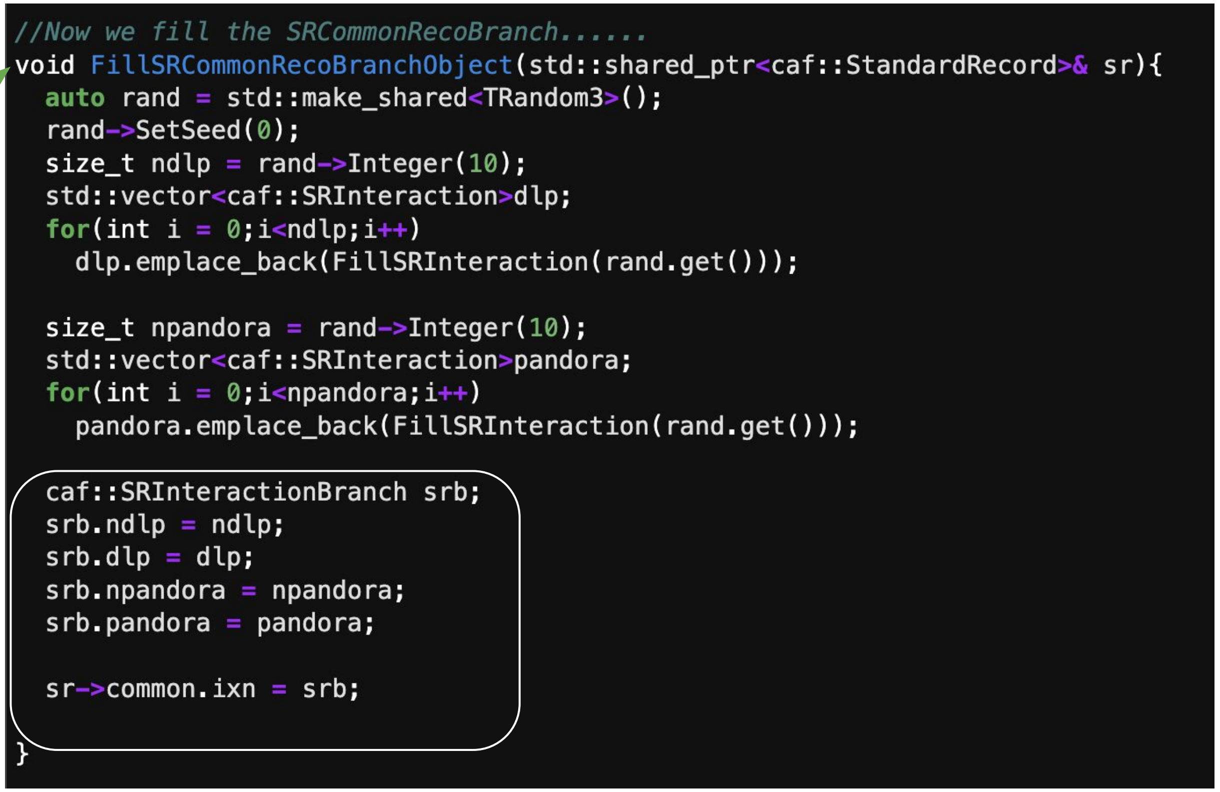
```
/// Common Analysis Files
namespace caf
{

  /// \brief   The StandardRecord is the primary top-level object in the
  ///          Common Analysis File trees.
  class StandardRecord
  {
    public:
      /// Metadata about the detectors
      SRDetectorMetaBranch meta;

      /// Information about the beam configuration and beam pulse for this event
      SRBeamBranch beam;

      /// Truth information
      SRTruthBranch mc;
```

HEP-CCE

Writing CAF Objects



```
void WriteStandardRecord(){
  auto model = RNTupleModel::Create();

  std::shared_ptr<caf::StandardRecord> field_sr = model->MakeField<caf::StandardRecord>("StandardRecord");

  auto ntuple = RNTupleWriter::Recreate(std::move(model),"NTuple",fname);

  for(int i = 0;i<rentries;i++){
    std::unique_ptr<REntry> entry = ntuple->CreateEntry();

    std::shared_ptr<caf::StandardRecord>f_sr = entry->GetPtr<caf::StandardRecord>("StandardRecord");

    //Write Write SRObjects....
     //now fill the beamObjects...
    FillSRBeamObject(f_sr);

    //now fill the truth object...
    FillSRTruthBranchObject(f_sr);

    //Fill the detector meta branch..
    FillSRDetectorMetaBranch(f_sr->meta);

    //FillSRTruthObject
    FillSRTruthBranchObject(f_sr);

    //Fill the SRCommonRecoBranch...
    FillSRCommonRecoBranchObject(f_sr);

    //Fill the SRFDBranch....
    FillSRFDBranch(f_sr);

    ntuple->Fill(*entry);
```

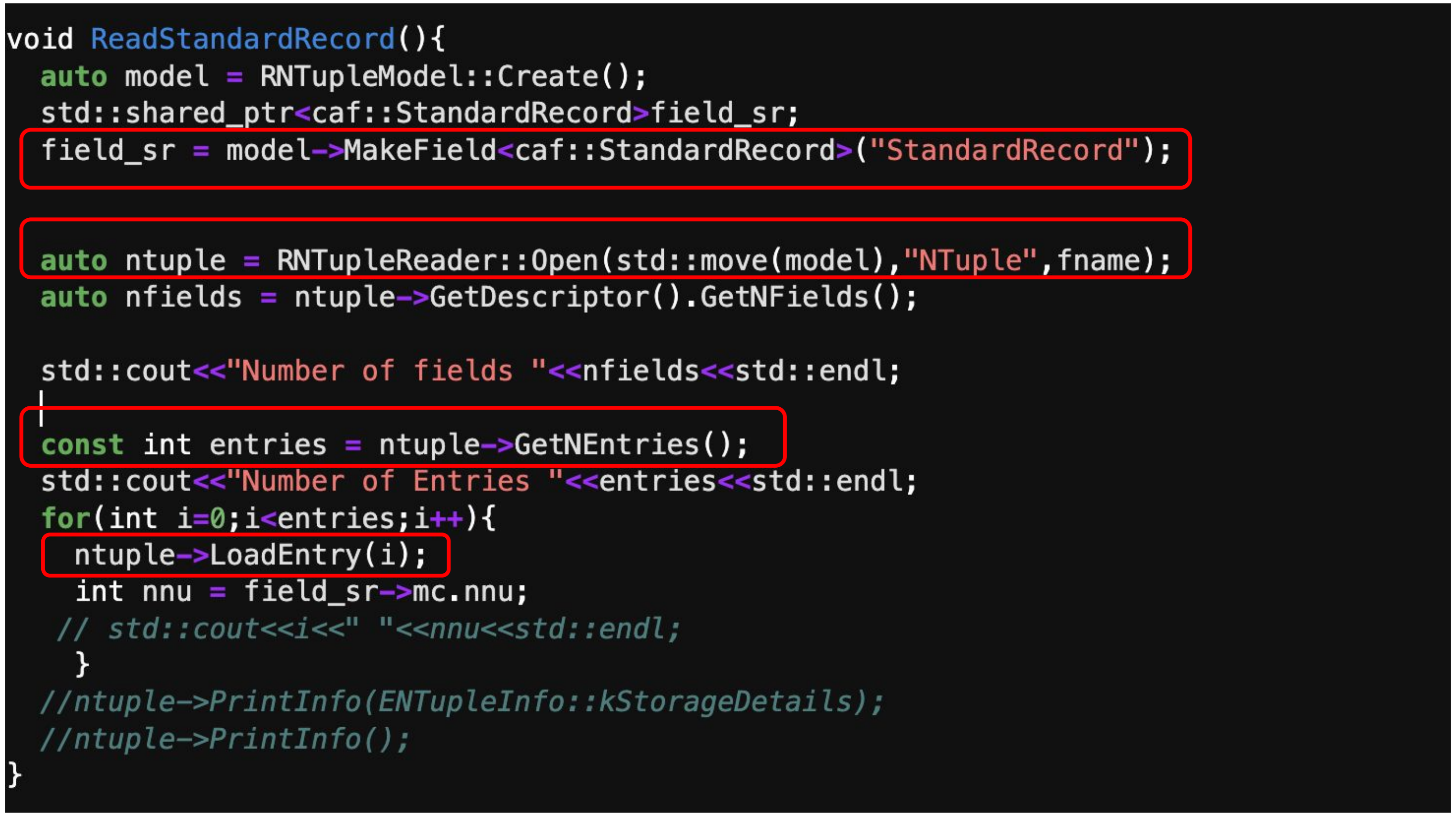


```
//Now we fill the SRCommonRecoBranch......
void FillSRCommonRecoBranchObject(std::shared_ptr<caf::StandardRecord>& sr){
  auto rand = std::make_shared<TRandom3>();
  rand->SetSeed(0);
  size_t ndlp = rand->Integer(10);
  std::vector<caf::SRInteraction>dlp;
  for(int i = 0;i<ndlp;i++)
    dlp.emplace_back(FillSRInteraction(rand.get()));

  size_t npandora = rand->Integer(10);
  std::vector<caf::SRInteraction>pandora;
  for(int i = 0;i<npandora;i++)
    pandora.emplace_back(FillSRInteraction(rand.get()));

  caf::SRInteractionBranch srb;
  srb.ndlp = ndlp;
  srb.dlp = dlp;
  srb.npandora = npandora;
  srb.pandora = pandora;

  sr->common.ixn = srb;
}
```

- Over 1400 objects in one (DUNE) SR object.
- Writing method based on DUNE's method of writing CAF in TTree.

An example of filling an SRCommonRecoBranch. Random variables to be replaced by experiment analysis variables.

Reading CAF Objects



```
void ReadStandardRecord(){
  auto model = RNTupleModel::Create();
  std::shared_ptr<caf::StandardRecord>field_sr;
  field_sr = model->MakeField<caf::StandardRecord>("StandardRecord");

  auto ntuple = RNTupleReader::Open(std::move(model),"NTuple",fname);
  auto nfields = ntuple->GetDescriptor().GetNFields();

  std::cout<<"Number of fields "<<nfields<<std::endl;

  const int entries = ntuple->GetNEntries();
  std::cout<<"Number of Entries "<<entries<<std::endl;
  for(int i=0;i<entries;i++){
    ntuple->LoadEntry(i);
    int nnu = field_sr->mc.nnu;
   // std::cout<<i<<" "<<nnu<<std::endl;
    }
  //ntuple->PrintInfo(ENTupleInfo::kStorageDetails);
  //ntuple->PrintInfo();
}
```

Persistence Type	Size
RNTuple	341 MB
TTree	467 MB

Persistence of 50000 SR entries

RNTupleReader/Writer API Calls

- RNTupleWriter Calls
  - **Recreate** : To create and write ntuple
  - **GetModel** : To access the model
  - **Fill** : To fill entries
  - Other API implementations depending on experiment specific requirements

- RNTupleReader Calls
  - **Open** : To open the ntuple
  - **GetNEntries** : To get total number of entries
  - **LoadEntry** : To load entries
  - Other API implementations depending on experiment specific requirements