RNTuple API Review: Input from ATLAS

Alaettin Serhan Mete[1], Marcin Nowak[2], Peter Van Gemmeren[1]
[1]Argonne National Laboratory, [2]Brookhaven National Laboratory

Weekly CCE-SOP Meeting
3 April 2024

Brookhaven National Laboratory

Argonne NATIONAL LABORATORY

Writing RNTuple in Athena

ROOT RNTuple API Calls

RNTupleContainer.cpp

```
if( mode & pool::CREATE ) {
   m_ntupleWriter = m_rootDb->getNTupleWriter(ntupleName, true);
   if( m_ntupleWriter ) {
      log << DbPrintLvl::Debug << "Created container " << m_name
          << " of type " << ROOTRNTUPLE_StorageType.storageName()
          << DbPrint::endmsg;
   } else {
      log << DbPrintLvl::Error << "Could not create container " << m_name
          << " of type " << ROOTRNTUPLE_StorageType.storageName()
          << DbPrint::endmsg;
      return Error;
   }
   // Prepare Field descriptions
   for( auto& dsc : m_fieldDescs ) {
      log << DbPrintLvl::Debug << "Adding new RNTuple Field: name=" << dsc.fieldname
          << "  typename=" << dsc.typeName() << DbPrint::endmsg;
      m_ntupleWriter->addField( dsc.fieldname, dsc.typeName() );
   }
}
```

RNTupleAuxDynWriter.cxx

```
  /// Definiton of a field info structure
  struct FieldDesc : public DbColumn
  {
    std::string fieldname;
#if ROOT_VERSION_CODE >= ROOT_VERSION( 6, 31, 0 )
    std::unique_ptr< RNTupleView<void, true> > view_p;  // pointer because lack of default xtor
#endif
    std::string sgkey;
    TClass*     clazz = nullptr;
    void*       object = nullptr;
```

```
   /// Add a new field to the RNTuple
   /// Used for data objects from RNTupleContainer, not for dynamic attributes
   void RNTupleAuxDynWriter::addField( const std::string& field_name, const std::string& attr_type )
   {
      if( m_attrDataMap.find(field_name) != m_attrDataMap.end() ) {
         throw std::runtime_error( std::string("Attempt to add existing field.  name: ")
                                   + field_name + "new type: " + attr_type );
      }
      ATH_MSG_DEBUG("Adding new object column, name="<< field_name << " of type " << attr_type);
      auto field = RFieldBase::Create(field_name, attr_type).Unwrap();
      if( !m_model ) {
#if ROOT_VERSION_CODE >= ROOT_VERSION( 6, 31, 0 )
         // first write was already done, need to update the model
         ATH_MSG_DEBUG("Adding late attribute " << field_name);
         auto updater = m_ntupleWriter->CreateModelUpdater();
         updater->BeginUpdate();
         updater->AddField( std::move(field) );
         updater->CommitUpdate();
#endif
      } else {
         m_model->AddField( std::move(field) );
      }
      m_attrDataMap[ field_name ] = nullptr;
   }
```

Disclaimer: Snippets are heavily redacted in most cases
Actual code can be found in the provided links

Writing RNTuple in Athena (cont'd)

RNTupleAuxDynWriter.cxx

RNTupleContainer.cpp

```
DbStatus RNTupleContainer::writeObject( ActionList::value_type& action )
{
   for( auto& dsc : m_fieldDescs ) {

      if( dsc.hasAuxStore() ) {
        num_bytes += m_ntupleWriter->writeAuxAttributes( dsc.fieldname, dsc.getIOStorePtr(), dsc.rows_written );
      }

      m_ntupleWriter->addFieldValue( dsc.fieldname, p.ptr );
      // fill the index field
      m_index = action.link.second;
      m_ntupleWriter->addFieldValue( "index_ref", &m_index );
      m_indexSize++;
   }

   if( !m_ntupleWriter->isGrouped() and m_ntupleWriter->needsCommit() ) {
      num_bytes += m_ntupleWriter->commit();
   }

}
```

```
/// handle writing of dynamic xAOD attributes of an object - called from RootTreeContainer::writeObject()
//  throws exceptions
int RNTupleAuxDynWriter::writeAuxAttributes( const std::string& base_name, SG::IAuxStoreIO* store, size_t /*rows_written*/ )
{
   const SG::auxid_set_t selection = store->getSelectedAuxIDs();
   ATH_MSG_DEBUG("Writing " << base_name << " with " << selection.size() << " Dynamic attributes");
   for(SG::auxid_t id : selection) {
      const std::string attr_type = SG::normalizedTypeinfoName( *store->getIOType(id) );
      const std::string attr_name = SG::AuxTypeRegistry::instance().getName(id);
      const std::string field_name = RootAuxDynIO::auxFieldName( attr_name, base_name );
      void* attr_data ATLAS_THREAD_SAFE = const_cast<void*>( store->getIOData(id) );

      addAttribute( field_name, attr_type, attr_data );
   }
   return 0;  // MN: can get bytes written only when calling Fill() at commit
}
```

```
void RNTupleAuxDynWriter::addAttribute( const std::string& field_name, const std::string& attr_type, void* attr_data )
{
   if( m_attrDataMap.find(field_name) == m_attrDataMap.end() ) {
      addField(field_name, attr_type);
   }
   addFieldValue(field_name, attr_data);
}
```

```
/// Supply data address for a given field
void RNTupleAuxDynWriter::addFieldValue( const std::string& field_name, void* attr_data )
{
   auto field_iter = m_attrDataMap.find(field_name);
   if( field_iter == m_attrDataMap.end() ) {
      std::stringstream msg;
      msg <<"Attempt to write unknown Field with name: '" << field_name << std::ends;
      throw std::runtime_error( msg.str() );
   }
   // already started writing
   field_iter->second = attr_data;
   m_needsCommit = true;
}
```

Writing RNTuple in Athena (cont'd)

RNTupleAuxDynWriter.cxx

RNTupleContainer.cpp

```
DbStatus RNTupleContainer::writeObject( ActionList::value_type& action )
{
   for( auto& dsc : m_fieldDescs ) {

      if( dsc.hasAuxStore() ) {
         num_bytes += m_ntupleWriter->writeAuxAttributes( dsc.fieldname, dsc.getIOStorePtr(), dsc.rows_written );
      }

      m_ntupleWriter->addFieldValue( dsc.fieldname, p.ptr );
      // fill the index field
      m_index = action.link.second;
      m_ntupleWriter->addFieldValue( "index_ref", &m_index );
      m_indexSize++;
   }

   if( !m_ntupleWriter->isGrouped() and m_ntupleWriter->needsCommit() ) {
      num_bytes += m_ntupleWriter->commit();
   }
}
```

```
// write only if there was data added, ignore empty commits
if( !needsCommit() ) {
   ATH_MSG_DEBUG("Empty Commit");
   return 0;
}
ATH_MSG_DEBUG("Commit, row=" << m_rowN << " : " << m_ntupleName );
if( !m_entry ) makeNewEntry();
// update index field before commit

int num_bytes = 0;
ATH_MSG_DEBUG(m_ntupleName << " has " <<  m_attrDataMap.size() << " attributes");
int attrN = 0;
for( auto& attr: m_attrDataMap ) {
   ATH_MSG_VERBOSE("Setting data ptr for field# " << ++attrN << ": " << attr.first << "  data=" << std::hex << attr.second << std::dec );
   // If an object already exists bind it to the field
   // Otherwise, create a new value and use its address
   if( attr.second ) {
      m_entry->BindRawPtr( attr.first, attr.second );
   } else {
      m_entry->EmplaceNewValue( attr.first );
      attr.second = m_entry->GetPtr<void>( attr.first ).get();
   }
}
num_bytes += m_ntupleWriter->Fill( *m_entry );
ATH_MSG_DEBUG("Filled RNTuple Row, bytes written: " << num_bytes);

m_entry.reset();
// forget all values to see if any object is missing in a new commit
for( auto& attr: m_attrDataMap ) { attr.second = nullptr; }
m_needsCommit = false;
m_rowN++;

return num_bytes;
```

```
void  RNTupleAuxDynWriter::makeNewEntry() {
   if( m_model ) {
      // prepare for writing of the first row
      if( !m_tfile ) {
         throw std::runtime_error( std::string("Attempt to write RNTuple ") + m_ntupleName + " without valid TFile ptr" );
      } else {
         // write into existing file
         ATH_MSG_DEBUG("Creating RNTuple " << m_tfile->GetName() << "/" << m_ntupleName);
         m_ntupleWriter = RNTupleWriter::Append(std::move(m_model), m_ntupleName, *m_tfile, m_opts);
         if( m_collectMetrics ) m_ntupleWriter->EnableMetrics();
      }
   }
#if ROOT_VERSION_CODE >= ROOT_VERSION( 6, 31, 0 )
   m_entry = m_ntupleWriter->GetModel().CreateBareEntry();
#else
   m_entry = m_ntupleWriter->GetModel()->CreateBareEntry();
#endif
}
```

Reading RNTuple in Athena

RNTupleContainer.cpp

```
      else if( mode & (pool::READ | pool::UPDATE) ) {
         // create (and keep in the descriptin object) the rntuple field for reading
         m_ntupleReader = m_rootDb->getNTupleReader(ntupleName);
#if ROOT_VERSION_CODE >= ROOT_VERSION( 6, 31, 0 )
         for( auto& dsc : m_fieldDescs ) {
           dsc.view_p = std::make_unique<RNTupleView<void,true>>( m_ntupleReader->GetView<void>(dsc.fieldname, nullptr) );
           if( dsc.hasAuxStore() ) {
              // atach RNTuple Reader (owned by the DB)
              const std::string type_name = dsc.view_p->GetField().GetTypeName();
              dsc.auxdyn_reader = RootAuxDynIO::getNTupleAuxDynReader( dsc.fieldname, type_name, m_ntupleReader );
              // If we set up a reader, then disable aging
              // for this file.  That will prevent POOL from
              // deleting the file while we still have
              // references to its branches.
              dbH.setAge(-10);
           }
         }
#endif
```

RNTupleAuxDynReader.cxx

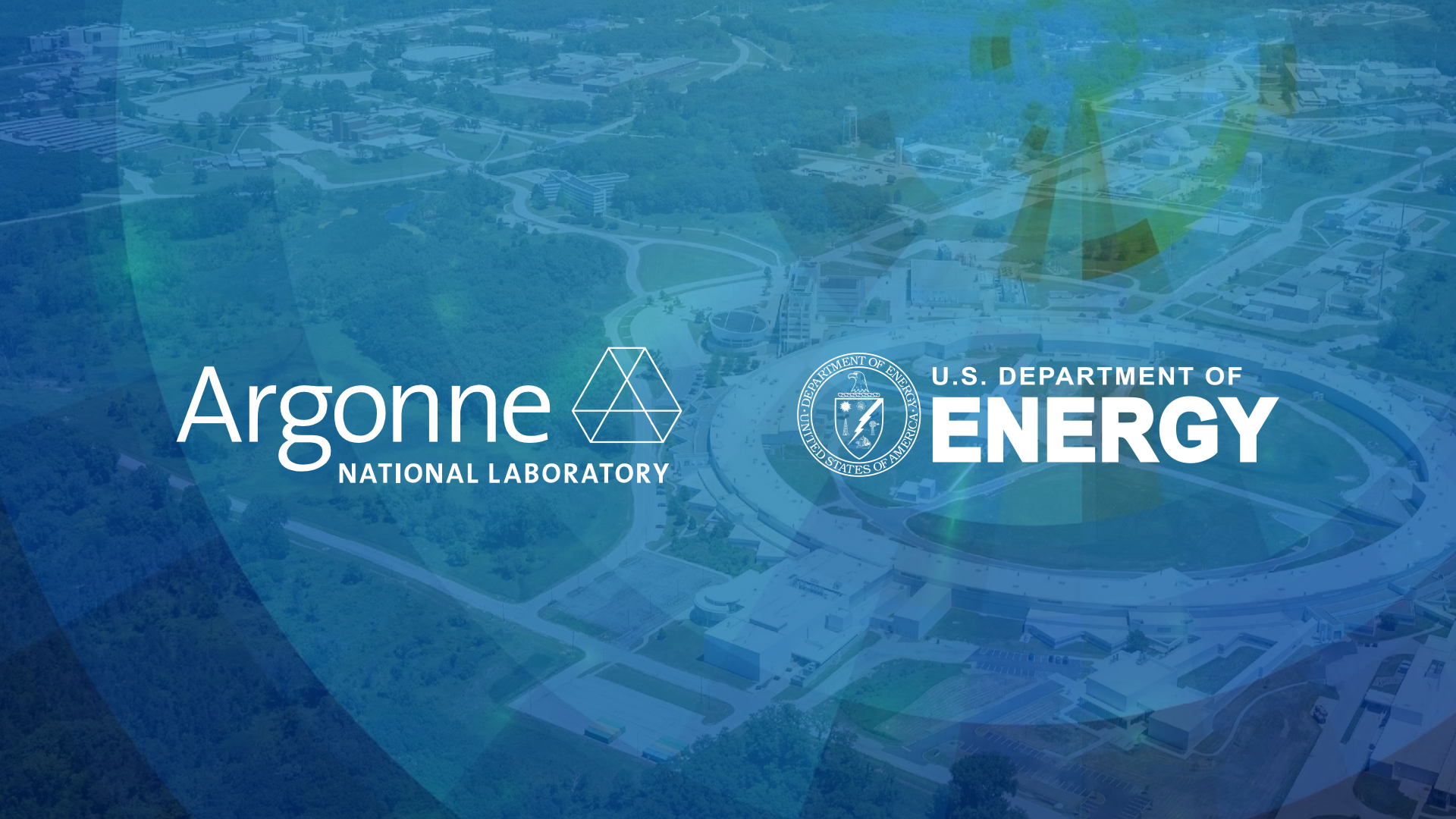
Contains some additional xAOD specific logic as well…

```
DbStatus RNTupleContainer::loadObject(void** obj_p, ShapeH, Token::OID_t& oid)
{
   int64_t evt_id = oid.second;
   if( (evt_id >> 32) > 0 ) {
      evt_id = m_rootDb->indexLookup(m_ntupleReader, evt_id);
   }
   // lock access to this DB for MT safety
   std::lock_guard<std::recursive_mutex>     lock( m_rootDb->ioMutex() );
   try {
      int numBytes = 0;
      for( auto& dsc : m_fieldDescs ) {
        if( !p.ptr ) {
          // create the object for the user and pass ownership to them
          p.ptr = dsc.view_p->GetField().CreateObject<void>().release();
          *obj_p = p.ptr;
        }
        dsc.view_p->BindRawPtr( p.ptr );
        // read into the object
        (*dsc.view_p)(evt_id);

        if (dsc.auxdyn_reader) {
           dsc.auxdyn_reader->addReaderToObject(*obj_p, evt_id, &m_rootDb->ioMutex());
        }
      }
}
```

Direct RNTupleReader/RNTupleWriter API Calls

- **Direct `RNTupleWriter` methods called in Athena:**
  - `Append` : To create an instance in an existing `TFile`
  - `GetModel` : To access the underlying model
  - `CreateModelUpdater` : To accommodate late model extensions
  - `Fill` : To fill the `RNTuple` with an `REntry`
  - `EnableMetrics` : To enable I/O metric collection
  - `GetMetrics` : To access/print the I/O metrics
- **Direct `RNTupleReader` methods called in Athena:**
  - `Open` : To create an instance
  - `GetDescriptor` : To access the cached descriptor
  - `GetModel` : To access the underlying model
  - `GetNEntries` : To read the total number of entries
  - `GetEntryRange` : To read through the entries
  - `GetView` : To read specific fields
  - `EnableMetrics` : To enable I/O metric collection
  - `GetMetrics` : To access/print the I/O metrics


Argonne
NATIONAL LABORATORY
U.S. DEPARTMENT OF
ENERGY

Examples of Various Other API Calls

Looping over top-level fields

```
void RNTupleAuxDynReader::init( bool standalone )
{
   if( m_initialized )  return;

   const SG::AuxTypeRegistry& reg = SG::AuxTypeRegistry::instance();
   const string field_prefix = m_storeFieldName + '.';
   const auto& desc = m_ntupleReader->GetDescriptor();
#if ROOT_VERSION_CODE >= ROOT_VERSION( 6, 31, 0 )
   for( const auto &f : desc.GetTopLevelFields() ) {
#else
   for( const auto &f : desc->GetTopLevelFields() ) {
#endif
      const string field_name = f.GetFieldName();
      if( field_name.rfind(field_prefix,0) == 0 ) {
         const string attr_infile = field_name.substr(field_prefix.size());
         const string attr_name = reg.inputRename(m_key, attr_infile);
         const string field_type = f.GetTypeName();

         SG::auxid_t auxid = getAuxIdForAttribute(attr_name, field_type, standalone);
         // add AuxID to the list
         // May still be null if we don't have a dictionary for this field
         if( auxid != SG::null_auxid ) {
            m_auxids.insert(auxid);
            m_fieldInfos[auxid].fieldName = field_name;
#if ROOT_VERSION_CODE >= ROOT_VERSION( 6, 31, 0 )
            m_fieldInfos[auxid].view_p = std::make_unique<RNTupleView<void,true>>(
               m_ntupleReader->GetView<void>(field_name, nullptr) );
#endif
         } else {
            errorcheck::ReportMessage msg (MSG::WARNING, ERRORCHECK_ARGS, "RNTupleAuxDynReader::init");
            msg << "Could not find auxid for " << attr_infile << " type: " << field_type
                << "  standalone=" << standalone;
         }
      }
   }
   m_initialized = true;
}
```

Collecting/Printing Metrics

```
// Print metrics if enabled - this can become DEBUG/VERBOSE
if( m_ntupleWriter->GetMetrics().IsEnabled() ) {
  auto& log = msg(MSG::INFO);
  log << "Printing I/O Statistics\n";
  m_ntupleWriter->GetMetrics().Print(log.stream());
  log << endmsg;
}
```

File Peeking (python)

```
# Get the number of entries in the file
pool_cont_token = re.compile(f'{root.APRDefaults.TTreeNames.DataHeader}(?!Form)|{root.APRDefaults.RNTupleNames.DataHeader}').match

for key in f.GetListOfKeys():
    key_name = key.GetName()
    match  = pool_cont_token(key_name)
    if not match:
        continue
    obj = f.Get(key_name)
    if isinstance(obj, root.TTree):
        nentries = obj.GetEntriesFast()
        break
    elif isinstance(obj, root.Experimental.RNTuple):
        reader = root.Experimental.RNTupleReader.Open(obj)
        nentries = reader.GetNEntries()
        break
```

Writing RNTuple vs TTree in Athena

RNTupleContainer.cpp

```
DbStatus RNTupleContainer::writeObject( ActionList::value_type& action )
{
   for( auto& dsc : m_fieldDescs ) {

       if( dsc.hasAuxStore() ) {
          num_bytes += m_ntupleWriter->writeAuxAttributes( dsc.fieldname, dsc.getIOStorePtr(), dsc.rows_written );
       }

      m_ntupleWriter->addFieldValue( dsc.fieldname, p.ptr );
      // fill the index field
      m_index = action.link.second;
      m_ntupleWriter->addFieldValue( "index_ref", &m_index );
      m_indexSize++;
   }

   if( !m_ntupleWriter->isGrouped() and m_ntupleWriter->needsCommit() ) {
      num_bytes += m_ntupleWriter->commit();
   }

}
```

RootTreeContainer.cpp

```
DbStatus RootTreeContainer::writeObject( ActionList::value_type& action )
{

    for(k=m_branches.begin(), icol=0; k !=m_branches.end(); ++k, ++icol) {

         if( dsc.auxdyn_writer ) {
            num_bytes += dsc.auxdyn_writer->writeAuxAttributes
               ( dsc.branch->GetName(), dsc.getIOStorePtr(), dsc.rows_written );
            aux_needs_fill = aux_needs_fill || dsc.auxdyn_writer->needsCommit();
         }

         dsc.branch->SetAddress(p.ptr);

        if( isBranchContainer() && !m_treeFillMode ) {
           num_bytes += dsc.branch->Fill();
        }
     }

      if( !isBranchContainer() ) {
         // Single Container per TTree - just Fill it now
         num_bytes = m_tree->Fill();
      } else
}
```

Writing RNTuple vs TTree in Athena (cont'd)

RNTupleAuxDynWriter.cxx

```
/// handle writing of dynamic xAOD attributes of an object - called from RootTreeContainer::writeObject()
//  throws exceptions
int RNTupleAuxDynWriter::writeAuxAttributes( const std::string& base_name, SG::IAuxStoreIO* store, size_t /*rows_written*/ )
{
   const SG::auxid_set_t selection = store->getSelectedAuxIDs();
   ATH_MSG_DEBUG("Writing " << base_name << " with " << selection.size() << " Dynamic attributes");
   for(SG::auxid_t id : selection) {
      const std::string attr_type = SG::normalizedTypeinfoName( *store->getIOType(id) );
      const std::string attr_name = SG::AuxTypeRegistry::instance().getName(id);
      const std::string field_name = RootAuxDynIO::auxFieldName( attr_name, base_name );
      void* attr_data ATLAS_THREAD_SAFE = const_cast<void*>( store->getIOData(id) );

      addAttribute( field_name, attr_type, attr_data );
   }
   return 0;  // MN: can get bytes written only when calling Fill() at commit
}
```

```
void RNTupleAuxDynWriter::addAttribute( const std::string& field_name, const std::string& attr_type, void* attr_data )
{
   if( m_attrDataMap.find(field_name) == m_attrDataMap.end() ) {
      addField(field_name, attr_type);
   }
   addFieldValue(field_name, attr_data);
}
```

```
/// Supply data address for a given field
void RNTupleAuxDynWriter::addFieldValue( const std::string& field_name, void* attr_data )
{
   auto field_iter = m_attrDataMap.find(field_name);
   if( field_iter == m_attrDataMap.end() ) {
      std::stringstream msg;
      msg <<"Attempt to write unknown Field with name: '" << field_name << std::ends;
      throw std::runtime_error( msg.str() );
   }
   // already started writing
   field_iter->second = attr_data;
   m_needsCommit = true;
}
```

```
/// handle writing of dynamic xAOD attributes of an object - called from RootTreeContainer::writeObject()
//  throws exceptions
int TBranchAuxDynWriter::writeAuxAttributes( const std::string& base_branchname,
                                             SG::IAuxStoreIO *store,
                                             size_t backfill_nrows )
{
   int bytes_written = 0;
   const SG::auxid_set_t selection = store->getSelectedAuxIDs();
   ATH_MSG_DEBUG("Writing " << base_branchname << " with " << selection.size() << " Dynamic attributes");

   // mark all attributes as not written yet
   for( auto& aux_info_entry : m_auxInfoMap )  aux_info_entry.second.written = false;
   m_needsFill = false;

   for(SG::auxid_t id : selection) {
      AuxInfo& attrInfo = m_auxInfoMap[id];
      if( !attrInfo.branch ) {
         // new attribute info, fill it
         attrInfo.typeinfo = store->getIOType(id);
         attrInfo.type_name = SG::normalizedTypeinfoName( *attrInfo.typeinfo );
         attrInfo.name = SG::AuxTypeRegistry::instance().getName(id);
         attrInfo.branch_name = RootAuxDynIO::auxBranchName(attrInfo.name, base_branchname);
         createAuxBranch( attrInfo );
         // backfill here
         if( backfill_nrows ) {
            // if this is not the first row, catch up with the rows written already to other branches
            ATH_MSG_DEBUG("  Backfilling " << backfill_nrows << " entries for " << attrInfo.name);
            // As of root 6.22, calling SetAddress with nullptr may not work as expected if the address had
            // previously been set to something non-null.
            // So we need to create the temp object ourselves.
            attrInfo.setDummyAddr();
            for( size_t r=0; r<backfill_nrows; ++r ) {
               bytes_written += attrInfo.branch->BackFill();
               ATH_MSG_VERBOSE("BACKFilled branch:" << m_ttree->GetName() << "::" << attrInfo.branch->GetName() <<
                               " Tree size=" << m_ttree->GetEntries()
                               << "  branch size:" << attrInfo.branch->GetEntries() );
            }
         }
      }
      void *obj ATLAS_THREAD_SAFE = const_cast<void*>( store->getIOData(id) );
      attrInfo.object = obj;
      attrInfo.branch->SetAddress( attrInfo.objectAddr() );
      attrInfo.written = true;
   }

   for( auto& aux_info_entry : m_auxInfoMap ) {
      AuxInfo& attrInfo = aux_info_entry.second;
      if( m_branchFillMode ) {
         // cout << "MN: BranchFill for " << attrInfo.branch_name << endl;
         bytes_written += attrInfo.branch->Fill();
      } else {
         m_needsFill = true;
      }
   }
}
```

TBranchAuxDynWriter.cxx

Reading RNTuple vs TTree in Athena

RNTupleContainer.cpp

```
DbStatus RNTupleContainer::loadObject(void** obj_p, ShapeH, Token::OID_t& oid)
{
   int64_t evt_id = oid.second;
   if( (evt_id >> 32) > 0 ) {
      evt_id = m_rootDb->indexLookup(m_ntupleReader, evt_id);
   }
   // lock access to this DB for MT safety
   std::lock_guard<std::recursive_mutex>     lock( m_rootDb->ioMutex() );
   try {
      int numBytes = 0;
      for( auto& dsc : m_fieldDescs ) {
         if( !p.ptr ) {
            // create the object for the user and pass ownership to them
            p.ptr = dsc.view_p->GetField().CreateObject<void>().release();
            *obj_p = p.ptr;
         }
         dsc.view_p->BindRawPtr( p.ptr );
         // read into the object
         (*dsc.view_p)(evt_id);

         if (dsc.auxdyn_reader) {
            dsc.auxdyn_reader->addReaderToObject(*obj_p, evt_id, &m_rootDb->ioMutex());
         }
      }
}
```

RootTreeContainer.cpp

```
DbStatus
RootTreeContainer::loadObject(void** obj_p, ShapeH /*shape*/, Token::OID_t& oid)
{
  auto evt_id = oid.second;
  if( evt_id < 0 || (uint64_t)evt_id >= size() ) {
     // -1 may be from a failed index lookup
     *obj_p = nullptr;
     // do not return Error to avoid error printouts in case someone just tries to iterate over all OIDs
     return Success;
  }
  // lock access to this DB for MT safety
  std::lock_guard<std::recursive_mutex>     lock( m_rootDb->ioMutex() );
  try {
     int numBytesBranch, numBytes = 0;
     bool hasRead(false);
     for( auto& dsc : m_branches ) {
        RootDataPtr p(nullptr), q(nullptr);
        int typ = dsc.column->typeID();
        dsc.branch->SetAddress(p.ptr);
        // read the object
        numBytesBranch = dsc.branch->GetEntry(evt_id);
        TTree::TClusterIterator clusterIterator = dsc.branch->GetTree()->GetClusterIterator(evt_id);
        clusterIterator.Next();
        if (evt_id == clusterIterator.GetStartEntry() && dsc.branch->GetTree()->GetMaxVirtualSize() != 0) {
           for (int i = dsc.branch->GetReadBasket(); i < dsc.branch->GetMaxBaskets()
                   && dsc.branch->GetBasketEntry()[i] < clusterIterator.GetNextEntry(); i++) {
              dsc.branch->GetBasket(i);
           }
        }
     }
}
```

TBranchAuxDynReader.cxx

Contains some additional xAOD specific logic as well…