(First look at) Metadata for RNTuple and ATLAS use-case

Maciej Szymański

Argonne National Laboratory

with the help of
Peter Van Gemmeren (ANL), Alaettin Serhan Mete (ANL),
Marcin Nowak (BNL), Attila Krasznahorkay (CERN) et al.

RNTuple Format and Feature Assessment, CERN, 6th Nov 2023

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

Intro

- Revisiting the storage technology used for the in–file metadata
  - in the context of migration to `RNTuple` / `ROOT7` of ATLAS event data
- Presenting here the use case of ATLAS in–file metadata
  - with the emphasis on `I/O`–related bits
- The goal is to find a solution to store metadata objects for Run 4 and beyond
  - *appropriate* and *performant*
- We may want to rethink the approach entirely
  - find the equivalent *standard* of storing HEP event data in `ROOT` `TTree` / `RNTuple`
  - e.g. consider concept of a *file* in the context of using object stores
- Would appreciate any feedback and guidance!
  - possibly looking for cross-experiment synergies

ATLAS in–file metadata

Information (non-event data) in event data files about the events in that file

- examples: run/simulation parameters, event summary
- *note*: metadata in central DBs is the independent development (framework does not need to write it)

Infrastructure within framework to read from input, propagate, and write to output

- propagating may also involve some processing (after opening new input file)
- creating metadata if requested, but not available (after each event processed)

Challenges:

- merging is not trivial
- supporting event-less files
- handling in multi-threading / multi-processing jobs

Use cases

- Configuration of the job using info from the input files
- Initialisation of software components
- Decoding trigger information
- Keeping track of event selection and luminosity blocks
- Annotations added by users

Essential for all workflows!

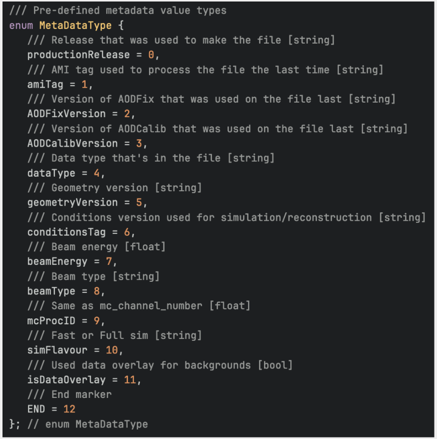
- including (but not limited to): reconstruction, simulation, derivation, analysis

(Non-exhaustive) list of metadata categories stored in files

Metadata category	Content
`FileMetaData`	event and provenance summary
`EventStreamInfo`	summary of the event content for production
`EventFormat`	summary of the event content for analysis
`ByteStream`	run parameters
`BookKeeping`	event selections, cuts
`LumiBlocks`	luminosity blocks stored in file
`TriggerMenu`	trigger configuration
`Truth`	MC weights, generator details

FileMetaData example

- Key–value store containing a summary of properties shared by all events in the file
- Built using the first set of values encountered in a job
- Values assumed to be constant throughout the job → `FileMetaData` objects cannot be directly merged
- It's convenient to use by analysts, outside of the framework
- Moving here more information is considered
- We could use `FileMetaData` as a testbed for a new storage technology



```
/// Pre-defined metadata value types
enum MetaDataType {
   /// Release that was used to make the file [string]
   productionRelease = 0,
   /// AMI tag used to process the file the last time [string]
   amiTag = 1,
   /// Version of AODFix that was used on the file last [string]
   AODFixVersion = 2,
   /// Version of AODCalib that was used on the file last [string]
   AODCalibVersion = 3,
   /// Data type that's in the file [string]
   dataType = 4,
   /// Geometry version [string]
   geometryVersion = 5,
   /// Conditions version used for simulation/reconstruction [string]
   conditionsTag = 6,
   /// Beam energy [float]
   beamEnergy = 7,
   /// Beam type [string]
   beamType = 8,
   /// Same as mc_channel_number [float]
   mcProcID = 9,
   /// Fast or Full sim [string]
   simFlavour = 10,
   /// Used data overlay for backgrounds [bool]
   isDataOverlay = 11,
   /// End marker
   END = 12
}; // enum MetaDataType
```

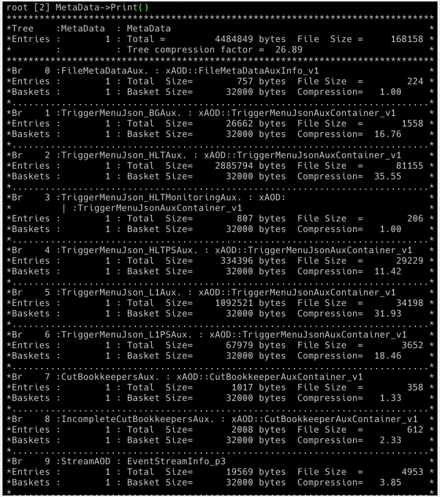
Current design of I/O layer

In-file metadata is stored in a dedicated `TTree` within a single entry

- separate branch is added if more objects of the same type needed
- I/O infrastructure shared with event data

Information grouped in categories represented by classes aggregating fields of:

- simple types (POD and `std::string`)
- containers (`std::vector`) of simple types
- (rare) cases of `std::set`, `std::map`, and nested vectors



```
root [2] MetaData->Print()
******************************************************************************
*Tree    :MetaData  : MetaData                                               *
*Entries :        1 : Total =         4484849 bytes  File  Size =     168158 *
*        :          : Tree compression factor =  26.89                       *
******************************************************************************
*Br    0 :FileMetaDataAux. : xAOD::FileMetaDataAuxInfo_v1                    *
*Entries :        1 : Total  Size=        757 bytes  File Size  =        224 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    1 :TriggerMenuJson_BGAux. : xAOD::TriggerMenuJsonAuxContainer_v1      *
*Entries :        1 : Total  Size=      26662 bytes  File Size  =       1558 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  16.76     *
*............................................................................*
*Br    2 :TriggerMenuJson_HLTAux. : xAOD::TriggerMenuJsonAuxContainer_v1     *
*Entries :        1 : Total  Size=    2885794 bytes  File Size  =      81155 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  35.55     *
*............................................................................*
*Br    3 :TriggerMenuJson_HLTMonitoringAux. : xAOD:                          *
*         | :TriggerMenuJsonAuxContainer_v1                                  *
*Entries :        1 : Total  Size=        807 bytes  File Size  =        206 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    4 :TriggerMenuJson_HLTPSAux. : xAOD::TriggerMenuJsonAuxContainer_v1   *
*Entries :        1 : Total  Size=     334396 bytes  File Size  =      29229 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  11.42     *
*............................................................................*
*Br    5 :TriggerMenuJson_L1Aux. : xAOD::TriggerMenuJsonAuxContainer_v1      *
*Entries :        1 : Total  Size=    1092521 bytes  File Size  =      34198 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  31.93     *
*............................................................................*
*Br    6 :TriggerMenuJson_L1PSAux. : xAOD::TriggerMenuJsonAuxContainer_v1    *
*Entries :        1 : Total  Size=      67979 bytes  File Size  =       3652 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  18.46     *
*............................................................................*
*Br    7 :CutBookkeepersAux. : xAOD::CutBookkeeperAuxContainer_v1            *
*Entries :        1 : Total  Size=       1017 bytes  File Size  =        358 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.33     *
*............................................................................*
*Br    8 :IncompleteCutBookkeepersAux. : xAOD::CutBookkeeperAuxContainer_v1  *
*Entries :        1 : Total  Size=       2008 bytes  File Size  =        612 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.33     *
*............................................................................*
*Br    9 :StreamAOD : EventStreamInfo_p3                                     *
*Entries :        1 : Total  Size=      19569 bytes  File Size  =       4953 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   3.85     *
*............................................................................*
```

Considerations relevant for the choice of storage technology

Size and `I/O` speed are not a concern, being much smaller than event data

- e.g. $\sim$ 160 kB in $\sim$ 220 MB example DAOD file from Run 3

Merging of metadata objects is not straightforward

- is *not* a simple appending in a general case, as it depends on:
  - metadata type
    - e.g. semantically it's incorrect to accumulate values assumed to be unique
  - knowledge if the input file was fully processed
    - important for propagating e.g. luminosity information
- needs some logic, perhaps implemented through a callback function?
  - merging rules depending on a metadata type, workflow, status of event processing
- as of now, it's a responsibility of *metadata tools* within the framework

More constraints for the storage

- Ease of use outside of `Athena` framework important for the analysis use case
- Fast file-peeking w/o framework would also be beneficial
  - frequent, small reads of metadata info needed to configure the job
- Metadata *attached* to event `RNTuple` need to be readable even w/o events
  - in multi-processing mode, workers may not process any events
  - case of skimmed data
- Schema evolution may be needed

Moving forward

- It seems that `TKey` / `TDirectory` approach would have been more suitable for storing metadata than single-entry `TTree`
  - infrastructure implemented in `Athena` years ago, but never really used
- Consequently, we may not want to necessarily store metadata in separate `RNTuple` (with single entry)
- Essentially, we need some kind of key–value store of custom objects attached to a corresponding event store (`RNTuple`)
  - `TKey` / `TDirectory` (`RDirectory`) approach ?
  - built-in utilities `RNTupleInspector` / `RNTupleDescriptor` ?
  - user-defined metadata envelope in `RNTuple` ?
- How to store the metadata when corresponding event data in `RNTuple` persistified in the object storage?

Backup

Metadata infrastructure in ATLAS

- Using `Gaudi`/`Athena` components
- `MetaDataSvc` orchestrates metadata tools through file incidents
  - ▶ in general thread-unsafe, but *ok* if minimised and used carefully
- Domain–specific metadata tools
  - ▶ propagate at the beginning of reading the input file
  - ▶ create after each event processed
- (Transient) object stores make metadata available to clients
  - ▶ `InputMetaDataStore`
    - filled with new file content
    - cleared when moving to the next one
  - ▶ `MetaDataStore`
    - gets new content for the output metadata containers