HEP-CCE/SOP: RNTuple API Review midterm report

Reviewers, Experts

ATLAS: Marcin Mowak, Serhan Mete, Peter van Gemmeren

CMS: Chris Jones, Matti Kortelainen, Dan Riley

CAF: Amit Bashyal

DUNE: Barnali Chowdhury

CCE: Saba Sehrish, Philippe Canal

Introduction

ROOT is developing a replacement for TTree called RNTuple. TTree has served the HEP community for decades and several Exa-Bytes of data are currently stored in TTree. TTree is expected to remain part of ROOT and will be readable and writable for the indefinite future. However, for performance and functionality enhancements, the ROOT team will focus on RNTuple which is based on more modern design and promises significant advantages for storage efficiency and I/O speed. To ensure the new RNTuple can serve the needs of HEP experiments, the ROOT team asked the HEP-CCE/SOP project, which includes US funded experts on experiment data processing software frameworks for ATLAS, CMS and (to a lesser degree) DUNE to conduct a review of the RNTuple API (RNTuple Format and Feature Assessment (6-7 November 2023) · Indico (cern.ch)).

Review process

The review process was started with Special CCE-SOP tele-conference: RNTuple API Review Kick Off (February 28, 2024) · INDICO-FNAL (Indico) where experts of the ROOT team presented the RNTuple developments and the scope of the proposed review.

The HEP-CCE/SOP team had three meetings dedicated to different components of RNTuple used by the experiments. These meetings did include some experts from outside HEP-CCE. Prior to the meeting, team members from ATLAS and CMS prepared short overviews on their usage and requirements for RNTuple modules. HEP-CCE included experts for using RNTuple for CAF (Common Analysis Format) data used by neutrino experiments, including DUNE, for analysis data products.

Special CCE-SOP tele-conference: RNTuple API Review Reader/Writer (April 3, 2024) · INDICO-FNAL (Indico)

Special CCE-SOP tele-conference: RNTuple API Review Model/Field/Entry (May 1, 2024) · INDICO-FNAL (Indico)

Special CCE-SOP tele-conference: RNTuple API Review Remaining modules (June 26, 2024) · INDICO-FNAL (Indico)

RNTuple API Review Reader/Writer

ATLAS, CMS and CAF maker are accessing RNTuple via RNTupleWriter and RNTupleReader API's. For RNTupleWriter (not all API's are used by all experiments):

```
Append : To create an instance in an existing TFile, a
non-default RNTupleWriteOptions is passed where all possible
options may be set
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
CreateEntry : To create an REntry that is then passed to
Fill
```

And for RNTupleReader:

```
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
LoadEntry : To read all fields when it is useful, e.g. for
framework metadata RNTuple (not used for event data)
```

The overall API functionality was found to be complete for the experiments' frameworks. Object ownership models provided by ROOT RNTuple are sufficient for the experiments' needs. Error handling is done via exceptions. Some possible improvements were found and are documented in the Outcome section.

RNTuple API Review Model/View/Field/Entry

Experiments' frameworks use the following interfaces for RNTupleModel:

```
Create : To create empty model
CreateBare : To create model with minimal memory
AddField : Add field, before first write
CreateBareEntry : Creates entry for each write with minimal
memory
CreateEntry : Creates entry for each write
GetToken : To get a token to allow a fast reference to a
field
GetFieldZero().GetSubFields : To loop over all top-level
fields of a model
```

ATLAS is also using the RNTupleUpdater for late model extensions.

For RNTupleView the experiments are using

```
GetField: Release object for a field
BindRawPtr: Bind object pointer to field
operator(): Fills the view for a given entry
GetValue().GetPtr : Gets the underlying value that was
filled (called only when the bool template argument is
false)
```

One finding here is that `RNTupleView<T, bool>` could be separated into two distinct classes for the two values of bool, e.g. `RNTupleView<T>` and `RNTupleUserOwnedView<T>`.

For RNTupleField:

```
Create : To create base field
BindRawPtr: Bind Raw Pointer to field, before first write
GetSubFields: To loop over the subfields of the FieldZero
GetFieldName: To get the name of the subfields
```

```
GetTypeName: To get the type of the subfields
```

And RNTupleEntry:

```
BindRawPtr: Bind Raw Pointer to entry
EmplaceNewValue: Fill entry with new value for each write
GetPtr<void>: To get the address for a type unknown at
compile time
GetPtr<T>: To get the address for a type known at compile
time
```

One consideration here is that for reading REntry is created with RNTupleModel, but for writing REntry is created with RNTupleWriter. We would suggest being consistent between the two.

The overall API functionality was found to be complete for the experiments' frameworks. Object ownership models provided by ROOT RNTuple are sufficient for the experiments' needs. Error handling is done via exceptions. Some possible improvements were found and are documented in the Outcome section.

RNTuple API Review remaining modules

At this point neither experiment framework is using lower-level API's

Early Review Outcome and Requests

While reviewing the above RNTuple API components, we found no fundamental problem preventing experiments' frameworks from using RNTuple for persistence to replace TTree. In fact, both ATLAS and CMS have prototypes to store their event data in RNTuple. And HEP-CCE developed a prototype for storing neutrino experiment CAF data in RNTuple. Nevertheless, we would like to ask for some extensions:

1. Need mechanism to customize page size for particular fields: ATLAS sees that few/certain data objects are too large to achieve good compression with standard page size (cases where TTree optimized baskets to be >~ 1 MB) and storage sizes go up significantly. If not addressed this can offset a large fraction of storage savings from RNTuple or even lead to an overall increase. Therefore, we would like the capability to overwrite/increase the default page size for some fields.

2. RClusterPool: Based on past experience with TTree I/O, the experiments desire configurability similar to TTreeCache.
3. Indexing capability similar to TTree to associate RNTuple to another instance with sparse entries (not required for current workflows, do have framework work-around).
4. The experiments need an ability to tune the memory usage in RNTupleWriter
5. CMS foresees to not to be able to use RNTupleParallelWriter as long as it has the restriction of having only one Writer per file (CMS presently stores several TTree objects in one file, and foresees to do the same with RNTuple)
6. RNTupleParallelWriter: for every writer's Fill() call, CMS will need to know what entry number that Fill() call corresponds to in the RNTuple. It is sufficient to know that mapping periodically when the data is being written (whereas waiting until the file close time is known to cause high memory usage).
7. When a user owns the memory to be filled from storage in any other way than via std::shared_ptr, the user has to pass a dummy std::shared_ptr<void> to RNTupleReader::GetView<void>(), after which the user has to call the RNTuple::BindRawPtr() to set the actual memory address. CMS would prefer a more direct interface that would avoid the dummy std::shared_ptr<void>.
8. Having to have two separate loops to call RNTupleModel::AddField() and RNTupleModel::GetToken() feels suboptimal (AddField() requires the model to be unfrozen, and GetToken() requires the model to be frozen).
9. REntry::RFieldToken not having a default constructor is somewhat inconvenient, even if it can be worked around with std::optional for use cases such as member data.

HEP-CCE2 SOP looked at the persistence of CAF (analysis format data of the DUNE experiment) objects in RNTuple on behalf of DUNE. Persistence of CAF objects in RNTuple does not seem to be an additional API requirement. We also looked at the I/O tools (specially Proxy writers) that write CAF objects as flat ntuples and the existing RNTuple API seems to be adequate to rewrite the Proxy writers using the RNTuple API.

The final phase of the review will focus on aspects such as error reporting and propagation, documentation and initial performance assessments. An early finding for documentation is, clarification would be helpful in these areas:

1. Relationship between RNTupleModel's GetToken() and freezing
    a. When exactly does the model become frozen or unfrozen?
    b. When can GetToken() be called safely?
    c. Are users allowed to call Freeze() and/or Unfreeze()?
2. Projected fields
    a. Are projected fields a property of the on-disk RNTupleModel?
    b. Or can they be created on the fly?
3. More explanation on "entry invalidation" when an RNtupleModel is extended.

We expect the review to be complete by October 2024 and plan to issue an updated report.