RNTuple API Review – Discussion of Midterm Findings

Jakob Blomer for the ROOT Team, CERN (EP-SFT)

HEP-CCE2/SOP

24 September 2024

Plans and Follow-Ups I

- `RNTupleView`
  - The class will be changed to behave like an `REntry` with a single field
  - Thus, the question of owning or non-owning storage will become a runtime decision
  - Tracked as issue #16321
- `REntry` use in `RNTupleReader`, `RNTupleWriter`
  - Intention for the reader and writer API wrt. REntry handling is to be symmetric
  - Both `RNTupleReader::LoadEntry()` and `RNTupleWriter::Fill()` take an optional `REntry` argument.
  - If not provided, they use the default entry of the `RNTupleModel`
  - One difference between reading and writing is that the model reconstructed from the file always has a default entry, even if not used. This will be fixed, tracked as #16324.

Plans and Follow-Ups II

- Page Size Tuning & Memory Consumption on Write
  - Addressed by a new, adaptive algorithm to set page sizes (merged).
  - The new algorithm grows the pages as needed, so that dense columns get large pages and sparse columns small ones.
  - Pages still have an absolute limit (default 1MB) and the overall memory budget used for page buffers is limited.
  - Good first results on CMS MiniAOD (smaller files than TTree, memory overhead wrt. TTree halfed); still room for memory improvement
- Flexible Control of `RClusterPool`
  - Tracked as issue ▸ #16325

Plans and Follow-Ups III

- Indexing
  - Larger scope; work on it has started.
  - A new class, the `RNTupleProcessor` implements iterations of non-trivial joins of RNTuples (in contrast to simple/single RNTuple iteration of the `RNTupleReader`
  - Initial version of the `RNTupleProcessor` and indexing capabilities merged.
  - Full functionality expected in 2025

Plans and Follow-Ups IV

- `RNTupleParallelWriter`
  - Clear guarantees about the locking around `TFile`
  - New method “FillNoCommit()” allows framework to control time of `TFile` access
  - New staged cluster committing allows to set the logical cluster ordering after flushing; facilitates “data barriers” such as lumi block separation
- We will implement the minor suggestions for API improvement (points 7–9)

Discussion of Open Questions

- `RNTupleModel` & `GetToken()`
  - The frozen state can be explicitly set by the user through `Freeze()` and `Unfreeze()` APIs. Both calls are idempotent.
  - Users can call `Freeze()` and `Unfreeze()`. Note that unfreezing a model will change the model id. As a result, after refreezing, existing `REntries` cannot be used anymore for reading and writing.
  - The model is implicitly frozen when passed to the `RNTupleWriter` / `RNTupleReader` and on committing a changeset for the late model extension (`RNTupleModel::RUpdater::CommitUpdate()`)
  - The model is implicitly unfrozen at the beginning of the `RNTupleUpdater` (`RNTupleModel::RUpdater::BeginUpdate()`).
  - `GetToken()` can be called on any frozen model. This will probably change such that tokens can also be created while constructing a model.
  - Note that currently tokens cannot be applied to clones of models. This will be fixed (▸ #16326).

Discussion of Open Questions

- Projected Fields
  - Field projections are stored as projections on-disk.
  - When reading, the user can decide whether the model reconstructed from disk should treat projections as projections, or present them as if they were physical fields (see `RCreateModelOptions`)
  - Note that models with projected fields cannot be used for the `RNTupleReader` (but, e.g., as a source for cloned model for skimming). The restriction on the `RNTupleReader` can be lifted if needed.
- Late Model extension
  - Late model extension will unfreeze the model at the beginning of the transaction and (re-)freeze the model when the extension is committed.
  - As a result, the model ID will change.
  - All existing `REntry` objects and tokens created from the model cannot be used anymore but new entries and tokens need to be retrieved.

Timeline and Next Steps

- Most of the points will be addressed this year
- Improvements to the `RClusterPool` may overflow into next year
- The work on indexing and the `RNTupleProcessor` will most likely conclude only in 2025

In terms of ROOT releases

- Target for the RNTuple 1.0 binary format is 6.34 (November):
  - Backwards-compatibility for data written in this format
  - We will *break* backwards compatibility for experimental RNTuple format versions (clean slate)
- Target for moving the reviewed set of classes out of experimental: ROOT 6.36 (H1/2025)

Many thanks for the thorough and useful feedback!