CMS feedback on `RNTupleModel`, `RField`, and `REntry`

Chris Jones, Matti Kortelainen, Dan Riley

HEP-CCE SOP meeting

1 May 2024

High level

- Interaction between `REntry::BindRawPtr()`, `RField`, and `RNTupleModel` feels suboptimal when using the `REntry::RFieldToken`
  - Presently
    - Create a field: `field = ROOT::Experimental::RFieldBase::Create(name, …)`
    - Add field to a model: `model->AddField(field)`
    - Get the token from the model: `token = model->GetToken(name)`
    - `REntry::BindRawPtr(token, ptr)`
  - Option that would feel to be simpler to use
    - Create a field: `field = ROOT::Experimental::RFieldBase::Create(name, …)`
    - Add field to a model and get token: `token = model->AddField(field)`
    - `REntry::BindRawPtr(token, ptr)`
  - `RNTupleModel::GetToken()` still makes sense for other purposes

High level

- Interaction between `REntry::BindRawPtr()`, `RField`, and `RNTupleModel` feels suboptimal when using the `REntry::RFieldToken`
  - However, we noticed `RNTupleModel::AddField()` requires the model to be unfrozen, and `RNTupleModel::GetToken()` requires the model to be frozen
    - Unfortunate, can't call `AddField()` and `GetToken()` in the same loop
    - Instead, have to do
      - Create the `RNTupleModel`
      - In one loop, add the `RFields`, need to keep the field names in a separate vector
      - Move the model to `RNTupleWriter/Reader`
      - Get a reference to the `RNTupleModel`
      - Loop over the field names and get the `RFieldTokens` from the model

`RNTupleModel`

- We would like to have more description on `RNTupleModel` being frozen
  - When exactly does the model become frozen or unfrozen?
  - When can `RNTupleModel::GetToken()` be called safely?
  - Are users allowed to call `RNTupleModel::Freeze()` / `Unfreeze()`?
- Are *projected fields* a property of the on-disk `RNTupleModel`? Or can they be created on the fly?
  - We would like to have more explanation of projected fields in general

`RNTupleModel` "late schema extension"

- We became concerned on thread safety of the late schema extension
  - The sequential write case, i.e. `RNTupleWriter::CreateModelUpdator()`, should be fine, because framework needs to synchronize at that point anyhow
  - With `RNTupleParallelWriter` things things seem to get weird (e.g. every `FillContext` seems to have a clone of the `RNTupleModel`), but then we found that `RNTupleParallelWriter` explicitly does not support late schema extension yet
    - Perhaps something to be looked at closely later, when `RNTupleParallelWriter` would gain (or be close to gain) that feature?
- We would like to have more explanation on "entry invalidation" when an `RNTupleModel` is extended

`RField`

- Does `RFieldBase::BindValue(shared_ptr<void>)` take shared ownership of the argument?
  - Presumably, but would be good to note explicitly in the doxygen documentation
- Out of curiosity, is there any active prevention for users extending the `RField` class hierarchy?

`REntry`

- For reading `REntry` is created with `RNTupleModel`, but for writing `REntry` is created with `RNTupleWriter`
  - We'd suggest to either add `CreateEntry()` function to `RNTupleReader`, or remove `CreateEntry()` from `RNTupleWriter`
- Having one `REntry` class for both reading and writing could be debated on
  - Why not continue the same read/write separation as with `RNTupleReader/Writer`?
  - With one `REntry` object, are we allowed to use it for both reading and writing?
    - I.e. can both `Bind*()` functions and `GetPtr()` function of one `REntry` be used?
    - Or can `Bind*()` functions be called from a reader `REntry` object?