# RNTuple Schema Evolution: Status and Discussion

2024-12-03

ROOT

Data Analysis Framework

https://root.cern

# Introduction

- Schema evolution: Read an on-disk field into a different in-memory type
  - Usually in the context of user-defined classes that change over time
  - But also applicable to native top-level fields (e.g., `std::vector<float> → RVec<float>`)

- Schema evolution governed by *rules*
  - Implicit (automatic) rules and
  - I/O customization rules / (manual) read rules
    - Overwrite implicit rules
    - Extend the capabilities of automatic rules

- Plan: RNTuple will implement the well-established ROOT I/O schema evolution mechanisms, with caveats (see later)
  - At the same time, this is an opportunity to improve the ROOT schema evolution in general
    - Documentation and ability to reason about the system
      (i.e.: make it clearer what users can and cannot rely on)
    - Fixing / better define behavior in edge cases
    - Passing custom information to the read rule

- Introduction question: to what extent do we expect classes of a given version to be subject to both TTree and RNTuple schema evolution?
  - E.g.: Event v2-v10 TTree only, *v10-v11 TTree and RNTuple (?)*, v12+ RNTuple only
  - (Note that types may also be stored as bare objects in the file, outside TTree and RNTuple)

# Examples

## Automatic Evolution

```
class Event {
   double fPt;
   int fProperties;
   int fDeprecated;
   std::vector<std::uint16_t> fTrackIndices;

   ClassDef(Event, 2)
};
```

```
class Event : public Base { // added base class
   int fProperties; // Order of members changed
   float fPt; // Changed from double to float
   // Removed member: fDeprecated
   float fNewMember;
   // Changed both: container type and item type
   ROOT::RVec<std::uint64_t> fTrackIndices;

   ClassDef(Event, 3)
};
```

## Manual Evolution

```
class Event {
   std::string fProperties;
   float fTemperature;
   float fX;
   float fY;

   ClassDef(Event, 2)
};
```

```
class Event {
   // no implicit conversion from string to int
   int fProperties;
   // Unit change from Kelvin to Celsius
   float fTemperature;
   // Change from Euclidian to Polar coordinates
   float fR;
   float fPhi;

   ClassDef(Event, 3)
};
```

*see later on the ``how''*

# Automatic Rules

| | Class Layout Change | RNTuple Support | Comment |
|---|---|---|---|
| **Class members** | Reorder members, remove member | Available | |
| | Add new member | Draft PR | |
| **Base classes (*not* intermediate)** | Reorder base classes, remove base class | Available | |
| | Read derived in-memory class from base on-disk class | → Manual rules | |
| | Add new base class | Draft PR | |
| **Types with identical on-disk representation** | std::pair ←→ std::tuple | Available | |
| | std::unique_ptr ←→ std::optional | Available | |
| | std::vector ←→ ROOT::RVec ←→ collection proxy ←→ `std::*set` | Available | Recursive evolution, e.g. `vector<int32_t>` → `RVec<int64_t>` |
| | Between std::[unordered_][multi]map **(unique constraint check tbd)** | Available | |
| | Between std::[unordered_][multi]map and sequential collection of std::pair | Available | |
| **PoD transformations (column-level transformation)** | Between bool and integral types (except std::byte) | Available | |
| | Between integral types with bounds checking (except std::byte) | Available | |
| | Between floating point types **(safety check for FP class still tbd)** | Available | |
| **Field-level transformations** | enum ←→ integral type | Tbd (simple) | |
| | std::atomic<T> ←→ T | Tbd (simple) | |
| | std::unique_ptr<T>, std::optional<T> ←→T | Tbd (intricate) | uni-directional only? |
| | fixed-sized array ←→sequential collection | Tbd (intricate) | currently available for RVec only |
| **Class hierarchy changes** | Move members between base and derived class | → Manual rules | Prefer to move to manual schema evolution, if feasible |
| | Insert or drop intermediate classes | → Manual rules | |

# Manual Rules

- Recap
  - I/O customization rules are part of the dictionary, not persistified in the ROOT file (in principle, code exists to do so but no clear case for it could be made)
  - Code snippets embedded in generated code

```
class Event {
   std::string fProperties;
   float fTemperature;
   float fX;
   float fY;

   ClassDef(Event, 2)
};
```

Linkdef.h

```
class Event {
   // no implicit conversion from string to int
   int fProperties;
   // Unit change from Kelvin to Celsius
   float fTemperature;
   // Change from Euclidian to Polar coordinates
   float fR;
   float fPhi;

   void SetFromEuclidian(float x, float y);

   ClassDef(Event, 3)
};
```

(additional rules for `fProperties` and `fTemperature`)

```
#pragma read sourceClass = "Event" source = "float fX; float fY" version = "[2]" \
   targetClass = "Event" target = "fPhi,fR" \
   code = "{ newObj->SetFromEuclidian(onfile.fX, onfile.fY); }"
```

Access to `newObj`, `onfile` struct and target member references (not shown)

# Manual Rules: Dictionary

We could hide the internal variables

write-only access

```
// Schema evolution read functions
static void read_Event_0( char* target, TVirtualObject *oldObj )
{
   // generated code
   struct Event_Onfile {
      float &fX;
      float &fY;
      Event_Onfile(float &onfile_fX, float &onfile_fY ): fX(onfile_fX), fY(onfile_fY) {}
   };
   static Long_t offset_Onfile_Event_fX = oldObj->GetClass()->GetDataMemberOffset("fX");
   static Long_t offset_Onfile_Event_fY = oldObj->GetClass()->GetDataMemberOffset("fY");
   char *onfile_add = (char*)oldObj->GetObject();
   Event_Onfile onfile(
      *(float*)(onfile_add+offset_Onfile_Event_fX),
      *(float*)(onfile_add+offset_Onfile_Event_fY) );
   static TClassRef cls("Event");
   static Long_t offset_fPhi = cls->GetDataMemberOffset("fPhi");
   float& fPhi = *(float*)(target+offset_fPhi);
   static Long_t offset_fR = cls->GetDataMemberOffset("fR");
   float& fR = *(float*)(target+offset_fR);
   Event* newObj = (Event*)target;
   //--- User's code ---
   newObj->SetFromEuclidian(onfile.fX, onfile.fY);
}
```

internal use

reference to target, can set e.g. private member

# Examples

1. Value update: temperature K ➔ C
2. Coordinate transformation euclidean to polar (multi-source, multi-target)
3. Non-trivial type change, e.g. string to integer

---

4. Update to/from a member within an object member (like `fTrack.fInfoLinkedToParent`)
5. Move member in the class hierarchy, e.g. to/from base class, to/from member
6. Add/remove intermediate classes
7. Class rename
8. Data member rename

- Some examples may be difficult to accomplish today
- From the experiment point of view, what are (were) the difficult cases?
- Which important cases are we missing?

# Discussion

- RNTuple currently supports rules targeting transient members if the in-memory and the on-disk class layout are identical
  - First full support for manual schema evolution requires the source member to be read into a staging area and passed to the rule (should not be difficult)
  - Plus the ability to deal with artificial (computed) fields (PR)
- We need to provide a way to schema-evolve streamer fields into native fields
- Not considered so far: class rename and member rename

- Manual rules work well for numerical members
  - Caveats related to changes to the class hierarchy shape
  - E.g., how should the user code ideally access on-disk information of complex, expired types

- Related feature requests
  - "After reading" rules
  - Automatically re-initialize transient members (optional behavior)
  - Pass a user-pointer down to the rule