RNTuple Attributes proposal

Giacomo Parolini
ROOT Team, CERN

What are attributes

- "**Attributes**" are arbitrary metadata that are linked to a *range of entries* in a RNTuple
- They have a user-defined schema specified through a RNTupleModel, This works just like regular entries with a few restrictions:
    - No late model extension
    - No projected fields
    - No streamer fields
    - No untyped collections / records
    - No write options
- A "global" attribute is just an attribute associated to the entire range of entries

What are attributes

event_id	pt.x	…
0	…	…
1	…	…
2	…	…

Each Set associates zero or more **attribute entries** to an entry range in the RNTuple

AttrSet 1

AttrSet 2

event_range	attr_foo
[0-1]	…

Attributes can overlap!

event_range	attr_bar	attr_baz
[0-2]	…	…
[2-10]	…	…

How to use - Writing

```
// Step 1: create model for the attribute set
auto attrModel = RNTupleModel::Create();
attrModel->MakeField<std::string>("lumiBlock");
attrModel->MakeField<int>("runNumber");

// Step 2: create the attribute set from the writer
auto attrSet = writer->CreateAttributeSet("MyAttrSet", std::move(attrModel));

// Step 3: open attribute range. attrRange has basically the same interface as REntry
auto attrRange = attrSet->BeginRange();

// Step 4: assign attribute values.
// Values can be assigned anywhere between BeginRange() and EndRange().
auto pLumi = attrRange->GetPtr<std::string>("lumiBlock");
*pLumi = GetLumiBlock();
for (int i = 0; i < 100; ++i) {
  auto entry = model->CreateEntry();
  // ... fill entry data ...
  writer->Fill(*entry);
}
auto pRunNumber = attrRange->GetPtr<int>("runNumber");
*pRunNumber = GetRunNumber();

// Step 5: close attribute range
attrSet->EndRange(std::move(attrRange));
```

Note: the Attributes API doesn't exist yet, this is just a proposal

How to use - Reading

```
// Step 1: retrieve your attribute set from the reader
const auto &attrSet = reader->GetAttributeSet("MyAttrSet");

// Step 2: access the attributes by entry range
for (auto i : reader->GetEntryRanges()) {
  // Access all attribute entries associated to this entry range.
  // Note that a range can have multiple “rows” of attributes associated.
  for (const auto &attrEntry : attrSet->GetAttributes(i)) {
    // uuid might for example be used to track the (full) provenance of this event
    auto pUuid = attrEntry->GetPtr<std::string>("uuid");
    std::cout << "Entry " << i << " has uuid " << *pUuid << "\n";
  }

  // Possible alternative syntax:
  for (const auto &uuid: attrSet->GetAttributes<std::string>("uuid", i))
    std::cout << "Entry " << i << " is associated to uuid " << uuid << "\n";
}

// Step 2': or, iterate directly over the attribute ranges
for (const auto &attr : attrSet->GetAttributeRanges()) {
  auto range = attr->GetRange();
  auto lumi = *attr->GetPtr<std::string>("lumiBlock");
  std::cout << "range " << Format(range) << " has lumi block " << lumi << "\n";
}
```

Note: the Attributes API doesn’t exist yet, this is just a proposal

Attribute Sets

- Attributes are organized in “**Attribute Sets**”
- Each Attribute Set has its own name and schema (1 RNTupleModel per Set)
- You can Begin and End attribute ranges from any Set
- There is a reserved implicitly-created “ROOT” Set (opt-out?)

Implementation details

- Attribute Sets are stored internally as RNTuples
- Linked to by the “main” RNTuple (probably in the footer)

Merging

- Since attributes are always linked to entry ranges, merging two RNTuples each with its own Attribute Sets is trivial: just concatenate the ranges for each Set
- This preserves the provenance of each attribute range
- A new range encompassing all merged ranges can be created as well, leading to a “matrioska” of attribute ranges
- We may want to provide a “squash” operation to handle attributes growing too much (but can be an external tool)

Open questions

- Do we want/need to support empty attribute ranges?
  - Upside: allow to still support “global” attributes for empty RNTuples
  - Downside: semantics are controversial, especially when merging
- Do we want/need to support creating overlapping attribute ranges (other than from merging) from the same attribute set?
  - I.e. do we allow arbitrarily interleaving Begin and EndRange?
- Is the Begin/EndRange API sufficient or do we want a more flexible API? (decouple metadata and data writing)