Native support for metadata specification in RDataFrame

I. Kabadzhov, V. E. Padulano, E. Guiraud

ROOT

Data Analysis Framework

https://root.cern

Agenda

1. Status of the current dataset specification from before
2. Creating RDF from semi-structured format (JSON)
3. New API to handle dataset specifications and metadata
4. Comparison with other HEP dataset specification APIs
5. Open Design Questions
6. Future steps

1. Recap of the “old” RDatasetSpec

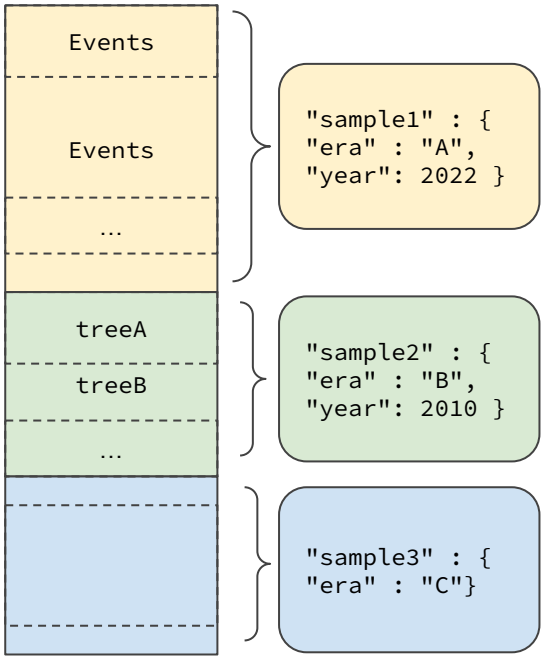
RDatasetSpec aims at providing user-friendly interface to easily:

1. Create vertical concatenations of datasets (**chains**)
2. Create horizontal concatenations of datasets (**friends**)
3. Allow **global ranges in MT** (solve #7702)

```
auto spec = RDatasetSpec({"treeA", "file0"}, {"treeB", "file1"}, {42, 1000});
spec.AddFriend({{"treeA", "file2"}, {"treeB", "file3"}}, "alias");
ROOT::RDataFrame df(spec);
```

1. Why not RDatasetSpec from before?

But, as part of the dataset specification, we would like to separate files in **groups**, such that each group has its own **metadata**. Obviously, the model from the previous PPP on RDatasetSpec was **too simplistic**.

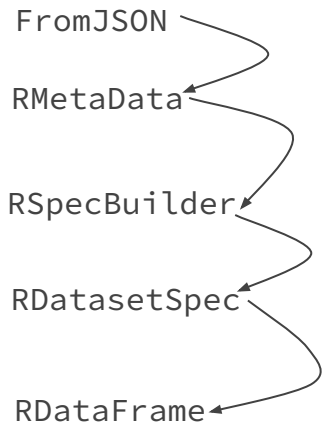


2. JSON dataset specification

Example JSON File (color code: mandatory):

```
{ "groups": [
      {"tag": "MC",
       "trees": ["Ev", "tree"],
       "files": ["a/f0*", "a/f1*"],
       "metadata" : { "energy_deposit" : 19.,
                      "era" : "A"}}, …}],
  "friends": {
      "fr0": { "trees": ["t0"], "files": ["f42.root", "f50.root"]}, …,},
  "range": [42, 1000]}
```

* internally:



Create an RDF like so:

```
ROOT::RDataFrame df = ROOT::RDF::FromJSON("myspec.json"); // then see DefinePerSample on next slide
```

3. New API (for advanced users)

1. An object, behaving as a **heterogeneous dictionary** to store the metadata

```
RMetaData meta;
// types do not matter upon constructing the metadata
meta.Add("energy_deposit", 19.) // double
    .Add("year", 2022) // int
    .Add("era", "B"); // string
```

```
// types DO matter upon retrieving
meta.GetD("energy_deposit"); // returns 19.
meta.GetI("year"); // returns 2022
meta.GetS("era"); // returns "B"
```

2. **Builder pattern** to construct a **`RDatasetSpec`** as follows:

```
RSpecBuilder builder;
builder.AddGroup("MC", "tree", "file*.root", metaMC)
       .AddGroup("Run3", {{"events0", "events1"}, {"f.root", "ff.root"}}, metaR3);
ROOT::RDataFrame df(builder.Build());
```

3. Finally, the metadata can then be accessed from **`DefinePerSample`**:

```
df.DefinePerSample("deposit_scaling", [](unsigned int, const ROOT::RDF::RSampleInfo &id) {
                   if (id.GetS("era") == "B") return id.GetD("energy_deposit") * 3.14;
                   else return id.GetD("energy_deposit") * 2.718});
```

4. Other frameworks: JSON config files

PocketCoffea

```
{ "DATA_DoubleMuon_2017_EraB": {
      "metadata": { "sample": "DATA",
                    "year": "2017",
                    "isMC": "False",
                    "era": "B",
                    "nevents": "14501767"},
      "files": ["f0.root", "f1.root", "f2.root"]},
  "DATA_DoubleMuon_2017_EraC": … }
```

WRemnants

```
{   'ttbar' : { 'name' : "ttbar",
                'filepaths' : ["a/*", "b/*"],
                'xsec' : 119.71},
    'wwPostVFP' : { 'name' : "WWPostVFP",
                    'filepaths' : ["c/*.root"],
                    'xsec' : 75.8,
                    'group' : "Diboson"
    }, … }
```

AGC

```
{ "ttbar": { "nominal": { "files": [ {"path": "file4.root", "nevts": 100},
                                     {"path": "file5.root", "nevts": 200}],
                          "nevts_total": 300 }}, …}
```

4. Other frameworks: non-JSON configs

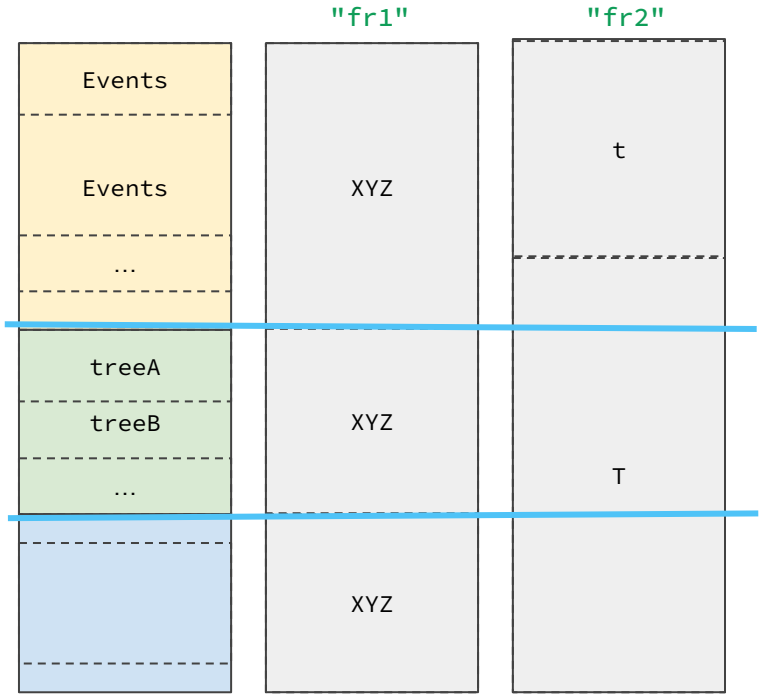
Coffea Processors

```
fileset = { 'treeA': ['file0.root', 'file1.root'],
            'treeB': ['file2.root', 'file3.root']}
out = processor.run_uproot_job(fileset, entry_start = 42, entry_stop = 10000,
                               metadata={"dataset": "DoubleMuon"})
```

Bamboo YAML files

```
TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8__2016ULpreVFP:
    era: 2016ULpreVFP
    files:
        - file0.root
        - file1.root
TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8__2016ULpostVFP: …
```

5. Open Design Questions: Friends



* caveat: user's responsibility to assure friends are aligned to the global chain

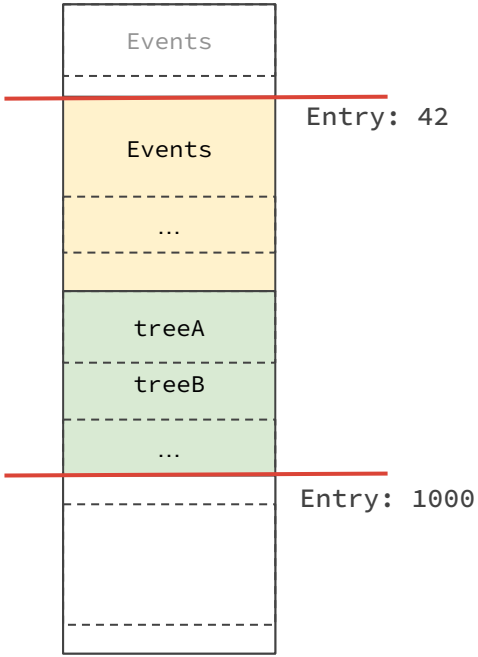
We do support horizontal concatenations of datasets (**friends**).

But with the current implementation we can only support global friends.

- Is this sufficient?
- Do we need friends per group?

```
RSpecBuilder builder;
builder.AddGroup(...).AddGroup(...).AddGroup(...);
builder.WithFriends("fr1", "XYZ", {"f0*", "f1*", "f2*"})
builder.WithFriends("fr2", {"t", "T"}, {"f.root", "g.root"})
ROOT::RDataFrame df(builder.Build());
```

5. Open Design Questions: Ranges



We introduced an alternative to Range for MT runs, but it is now only global.

- Is it sufficient?
- Do we need ranges per group?

```
RSpecBuilder builder;
builder.AddGroup(...).AddGroup(...).AddGroup(...);
builder.WithRange(42, 1000);
ROOT::RDataFrame df(builder.Build());
```

5. Open Design Questions: Repetitions

- Internally, a tree is uniquely identified by: treename + filename
- However, this disallows the case where the same treename + filename is in >=2 different groups and we want to retrieve metadata for each group separately!
- Are there scenarios in which we need the same file in more than one group?

```
RSpecBuilder builder;
builder.AddGroup("MC", "tree", "file.root", metaMC).
       .AddGroup("Run3", "tree", "file.root", metaR3);
ROOT::RDataFrame df(builder.Build()); // throw here!
```

5. Open Design Questions: Metadata

We saw from the different formats that the metadata keys in different groups varies - for instance:

```
{"tag": "WW",
 "trees": ["Ev", "tree"],
 "files": ["a/f0*", "a/f1*"],
 "metadata" : { "xsec" : 119.71,
                "era" : "A"}},
{"tag": "ttbar",
       "trees": ["tree"],
       "files": ["c/*.root"],
       "metadata" : { "era" : "B"}}
```

`DefinePerSample` would create a new column, but how to fill the entries for which the key is not present?

```
df.DefinePerSample("xsec_col", [](unsigned int,
   const ROOT::RDF::RSampleInfo &id) {
      return id.GetD("xsec");
});
```

```
// Our proposal: getters have default values, i.e.:
double RMetaData::GetD(const std::string &key,
                       double defaultValue=0.);
// if the key is not present, return the default value
```

6. Future steps

1. Dataset specification and metadata handling to appear in 6.28.
2. Implementation will be updated based on today's discussion and user feedback