RAL: Key4hep standard analysis library

Juan Carlos Aranda Huecas, Juraj Smiesko

Sep, 2024

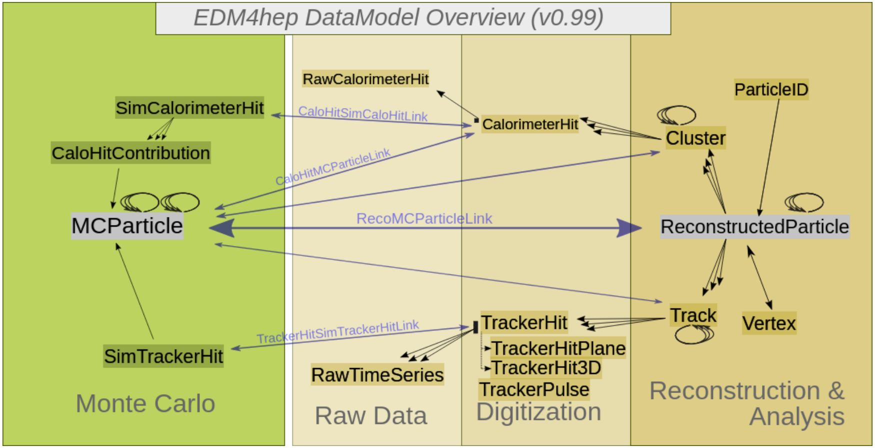
FCCAnalysis and EDM4hep

The FCCAnalyses framework is used for the majority of the FCC related analyses. Individual analyses are designed to use the ROOT RDataFrame as the main event processor and the analyses input files contain physics objects in EDM4hep format.

EDM4hep is a generic event data model for future HEP collider experiments. It uses a YAML file to describe the data model and PODIO to generate the code at runtime.

FCC simulation results are stored using this data model in ROOT files. Every analysis needs to include a family of basic functions.

EDM4HEP


EDM4hep DataModel Overview (v0.99)
RawCalorimeterHit
SimCalorimeterHit
CaloHitSimCaloHitLink
CalorimeterHit
CaloHitContribution
CaloHitMCParticleLink
ParticleID
Cluster
MCParticle
RecoMCParticleLink
ReconstructedParticle
Track
Vertex
TrackerHit
TrackerHitSimTrackerHitLink
SimTrackerHit
TrackerHitPlane
TrackerHit3D
RawTimeSeries
TrackerPulse
Monte Carlo
Raw Data
Digitization
Reconstruction & Analysis

RAL motivations and objectives

- The FCCanalysis library is not organized and does not cover all EDM4HEP Classes.
- Having analyzers in a different library alow us to choose the version of EDM4hep on runtime depending on the dataset.
- Being able to plot quickly some information of the simulation in an interactive way, using root interpreter.

Objective: Create a library that access data, sort, select and reduce the different collections of EDM4hep data classes in an standard way.

Project milestones

Development of ral library:

- Set up build system.
- Set up documentation system and page.
- Set up Github pipelines.
- Create a set of standard functions for multiple EDM4HEP Classes.
- Create a unittests and integrations tests for the library.

Build system

CMake was selected to build the project.

- The build script was kept minimal.
- C++ 20 was selected as the standard for the project.

Documentation system

Doxygen was selected as the framework for building the documentation. Documentation is also build with CMake.

Doc page: https://hep-fcc.github.io/ral/

Github Pipelines

GitHub Actions

There are **3 pipelines** at this moment that are triggered by pull requests.

1. Compile and running tests with CMake
2. Checking code formatting with clang-format
3. Building documentation and publishing it

Future upgrades:

- Create a new pipeling for publishing the library in the key4hep stack

Library design

The library should be able to apply this actions on EDM4hep classes:

- Take one component out of every objects in a collection
- Be able to debug/print components from a collection
- Create complex filters and select objects base on them
- Sort collections depending on their components
- Reduce collections and their components
- Access related collections

The library should be able to act on:

- RVec of EDM4hep Data classes
- EDM4hep Collection classes

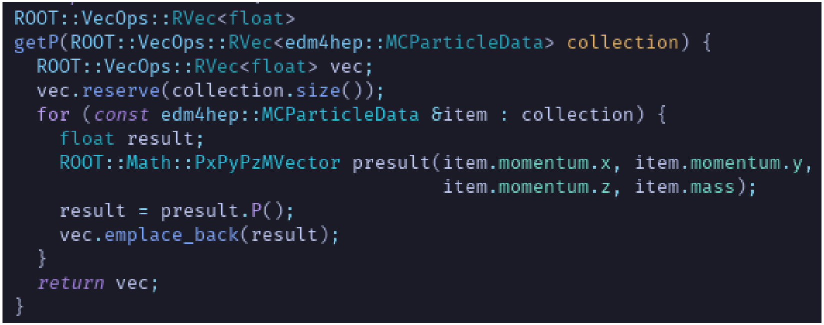
Metaprograming

- Since the code for every class in EDM4hep is going to be similar, **python package has been developed to generate ral code**.
- First prototype does not have any third party dependency.
- **Each EDM4hep class has its python script** that specifies the methods to generate using the metaprograming package.
- **Could be improved using Jinja Templating**.

Getters

- Output and RVec with a component of every element in the collection.
- No great differences between RVec and Collection implementation.

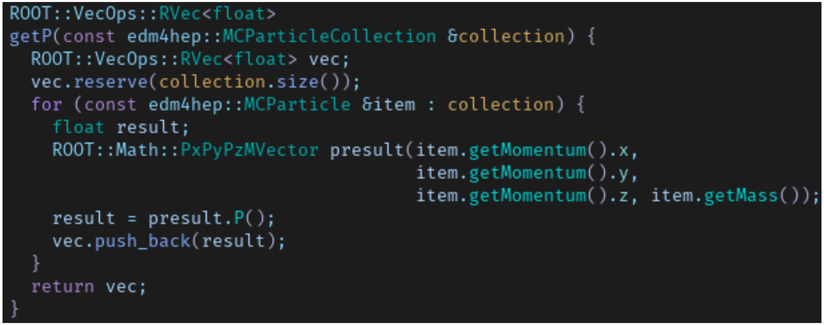
RVec implementation:



```
ROOT::VecOps::RVec<float>
getP(ROOT::VecOps::RVec<edm4hep::MCParticleData> collection) {
  ROOT::VecOps::RVec<float> vec;
  vec.reserve(collection.size());
  for (const edm4hep::MCParticleData &item : collection) {
    float result;
    ROOT::Math::PxPyPzMVector presult(item.momentum.x, item.momentum.y,
                                      item.momentum.z, item.mass);
    result = presult.P();
    vec.emplace_back(result);
  }
  return vec;
}
```

Getters

Collection implementation:

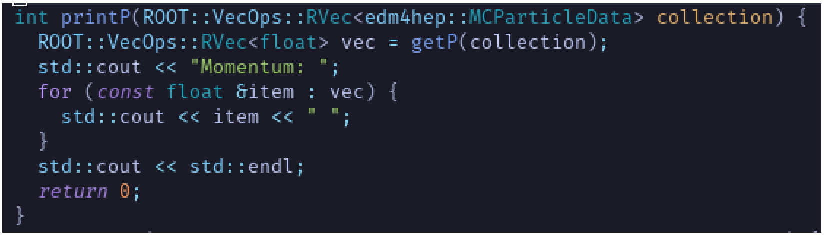


```
ROOT::VecOps::RVec<float>
getP(const edm4hep::MCParticleCollection &collection) {
  ROOT::VecOps::RVec<float> vec;
  vec.reserve(collection.size());
  for (const edm4hep::MCParticle &item : collection) {
    float result;
    ROOT::Math::PxPyPzMVector presult(item.getMomentum().x,
                                      item.getMomentum().y,
                                      item.getMomentum().z, item.getMass());

    result = presult.P();
    vec.push_back(result);
  }
  return vec;
}
```

Printers

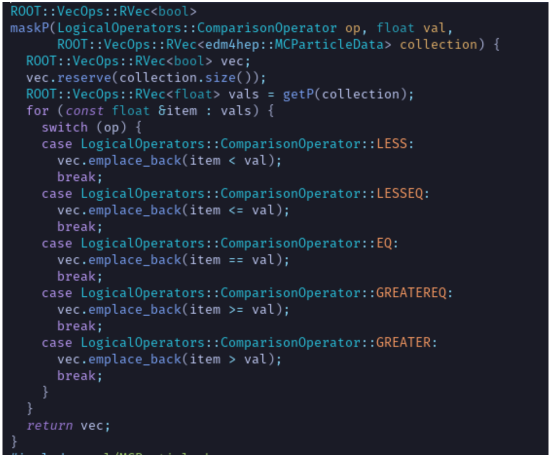
- Use getters to obtain the info to print.
- Print the info through the standard output.
- Their purpose is to be used as a debugging tool.



```
int printP(ROOT::VecOps::RVec<edm4hep::MCParticleData> collection) {
  ROOT::VecOps::RVec<float> vec = getP(collection);
  std::cout << "Momentum: ";
  for (const float &item : vec) {
    std::cout << item << " ";
  }
  std::cout << std::endl;
  return 0;
}
```

Masking

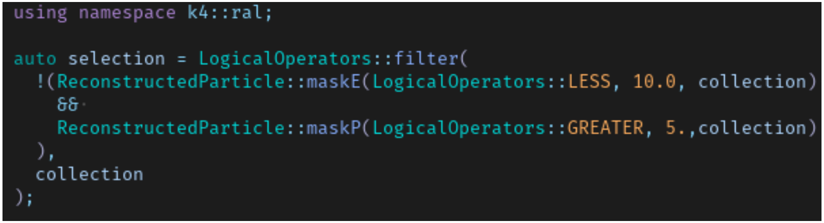
- Create boolean masks from conditions over class components that can be combined into complex masks.
- There is a filter method implemented to apply the masks.



```
ROOT::VecOps::RVec<bool>
maskP(LogicalOperators::ComparisonOperator op, float val,
      ROOT::VecOps::RVec<edm4hep::MCParticleData> collection) {
  ROOT::VecOps::RVec<bool> vec;
  vec.reserve(collection.size());
  ROOT::VecOps::RVec<float> vals = getP(collection);
  for (const float &item : vals) {
    switch (op) {
    case LogicalOperators::ComparisonOperator::LESS:
      vec.emplace_back(item < val);
      break;
    case LogicalOperators::ComparisonOperator::LESSEQ:
      vec.emplace_back(item <= val);
      break;
    case LogicalOperators::ComparisonOperator::EQ:
      vec.emplace_back(item == val);
      break;
    case LogicalOperators::ComparisonOperator::GREATEREQ:
      vec.emplace_back(item >= val);
      break;
    case LogicalOperators::ComparisonOperator::GREATER:
      vec.emplace_back(item > val);
      break;
    }
  }
  return vec;
}
```

Complex Filtering

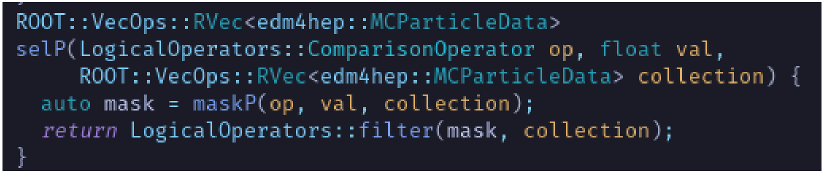
- Masks can be combined to create complex filters


```
using namespace k4::ral;

auto selection = LogicalOperators::filter(
  !(ReconstructedParticle::maskE(LogicalOperators::LESS, 10.0, collection)
    &&
    ReconstructedParticle::maskP(LogicalOperators::GREATER, 5.,collection)
  ),
  collection
);
```

Selectors

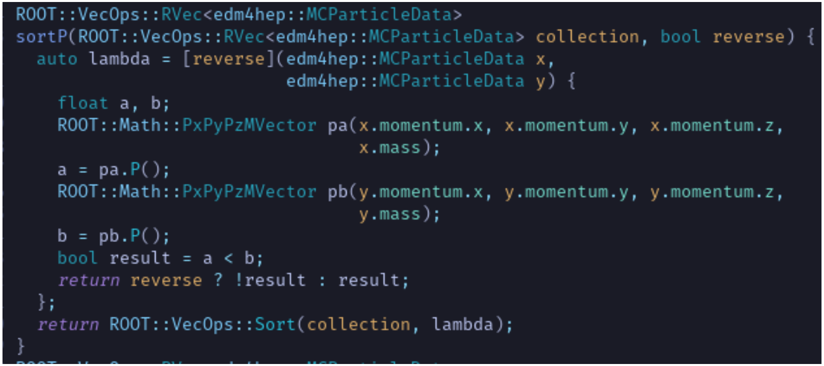
- Use mask methods to obtain the boolean mask that will be used to filter the collection.
- They are just simple filtering methods.
- **Possible improvement:** Instead of using our custom implementation of filter, ROOT::VecOps implementation could be used.



```
ROOT::VecOps::RVec<edm4hep::MCParticleData>
selP(LogicalOperators::ComparisonOperator op, float val,
     ROOT::VecOps::RVec<edm4hep::MCParticleData> collection) {
  auto mask = maskP(op, val, collection);
  return LogicalOperators::filter(mask, collection);
}
```

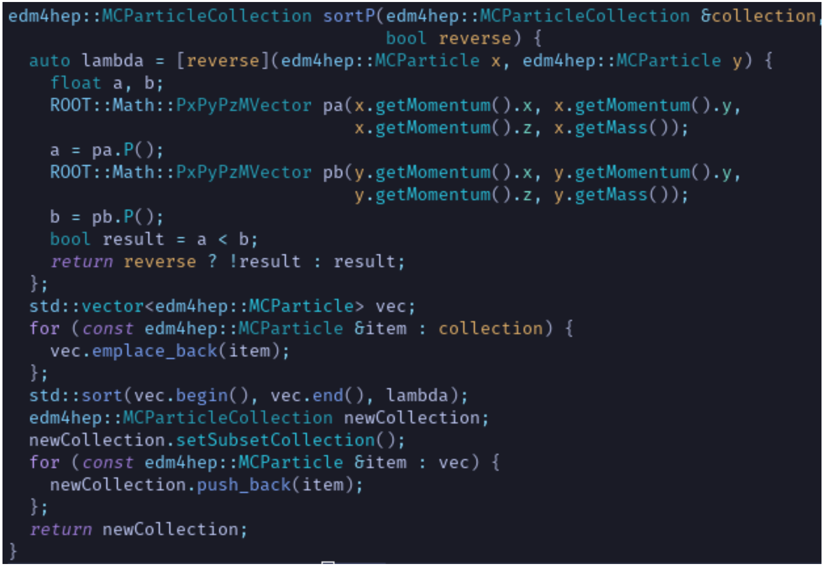
Sorters

- Use the VecOps Sort method with a lambda to generate the shorting decision.



```
ROOT::VecOps::RVec<edm4hep::MCParticleData>
sortP(ROOT::VecOps::RVec<edm4hep::MCParticleData> collection, bool reverse) {
  auto lambda = [reverse](edm4hep::MCParticleData x,
                          edm4hep::MCParticleData y) {
    float a, b;
    ROOT::Math::PxPyPzMVector pa(x.momentum.x, x.momentum.y, x.momentum.z,
                                 x.mass);
    a = pa.P();
    ROOT::Math::PxPyPzMVector pb(y.momentum.x, y.momentum.y, y.momentum.z,
                                 y.mass);
    b = pb.P();
    bool result = a < b;
    return reverse ? !result : result;
  };
  return ROOT::VecOps::Sort(collection, lambda);
}
```

Sorters



```
edm4hep::MCParticleCollection sortP(edm4hep::MCParticleCollection &collection,
                                    bool reverse) {
  auto lambda = [reverse](edm4hep::MCParticle x, edm4hep::MCParticle y) {
    float a, b;
    ROOT::Math::PxPyPzMVector pa(x.getMomentum().x, x.getMomentum().y,
                                 x.getMomentum().z, x.getMass());
    a = pa.P();
    ROOT::Math::PxPyPzMVector pb(y.getMomentum().x, y.getMomentum().y,
                                 y.getMomentum().z, y.getMass());
    b = pb.P();
    bool result = a < b;
    return reverse ? !result : result;
  };
  std::vector<edm4hep::MCParticle> vec;
  for (const edm4hep::MCParticle &item : collection) {
    vec.emplace_back(item);
  };
  std::sort(vec.begin(), vec.end(), lambda);
  edm4hep::MCParticleCollection newCollection;
  newCollection.setSubsetCollection();
  for (const edm4hep::MCParticle &item : vec) {
    newCollection.push_back(item);
  };
  return newCollection;
}
```

VecOps Library

Some of the objectives of the ral library are already implemented in the ROOT::VecOps library. There is no need to implemented them again.

Accesing subcollections by index

- ROOT::VecOps::Take

Reducing collections

- ROOT::VecOps::Sum
- ROOT::VecOps::Mean
- ROOT::VecOps::Max
- ...

Current Status

- Methods for 7 EDM4HEP classes are implemented:
  - MCParticle
  - ReconstructedParticle
  - CalorimeterHit
  - TrackerHit3D
  - Cluster
  - Track
  - Vertex
- Metaprograming needs to be expanded to generate tests for every new class implemented.
- Functions for accesing linked collections need to be implmented (ex. MCParticle - ReconstructedParticle).

Thank you!