EDM4hep.jl Prototype

Pere Mato / CERN
19 December 2023

https://github.com/peremato/EDM4hep.jl

Motivation

- Generate Julia ‘friendly’ structures for the EDM4hep data model
- Be able to read event data files (in ROOT) written by C++ programs from Julia (using the UnROOT.jl package)
- Later, be able also to write RNTuple files from Julia

Implementing EDM4hep in Julia is a pre-requisite for introducing the Julia language in Simulation and Reconstruction workflows

Main Design Features

- All entities are **immutable structs** for better performance, SoA, GPUs, etc.
  - POD with basic types and structs, including the relationships (one-to-one and one-to-many)
  - Objects attributes cannot be changed, new instances must be created (Accessors.jl)
- Constructors have keyword arguments with reasonable default values
- New objects are by default not registered, they are "free floating". Explicit registration or setting relationships will register them to containers.
- Note that operations like **register**, **setting relationships** will automatically create a new instances. The typical pattern is to overwrite the user variable with the new instance, e.g.:

```
p1 = MCParticle(...)
p1 = register(p1)
p1, d1 = add_daugther(p1, MCParticle(...))
```

- Reading EDM4hep containers from ROOT should result in **StructArrays**
  - Very efficient access by column and the same time provide convenient views as object instances

PODIO Generation

- Written small Julia script to generate Julia structs from YAML file
  - Added a ObjectID to each object to control its registration state
  - Relations implemented with ObjectID and Relations structs with just indices (isbits())
- Two files: genComponents.jl, genDatatypes.jl generated that can be complemented with useful methods

```
"""
struct MCParticle

    Description: The Monte Carlo particle - based on the lcio::MCParticle.
    Author: F.Gaede, DESY
"""
struct MCParticle <: POD
    index::ObjectID{MCParticle}      # ObjectID of himself

    #---Data Members
    PDG::Int32                       # PDG code of the particle
    generatorStatus::Int32           # status of the particle as defined by the ...
    simulatorStatus::Int32           # status of the particle from the simulation ...
    charge::Float32                  # particle charge
    time::Float32                    # creation time of the particle in [ns] wrt. ...
    mass::Float64                    # mass of the particle in [GeV]
    vertex::Vector3d                 # production vertex of the particle in [mm].
    endpoint::Vector3d               # endpoint of the particle in [mm]
    momentum::Vector3f               # particle 3-momentum at the production vertex..
    momentumAtEndpoint::Vector3f     # particle 3-momentum at the endpoint in [GeV]
    spin::Vector3f                   # spin (helicity) vector of the particle.
    colorFlow::Vector2i              # color flow as defined by the generator

    #---OneToManyRelations
    parents::Relation{MCParticle,1}  # The parents of this particle.
    daughters::Relation{MCParticle,2}  # The daughters this particle.
end
```

```
"""
struct SimTrackerHit

    Description: Simulated tracker hit
    Author: F.Gaede, DESY
"""
struct SimTrackerHit <: POD
    index::ObjectID{SimTrackerHit}   # ObjectID of himself
    #---Data Members
    cellID::UInt64                   # ID of the sensor that created this hit
    EDep::Float32                    # energy deposited in the hit [GeV].
    time::Float32                    # proper time of the hit in the lab frame in ...
    pathLength::Float32              # path length of the particle in the sensiti ...
    quality::Int32                   # quality bit flag.
    position::Vector3d               # the hit position in [mm].
    momentum::Vector3f               # the 3-momentum of the particle at the hits ...
    #---OneToOneRelations
    mcparticle_idx::ObjectID{MCParticle}  # MCParticle that caused the hit.
end
```

Building Model in Memory

```
#---MCParticles--------------------------------------------------------------------------------------------
p1 = MCParticle(PDG=2212, mass=0.938, momentum=(0.0, 0.0, 7000.0), generatorStatus=3)
p2 = MCParticle(PDG=2212, mass=0.938, momentum=(0.0, 0.0, -7000.0), generatorStatus=3)

p3 = MCParticle(PDG=1, mass=0.0, momentum=(0.750, -1.569, 32.191), generatorStatus=3)
p3, p1 = add_parent(p3, p1)

p4 = MCParticle(PDG=-2, mass=0.0, momentum=(-3.047, -19.000, -54.629), generatorStatus=3)
p4, p2 = add_parent(p4, p2)

p5 = MCParticle(PDG=-24, mass=80.799, momentum=(1.517, -20.68, -20.605), generatorStatus=3)
p5, p1 = add_parent(p5, p1)
p5, p2 = add_parent(p5, p2)

p6 = MCParticle(PDG=22, mass=0.0, momentum=(-3.813, 0.113, -1.833), generatorStatus=1)
p6, p1 = add_parent(p6, p1)
p6, p2 = add_parent(p6, p2)

p7 = MCParticle(PDG=1, mass=0.0, momentum=(-2.445, 28.816, 6.082), generatorStatus=1)
p7, p5 = add_parent(p7, p5)

p8 = MCParticle(PDG=-2, mass=0.0, momentum=(3.962, -49.498, -26.687), generatorStatus=1)
p8, p5 = add_parent(p8, p5)

#---Simulation tracking hits-------------------------------------------------------------------------
for j in 1:5
  sth1 = SimTrackerHit(cellID=0xabadcaffee, EDep=j*0.000001, position=(j * 10., j * 20., j * 5.), mcparticle=p7)
  sth1 = register(sth1)
  sth2 = SimTrackerHit(cellID=0xcaffeebabe, EDep=j*0.001, position=(-j * 10., -j * 20., -j * 5.), mcparticle=p8)
  sth2 = register(sth2)
end
```

Navigating Relationships

```
for p in getEDStore(MCParticle).objects
    println("MCParticle $(p.index) with PDG=$(p.PDG) and momentum $(p.momentum) has $(length(p.daughters)) daughters")
    for d in p.daughters
        println("   ---> $(d.index) with PDG=$(d.PDG) and momentum $(d.momentum)")
    end
end

for s in getEDStore(SimTrackerHit).objects
    println("SimTrackerHit in cellID=$(string(s.cellID, base=16)) with EDep=$(s.EDep) and position=$(s.position)
             associated to particle $(s.mcparticle.index)")
end
```

```
MCParticle #1 with PDG=1 and momentum (0.75,-1.569,32.191) has 0 daughters
MCParticle #2 with PDG=2212 and momentum (0.0,0.0,7000.0) has 3 daughters
   ---> #1 with PDG=1 and momentum (0.75,-1.569,32.191)
   ---> #5 with PDG=-24 and momentum (1.517,-20.68,-20.605)
   ---> #6 with PDG=22 and momentum (-3.813,0.113,-1.833)
MCParticle #3 with PDG=-2 and momentum (-3.047,-19.0,-54.629) has 0 daughters
MCParticle #4 with PDG=2212 and momentum (0.0,0.0,-7000.0) has 3 daughters
   ---> #3 with PDG=-2 and momentum (-3.047,-19.0,-54.629)
   ---> #5 with PDG=-24 and momentum (1.517,-20.68,-20.605)
   ---> #6 with PDG=22 and momentum (-3.813,0.113,-1.833)
MCParticle #5 with PDG=-24 and momentum (1.517,-20.68,-20.605) has 2 daughters
   ---> #7 with PDG=1 and momentum (-2.445,28.816,6.082)
   ---> #8 with PDG=-2 and momentum (3.962,-49.498,-26.687)
MCParticle #6 with PDG=22 and momentum (-3.813,0.113,-1.833) has 0 daughters
MCParticle #7 with PDG=1 and momentum (-2.445,28.816,6.082) has 0 daughters
MCParticle #8 with PDG=-2 and momentum (3.962,-49.498,-26.687) has 0 daughters
SimTrackerHit in cellID=abadcaffee with EDep=1.0e-6 and position=(10.0,20.0,5.0) associated to particle #7
SimTrackerHit in cellID=caffeebabe with EDep=0.001 and position=(-10.0,-20.0,-5.0) associated to particle #8
SimTrackerHit in cellID=abadcaffee with EDep=2.0e-6 and position=(20.0,40.0,10.0) associated to particle #7
SimTrackerHit in cellID=caffeebabe with EDep=0.002 and position=(-20.0,-40.0,-10.0) associated to particle #8
SimTrackerHit in cellID=abadcaffee with EDep=3.0e-6 and position=(30.0,60.0,15.0) associated to particle #7
SimTrackerHit in cellID=caffeebabe with EDep=0.003 and position=(-30.0,-60.0,-15.0) associated to particle #8
SimTrackerHit in cellID=abadcaffee with EDep=4.0e-6 and position=(40.0,80.0,20.0) associated to particle #7
SimTrackerHit in cellID=caffeebabe with EDep=0.004 and position=(-40.0,-80.0,-20.0) associated to particle #8
SimTrackerHit in cellID=abadcaffee with EDep=5.0e-6 and position=(50.0,100.0,25.0) associated to particle #7
SimTrackerHit in cellID=caffeebabe with EDep=0.005 and position=(-50.0,-100.0,-25.0) associated to particle #8
```

Integrated in the Julia ecosystem

- Simple structs (isbits) and vectors of them integrate well with the rest of the Julia ecosystem. Examples:
  - A container of EDM4hep datatypes can be converted to a **DataFrame** immediately
  - GPU array programming

```julia
using DataFrames
df = DataFrame(getEDStore(MCParticle).objects)
```

8×15 DataFrame

Row	index ObjectID…	PDG Int32	generatorStatus Int32	simulatorStatus Int32	charge Float32	time Float32	mass Float64	vertex Vector3d	endpoint Vector3d	momentum Vector3f	momentumAtEndpoint Vector3f	spin Vector3f	colorFlow Vector2i	parents Relation…	daughters Relation…	…
1	#1	1	3	0	0.0	0.0	0.0	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0.75,-1.569,32.191)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[2]	MCParticle#[]	…
2	#2	2212	3	0	0.0	0.0	0.938	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0.0,0.0,7000.0)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[]	MCParticle#[1, 5,	
3	#3	-2	3	0	0.0	0.0	0.0	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(-3.047,-19.0,-54.629)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[4]	MCParticle#[]	
4	#4	2212	3	0	0.0	0.0	0.938	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0.0,0.0,-7000.0)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[]	MCParticle#[3, 5,	
5	#5	-24	3	0	0.0	0.0	80.799	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(1.517,-20.68,-20.605)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[2, 4]	MCParticle#[7, 8]	…
6	#6	22	1	0	0.0	0.0	0.0	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(-3.813,0.113,-1.833)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[2, 4]	MCParticle#[]	
7	#7	1	1	0	0.0	0.0	0.0	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(-2.445,28.816,6.082)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[5]	MCParticle#[]	
8	#8	-2	1	0	0.0	0.0	0.0	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(3.962,-49.498,-26.687)	(0.0,0.0,0.0)	(0.0,0.0,0.0)	(0,0)	MCParticle#[5]	MCParticle#[]	

1 column omitted

Reading from a ROOT (TTree) File

```
using EDM4hep
using EDM4hep.RootIO

cd(@__DIR__)

f = "ttbar_edm4hep_digi.root"

reader = RootIO.Reader(f)
events = RootIO.get(reader, "events")

evt = events[1];

hits = RootIO.get(reader, evt, "InnerTrackerBarrelCollection")
mcps = RootIO.get(reader, evt, "MCParticle")

for hit in hits
    println("Hit $(hit.index) is related to MCParticle $(hit.mcparticle.index) with name $(hit.mcparticle.name)")
end

#---Loop over events-------------------------------------------------------------
for (n,e) in enumerate(events)
    ps =  RootIO.get(reader, e, "MCParticle")
    println("Event #$(n) has $(length(ps)) MCParticles with a charge sum of $(sum(ps.charge))")
end
```

```
Hit #1 is related to MCParticle #65 with name pi+
Hit #2 is related to MCParticle #65 with name pi+
Hit #3 is related to MCParticle #65 with name pi+
Hit #4 is related to MCParticle #65 with name pi+
Hit #5 is related to MCParticle #66 with name pi−
Hit #6 is related to MCParticle #66 with name pi−
Hit #7 is related to MCParticle #66 with name pi−
Hit #8 is related to MCParticle #49 with name pi+
Hit #9 is related to MCParticle #49 with name pi+
Hit #10 is related to MCParticle #49 with name pi+
Hit #11 is related to MCParticle #27 with name K−
Hit #12 is related to MCParticle #27 with name K−
Hit #13 is related to MCParticle #27 with name K−
Hit #14 is related to MCParticle #95 with name e−
Hit #15 is related to MCParticle #95 with name e−
...
```

~ 1500 times faster than Python

A real StructureOfArrays (SoA)

- Storage in memory consists of a set of column arrays
  - very fast access by column
- Materialize, when requested, object instances (usually on the stack) to be able to call object methods
  - to achieve a user friendly access

```
julia> typeof(mcps[1])
MCParticle

julia> typeof(mcps.charge)
SubArray{Float32, 1, Vector{Float32},
Tuple{UnitRange{Int64}}, true}

julia> length(mcps.charge)
211

julia> mcps[1:2].momentum
2-element StructArray(::Vector{Float32}, ::Vector{Float32},
::Vector{Float32}) with eltype Vector3f:
 (0.5000167,0.0,50.0)
 (0.5000167,0.0,-50.0)

julia> sum(mcps[1:2].momentum)
(1.0000334,0.0,0.0)
```

What's Next?

- Need to generate and support **VectorMembers**
  - Implementation similar to One-to-Many relations. For the time being are ignored
- Handle cyclic datatype dependencies
  - One case that is not yet resolved. `Vertex` <--> `ReconstructedParticle`. Use the abstract class in this case
- Better handle collectionID in one-to-many relations
- Be able to read RNTuple files in addition to TTree files
- Easy schema evolution profiting from the fact that ROOT files are self-described and Julia types can be introspected