KM3io.jl

Making UnROOT.jl comfortable for KM3NeT...

Tamas Gal (ECAP)

November 6, 2023

- KM3io.jl is a tiny Julia package to read KM3NeT ROOT files
- Uses the UnROOT.jl library to access ROOT data
- Provides convenient wrapper functions to iterate through events and other types of structures
- Data are read lazily - only loaded into memory when directly accessed
- Includes many commonly used utility functions

Why do we not use directly?

- Because the KM3NeT ROOT dataformat is a bit more complicated...
- Many custom classes (C++)
- Low branch-splitting (even uproot chokes on them)
- Some bit-fiddling tricks to save space
- Magic numbers for masks and indices which we want to hide from the user (and prevent pollution downstreams)

Let's try to read some PMT hit-level data

```
julia> using UnROOT, KM3NeTTestData

julia> f = ROOTFile(datapath("online", "km3net_online.root"));

julia> LazyBranch(f, "KM3NET_EVENT/KM3NET_EVENT/snapshotHits")
ERROR: MethodError: no method matching getindex(::Nothing)
Stacktrace:
 [1] LazyBranch(f::ROOTFile, b::UnROOT.TBranchElement_9)
   @ UnROOT ~/Dev/UnROOT.jl/src/iteration.jl:119
 [2] LazyBranch(f::ROOTFile, s::String)
   @ UnROOT ~/Dev/UnROOT.jl/src/iteration.jl:133
 [3] top-level scope
   @ REPL[21]:1
```

UnROOT.jl chokes on it and does not give us any good hints (PR on the way to make the error message more mind-reading ;)

However, directly accessing the branch gives us a bit more information (here truncated for clarity)

```
julia> f["KM3NET_EVENT/KM3NET_EVENT/snapshotHits"]
UnROOT.TBranchElement_9
  cursor: UnROOT.Cursor
  fName: String "snapshotHits"
  fTitle: String "snapshotHits"
  fFillColor: Int16 0
  fFillStyle: Int16 1001
  ...
  ...
  fClassName: String "KM3NETDAQ::JDAQEvent"
  fParentName: String "KM3NETDAQ::JDAQEvent"
  ...
```

We now need to know that UnROOT.jl uses "dot-concatenation" as identifier, similar to uproot, so what we need to teach UnROOT is how to read a branch with the "type" `KM3NETDAQ::JDAQEvent.snapshotHits`

How does a "SnapshotHit" look like in the corresponding C++ framework?

```
typedef  unsigned char             JPMT_t;    //!< PMT channel in FPGA
typedef  unsigned int              JTDC_t;    //!< leading edge [ns]
typedef  unsigned char             JTOT_t;    //!< time over threshold [ns]
```

...and it derives from another class from which it inherits another `int` field...

Let's teach how to read a SnapshotHit and a vector of those

We start with a struct:

```
struct SnapshotHit <: AbstractDAQHit
    dom_id::UInt32
    channel_id::UInt8
    t::Int32
    tot::UInt8
end
```

...and tell how to read a single piece and a vector of those

```
function UnROOT.readtype(io, T::Type{SnapshotHit})
    T(UnROOT.readtype(io, Int32), read(io, UInt8), read(io, Int32), read(io, UInt8))
end
```

- That's not the full story since when looking at the C++ code, we realise that snapshotHits (a field of the `DAQEvent`) is actually a std::vector of SnapshotHit.
- We can now teach UnROOT.jl how to read that for a single instance of `DAQEvent` (requires some special knowledge about the ROOT serialisation...)

```
function UnROOT.interped_data(rawdata, rawoffsets, ::Type{Vector{SnapshotHit}}, ::Type{T}) where {T
    UnROOT.splitup(rawdata, rawoffsets, SnapshotHit, skipbytes=10)
end
```

We also need to pass this custom parsing information to UnROOT (I added a few more examples)

```
customstructs = Dict(
    "KM3NETDAQ::JDAQEvent.snapshotHits" => Vector{SnapshotHit},
    "KM3NETDAQ::JDAQEvent.triggeredHits" => Vector{TriggeredHit},
    "KM3NETDAQ::JDAQEvent.KM3NETDAQ::JDAQEventHeader" => EventHeader,
    "KM3NETDAQ::JDAQSummaryslice.KM3NETDAQ::JDAQSummarysliceHeader" => SummarysliceHeader,
    "KM3NETDAQ::JDAQSummaryslice.vector<KM3NETDAQ::JDAQSummaryFrame>" => Vector{SummaryFrame}
)
f = UnROOT.ROOTFile(filename, customstructs=customstructs)
```

Et voila

```
julia> b = LazyBranch(f, "KM3NET_EVENT/KM3NET_EVENT/snapshotHits")
3-element LazyBranch{Vector{SnapshotHit}, UnROOT.Nojagg, Vector{Vector{SnapshotHit}}}:
 SnapshotHit[SnapshotHit(0x30117974, 0x0a, 30733918, 0x1a), SnapshotHit(0x30117974, 0x0d, 30733916,
...
...
julia> b[1][1:4]
4-element Vector{SnapshotHit}:
 SnapshotHit(0x30117974, 0x0a, 30733918, 0x1a)
 SnapshotHit(0x30117974, 0x0d, 30733916, 0x13)
 SnapshotHit(0x30118a06, 0x00, 30733256, 0x19)
 SnapshotHit(0x30118a06, 0x03, 30733871, 0x16)
```

It can get a bit more complicated (or much more complicated):

Reading some other structures requires some sessions in a hex-editor and ROOT ;)

```
packedsizeof(::Type{EventHeader}) = 76
function UnROOT.readtype(io::IO, T::Type{EventHeader})
    skip(io, 18)
    detector_id = UnROOT.readtype(io, Int32)
    run = UnROOT.readtype(io, Int32)
    frame_index = UnROOT.readtype(io, Int32)
    skip(io, 6)
    UTC_seconds = UnROOT.readtype(io, UInt32)
    UTC_16nanosecondcycles = UnROOT.readtype(io, UInt32)
    skip(io, 6)
    trigger_counter = UnROOT.readtype(io, UInt64)
    skip(io, 6)
    trigger_mask = UnROOT.readtype(io, UInt64)
    overlays = UnROOT.readtype(io, UInt32)
    T(detector_id, run, frame_index, UTCExtended(UTC_seconds, UTC_16nanosecondcycles), trigger_coun
end
```

```
function UnROOT.interped_data(rawdata, rawoffsets, ::Type{EventHeader}, ::Type{T}) where {T <: UnRO
    UnROOT.splitup(rawdata, rawoffsets, EventHeader, jagged=false)
end
```

Lazy trees and branches are your friends to create a nice and easy-to-use API

```
struct OnlineTree
    _fobj::UnROOT.ROOTFile
    events::EventContainer
    summaryslices::SummarysliceContainer

    function OnlineTree(fobj::UnROOT.ROOTFile)
        new(fobj,
            EventContainer(
                UnROOT.LazyBranch(fobj, "KM3NET_EVENT/KM3NET_EVENT/KM3NETDAQ::JDAQEventHeader"),
                UnROOT.LazyBranch(fobj, "KM3NET_EVENT/KM3NET_EVENT/snapshotHits"),
                UnROOT.LazyBranch(fobj, "KM3NET_EVENT/KM3NET_EVENT/triggeredHits"),
            ),
            SummarysliceContainer(
                UnROOT.LazyBranch(fobj, "KM3NET_SUMMARYSLICE/KM3NET_SUMMARYSLICE/KM3NETDAQ::JDAQSum
                UnROOT.LazyBranch(fobj, "KM3NET_SUMMARYSLICE/KM3NET_SUMMARYSLICE/vector<KM3NETDAQ::
            )
        )

    end
end
```

Which can be utilised to tackle even more inaccessible stuff

(truncated source from `src/root/offline.jl` -> `OfflineTree{T}`)

```
branch_paths =[
    bpath * "/id",
    bpath * "/det_id",
    Regex(bpath * "/t/t.f(Sec|NanoSec)\$") => s"t_\1",
    Regex(bpath * "/hits/hits.(id|dom_id|channel_id|tdc|tot|trig|t)\$") => s"hits_\1",
    Regex(bpath * "/hits/hits.(pos|dir).([xyz])\$") => s"hits_\1_\2",
    Regex(bpath * "/trks/trks.(id|t|E|len|lik|rec_type|rec_stages|fitinf)\$") => s"trks_\1",
    Regex(bpath * "/trks/trks.(pos|dir).([xyz])\$") => s"trks_\1_\2",
    ...
    ...
    Regex(bpath * "/mc_event_time/mc_event_time.f(Sec|NanoSec)\$") => s"mc_event_time_\1",
    Regex(bpath * "/mc_hits/mc_hits.(id|pmt_id|t|a|pure_t|pure_a|type|origin)\$") => s"mc_hits_\1",
    Regex(bpath * "/mc_hits/mc_hits.(pos|dir).([xyz])\$") => s"mc_hits_\1_\2",
    Regex(bpath * "/mc_trks/mc_trks.(id|t|E|len|type|status|mother_id|counter)\$") => s"mc_trks_\1"
    Regex(bpath * "/mc_trks/mc_trks.(pos|dir).([xyz])\$") => s"mc_trks_\1_\2",
    bpath * "/index",
    bpath * "/flags",
```

```
]

t = UnROOT.LazyTree(fobj, tpath, branch_paths)
```

How does it feel like?

```
julia> using KM3io, KM3NeTTestData, Statistics

julia> f = ROOTFile(datapath("online", "km3net_online.root"))
ROOTFile{OnlineTree (3 events, 3 summaryslices), OfflineTree (0 events)}

julia> for event in f.online.events
           @show length(event.snapshot_hits) event.snapshot_hits[end]
       end
length(event.snapshot_hits) = 96
event.snapshot_hits[end] = SnapshotHit(0x3040a97d, 0x11, 30735112, 0x1b)
length(event.snapshot_hits) = 124
event.snapshot_hits[end] = SnapshotHit(0x3040a97d, 0x08, 58729262, 0x1b)
length(event.snapshot_hits) = 78
event.snapshot_hits[end] = SnapshotHit(0x3040a97d, 0x0a, 63512892, 0x17)

julia> for susl in f.online.summaryslices
           for frame in susl.frames
               println("$(frame.dom_id) -> median PMT rate: $(median(pmtrates(frame)))")
           end
       end
806451572 -> median PMT rate: 9624.605487994835
806455814 -> median PMT rate: 8873.374661957223
806465101 -> median PMT rate: 8636.2282772677
806483369 -> median PMT rate: 10160.436093826042
```

Another example using MC and reco data

```
julia> using KM3io, KM3NeTTestData

julia> f = ROOTFile(datapath("offline", "numucc.root"))
ROOTFile{OnlineTree (0 events, 0 summaryslices), OfflineTree (10 events)}

julia> f.offline[3]
Evt (3680 hits, 28 MC hits, 38 tracks, 12 MC tracks)

julia> f.offline[3].mc_trks[1:3]
3-element Vector{MCTrk}:
 MCTrk(0, [514.8075390949283, 252.68521293959793, 954.126], [-0.238656, 0.028334, -0.970691], 0.0,
 MCTrk(0, [514.8075390949283, 252.68521293959793, 954.126], [-0.238656, 0.028334, -0.970691], 0.0,
 MCTrk(1, [514.8075390949283, 252.68521293959793, 954.126], [-0.203636, -0.005391, -0.979032], 0.0,

julia> f.offline[3].mc_trks[2].pos.y
252.68521293959793
```

Properly typed instances

If you look closely, you notice that `KM3io.jl` hooked into the LazyTree and uses its own types when materialising the lazy data:

```
julia> f.offline[3].mc_trks[2].pos
3-element Position{Float64} with indices SOneTo(3):
 514.8075390949283
 252.68521293959793
 954.126

julia> typeof(f.offline[3].mc_trks[2].pos)
Position{Float64}
```

The magic behind it is creating a container struct for the `LazyTree` and implementing `Base.getindex(f::OfflineT`
`idx::Integer)`

Some truncated source code of the getindex function

```
function Base.getindex(f::OfflineTree, idx::Integer)
    e = f._t[idx]  # the event as NamedTuple: struct of arrays
    ...
    n = length(e.mc_trks_id)
    mc_trks = sizehint!(Vector{MCTrk}(), n)
    # legacy format support
    skip_mc_trks_status = !hasproperty(e, :mc_trks_status)
    skip_mc_trks_mother_id = !hasproperty(e, :mc_trks_mother_id)
    skip_mc_trks_counter = !hasproperty(e, :mc_trks_counter)
    for i  1:n
        push!(mc_trks,
            MCTrk(
                e.mc_trks_id[i],
                Position(e.mc_trks_pos_x[i], e.mc_trks_pos_y[i], e.mc_trks_pos_z[i]),
                Direction(e.mc_trks_dir_x[i], e.mc_trks_dir_y[i], e.mc_trks_dir_z[i]),
                e.mc_trks_t[i],
                e.mc_trks_E[i],
                e.mc_trks_len[i],
                e.mc_trks_type[i],
                skip_mc_trks_status ? 0 : e.mc_trks_status[i],
                skip_mc_trks_mother_id ? 0 : e.mc_trks_mother_id[i],
                skip_mc_trks_counter ? 0 : e.mc_trks_counter[i],
            )
        )
    end
    ...
end
```

Ressources

- ROOT is not an easy format to read (let alone write)

- If you need help with ROOT files which are inaccessible, open an issue at `https://github.com/JuliaHEP/UnROOT.jl` and we will try our best to help you!
- Docs: `https://common.pages.km3net.de/KM3io.jl`
- Repository: `https://git.km3net.de/common/KM3io.jl`