NAIA User Experience

Jian Tian 21/Dec/2022

Ntuple:
/storage/gpfs_ams/ams/groups/AMS-Italy/ntuples/v1.0.0/ISS.B1236/pass8

from 1305853512.root to 1635855691.root

20 May 2011 —> 2 Nov 2021

Ntuple

Step 1: FillSampleTree

(Some common selections)

Step 2: PlotEvent

(Customise selections)

(Draw the plots)

Selections:

Trigger: HasPhysicsTrigger;

RTI Selections;

Inner Tracker:

- Hit Pattern: Hits on Y side > 4 && at least one on each plane;
- Hits ChiSquare < 10;
- ChargeRMS/InnerCharge < 0.55;
- Fiducial volume in inner tracker;

TOF:

- Beta>0.3;
- Good hits on Upper TOF: GoodPathlength(0b0011);
- Charge in Upper TOF (charge - 0.75f, ∞);

Inner Tracker:

- Charge on each layer (charge - 0.3f, charge + 0.5f);
- Hits on Y side > 6;
- ChargeRMS/InnerCharge < 0.12;

TOF:

- TofPlus.Chi2Coo < 5.0f;
- TofPlus.Chi2Time < 10.0f;
- Charge in Upper TOF (charge - 0.5f, charge + 0.75f);
- Charge in Lower TOF (charge - 0.5f, charge + 0.75f);

NSL: the NAIA selection library

```
namespace ns = NSL::Selections;
namespace is = Ions::Selections;
```

lots of functions have been implemented

```
innerTrackerSel &=
    // is::InnerTracker::ChargeInRange(charge, chargeRecoType) &&
    ns::InnerTracker::ChargeRMSLessThan(0.55, chargeRecoType) &&
    ns::InnerTracker::NHitsGreaterThan(4, NAIA::TrTrack::Side::Y);
```

Some functions are really making the code easier to write and read

```
if (!triggerSel(event))
  continue;

// if (!TrackerL1Sel(event))
//   continue;

if (!tofSel(event))
  continue;

if (!innerTrackerSel(event))
  continue;

// if (!innerL1Sel(event))
//   continue;
```

Analysis with ROOT::RDataFrame

```
TChain *chain = new TChain("NAIAChain");
// chain->Add("../output/file_*.root");
chain->Add(inputFiles);
```

```
ROOT::RDataFrame rdf{*chain};
```

```
auto augmented_d =
    rdf
```

Filter: take a branch(or more branches),
filter rows based on user-defined conditions.

```
.Filter(
    [](NAIA::TrTrackBaseData &trBase) {
      return trBase.Rigidity[fitType][NAIA::TrTrack::Span::InnerOnly] < 0.f;
    },
    {"TrTrackBaseData"})
```

```
.Filter(
    [=](NAIA::TrTrackBaseData &trtrack) {
      return trtrack.InnerChargeRMS[NAIA::TrTrack::ChargeRecoType::STD] /
                    trtrack.InnerCharge[NAIA::TrTrack::ChargeRecoType::STD] <
                0.12;
    },
    {"TrTrackBaseData"})
```

Analysis with ROOT::RDataFrame

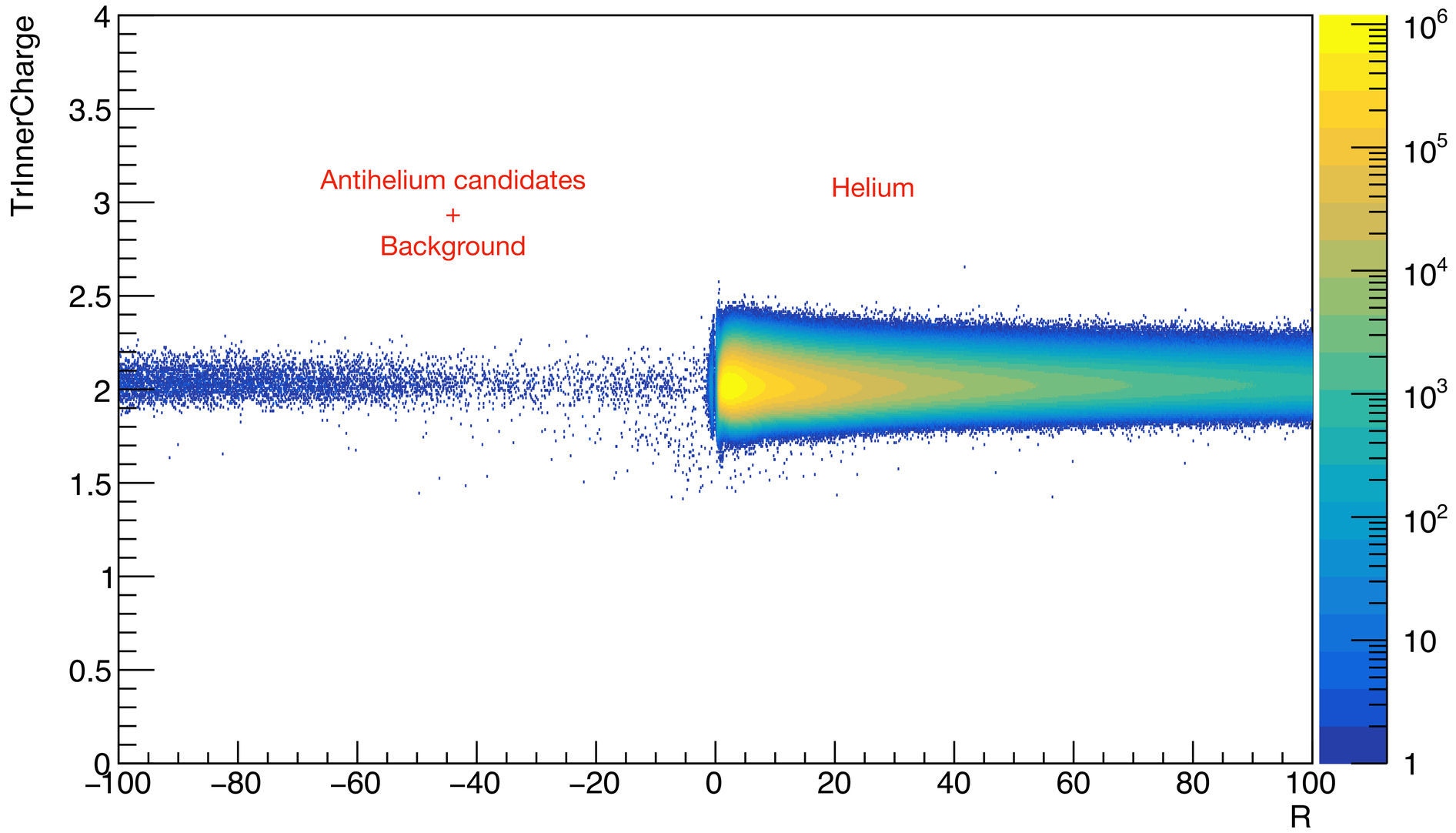
Define: Create a new column in the dataset.

```
.Define(
    "vRigidity",
    [](NAIA::TrTrackBaseData &trtrack) { return trtrack.Rigidity[fitType]
    [NAIA::TrTrack::Span::InnerOnly]; },
    {"TrTrackBaseData"})
```

```
.Define("vMass",
        [](NAIA::TofBaseData &TofBase, NAIA::TrTrackBaseData &trBasetrack) {
          return fGetMass(TofBase, trBasetrack);
        },
        {"TofBaseData", "TrTrackBaseData"})
```

```
auto hMass_vs_R =
    augmented_d.Histo2D({"Mass_vs_R", ";R;Mass", 2000u, -100., 100., 200u, -2., 18.},
     "vRigidity", "vMass", "weight");
```

Fill a one-, two-, three-dimensional histogram with the processed column values.


TrInnerCharge
4
3.5
3
2.5
2
1.5
1
0.5
0
−100
−80
−60
−40
−20
0
20
40
60
80
100
R
Antihelium candidates
+
Background
Helium
10^6
10^5
10^4
10^3
10^2
10
1

The NAIA data model

The NAIA data model is vaguely inspired by gbatch. The first thing needed to access data is to create a `NAIAChain` object

```
// ...
#include "Chain/NAIAChain.h"

int main(int argc, char const *argv[]) {
  NAIA::NAIAChain chain;
  chain.Add("somefile.root");
  chain.SetupBranches();
}
```

the `chain.SetupBranches()` is mandatory (with some work it could be made automatic with the instantiation of a chain, but this might come in a future release) and takes care of setting up the whole “read-on-demand” mechanism.

Looping

The chain contains all the events in the added runs, looping over events is particularly easy:

```
for (Event& event : chain){
  // your analysis here :)
}
```

If you’re uncomfortable with range-based for loops you can still do it the old fashioned way

```
unsigned long long nEntries = chain.GetEntries());

for (unsigned long long iEv = 0; iEv < nEntries; iEv++) {
  Event &event = chain.GetEvent(iEv);
```

NAIA
1.0.0
Search docs
NAIA
Getting started
The NAIA data model
The Event class
Skimming
Examples
CMake
Makefile
ROOT macros
RDataFrame
Simple macro

RDataFrame

Note

NB: this mode is not particularly tested, and usage of containers is slightly different

However, it is extremely cool

This example shows how to plot one histogram on NAIA events applying some simple selections, using the `RDataFrame` approach. There are a few caveats when using this approach:

- You don't use `NAIAChain`, instead you have to create the `RDataFrame` object reading the original tree from file, or creating a traditional `TChain`. How this ties with the `RTIInfo` and `FileInfo` trees is to be investigated.
- You have to work with the "Data" container classes, without the "read-on-demand" part. `RDataFrame` is supposed to take care of the rest by itself.
- You have to use the correct branch name in all the operations, which should be the same as the corresponding container "Data" class.

Simple macro

Note

NB: this mode is not particularly tested, and generally discouraged

This example shows how to loop on a `NAIAChain` from a root macro. It is identical to the simple CMake and Makefile examples.

Previous

NAIA

Main Page | Modules | Namespaces | Classes | Files | Search

Class List | Class Index | Class Hierarchy | Class Members

All Classes Namespaces Files Functions Variables Typedefs Enumerations Enumerator Friends Macros Groups

NAIA › TrTrackBaseData

Public Member Functions | Public Attributes | Private Attributes | Friends | List of all members

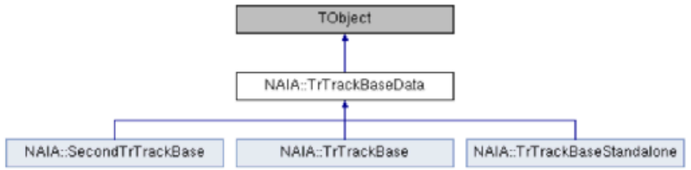
NAIA::TrTrackBaseData Class Reference

Container classes

Container class for base TrTrack info. More...

`#include <Containers/TrTrack.h>`

Inheritance diagram for NAIA::TrTrackBaseData:



Public Member Functions

void	**Clear** () Clear container content. More...
void	**Dump** () const Dump on screen container content. More...
TrTrack::Span	**GetBestSpan** (TrTrack::Fit fit=TrTrack::Fit::Choutko) const Get the best Span available for this track. More...
bool	**FitIDExists** (TrTrack::Fit fit, TrTrack::Span span) const Check if a given combination of fit and span is available for this track. More...
std::bitset< 9 >	**GetTrackPattern** (TrTrack::Side side) Get the track pattern for a given tracker side. One bit per layer, set to 1 if there is a hit on that layer on that side. You can also get the total number of hits using std::bitset::count. More...

Public Attributes

LayerVariable< std::array < float, 3 > >	TrTrackHitPos

Conclusions:

NAIA works well in my antihelium analysis.

NAIA is easy to use. The NAIA instruction online is very helpful.