

V. FORMATO - 11/03/2021 - AMS ITALY MEETING

---

**A NEW DST FOR AMS-ITALY**

## MOTIVATION

Until now each analysis group has been working with its own set of data. While this is certainly fine, on the long run it is introducing a few issues:

- ▶ Groups are competing for resources for ntuple production

*Usually happens when there is a new extension/pass*

- ▶ Difficulty cross-checking/ debugging / exchanging informations between groups (each one speaks its own "language")
- ▶ Increasing disk requests to accomodate N DST sets, some of which are "overlapping"

## SOLUTION AND REQUIREMENTS

- ▶ Have one DST for the whole AMS-Italy
  - ▶ It must be suited for all ongoing and near-future analyses
  - ▶ It should be performant (read only what needed)
  - ▶ It should be as light as possible (store only what's available)
  - ▶ It should be easy to read and use

NAIA: Ntuples (for) AMS-Italy Analysis

# SOLUTION AND REQUIREMENTS

- ▶ It must be suited for all ongoing and near-future analyses
- ▶ Process ALL events.
- ▶ Detailed list of all needed variables from every analysis group.
- ▶ Several possibilities for versioning and addition of new variables in the future. Still TBD.

Variable	PG	RM2	TN	BO	MIB	Legend
<b>Non-detector variables</b>						
Run	✓	✓	✓	✓	✓	Used in the analysis
Run Tag	✓	✓	✓	✓	✓	In the data model but maybe unused
RunAnalysisTag	✓	✓	✓	✓	✓	Not needed/used
Event number	✓	✓	✓	✓	✓	
UTime	✓	✓	✓	✓	✓	
<b>Event summary info</b>						
NAnticluster	✓	✓	✓	✓	✓	
NTofCluster	✓	✓	✓	✓	✓	
NTrackHits	✓	✓	✓	✓	✓	total number di hits nel TOF: è possibile dividerle
NTracks	✓	✓	✓	✓	✓	total number of hits in tracker
NRichRing	✓	✓	✓	✓	✓	
NEcalShowers	✓	✓	✓	✓	✓	
NParticles	✓	✓	✓	✓	✓	
<b>DAQ</b>						
DAQ Event length	✓	✓	✓	✓	✓	
DAQ errors	✓	✓	✓	✓	✓	8 flags
DAQ reply code errors	✓	✓	✓	✓	✓	14 flags
JINF room errors	✓	✓	✓	✓	✓	24 flags
JINJ slaves room errors	✓	✓	✓	✓	✓	24 flags
4x JINJ status	✓	✓	✓	✓	✓	
length of corresponding slave in JINJ block	✓	✓	✓	✓	✓	24 ints
<b>Trigger</b>						
PhysBPatt	✓	✓	✓	✓	✓	
PhysBPatt not restored	✓	✓	✓	✓	✓	PG: before lvl1->RestorePhysBPatt();
JMemBPatt	✓	✓	✓	✓	✓	
nACC	✓	✓	✓	✓	✓	for(int iacc=0; iacc<8; iacc++) if( ( (trigAntiPat
<b>Particle</b>						
Cutoff	✓	✓	✓	✓	✓	
Momentum	✓	✓	✓	✓	✓	
Beta	✓	✓	✓	✓	✓	
<b>Tracker</b>						
Track ID number	✓	✓	✓	✓	✓	
Rigidity and chi2 (X and Y):	✓	✓	✓	✓	✓	
(at mid-inner tracker)						
- Max span found (with its span id)	✓	✓	✓	✓	✓	PG: trkDefRig = trk_track->GetRigidity(idenf); int id
- Inner tracker fit	✓	✓	✓	✓	✓	
- Inner tracker + L1	✓	✓	✓	✓	✓	
- Inner tracker + L9	✓	✓	✓	✓	✓	
- Full Span	✓	✓	✓	✓	✓	
- Upper half	✓	✓	✓	✓	✓	PG: int ihup = trk_track->iTrTrackPar(fit_algo,5,ref
- Lower half	✓	✓	✓	✓	✓	PG: int ihdwn = trk_track->iTrTrackPar(fit_algo,6,ref
- Upper half inner	✓	✓	✓	✓	✓	
- Lower half inner	✓	✓	✓	✓	✓	
- Inner no MS	✓	✓	✓	✓	✓	
(at TOI)						
- Max span found (with its span id)	✓	✓	✓	✓	✓	PG: trkDefRig = trk_track->GetRigidity(idenf); int id
- Inner tracker fit	✓	✓	✓	✓	✓	
- Inner tracker + L1	✓	✓	✓	✓	✓	
- Inner tracker + L9	✓	✓	✓	✓	✓	
- Full Span	✓	✓	✓	✓	✓	
- Upper half	✓	✓	✓	✓	✓	PG: int ihup = trk_track->iTrTrackPar(fit_algo,5,ref
- Lower half	✓	✓	✓	✓	✓	PG: int ihdwn = trk_track->iTrTrackPar(fit_algo,6,ref
- Upper half inner	✓	✓	✓	✓	✓	
- Lower half inner	✓	✓	✓	✓	✓	
- Inner no MS	✓	✓	✓	✓	✓	
Global Tracker charge and RMS:	✓	✓	✓	✓	✓	

## SOLUTION AND REQUIREMENTS

- ▶ Should be **performant** (read only what needed)

This requires careful thinking of the data model structure:

Plain leaves make this hard to handle (due to having to call `TBranch::GetEntry` on each single branch before reading the values).

This could be hidden away behind getters and helper functions but it becomes quickly hard to maintain as the number of variables grows.

We discard the "plain leaves" option and opt to group variables in "container" classes, meant to hold variables from the same subdetector, to ease logic compartmentalization.

We chose to implement this "read on demand" behavior in a base class and apply it to all detector information.

# CONTAINER CLASSES

A container class is meant to just "hold data"

Each subdetector /element has one or more associated container classes

"Base" container classes hold the variables used most often

The screenshot displays the Doxygen-generated class reference for `NAIA::TofBaseData`. The page is titled "NAIA" and includes a navigation menu with "Main Page", "Namespaces", "Classes", and "Files". The current page is "NAIA::TofBaseData Class Reference".

**NAIA::TofBaseData Class Reference**

Container class for base `Tof` info. [More...](#)

`#include <Containers/Tof.h>`

Inheritance diagram for `NAIA::TofBaseData`:

```

graph BT
    TofBaseData["NAIA::TofBaseData"] --> TObject["TObject"]
    TofBase["NAIA::TofBase"] --> TofBaseData
    TofBaseStandalone["NAIA::TofBaseStandalone"] --> TofBaseData
  
```

**Public Member Functions**

- `void Clear ()`  
Clear container content. [More...](#)
- `void Dump ()`  
Dump on screen container content. [More...](#)
- `TVector3 InterpolateAtZ (float z) const`  
Get `Tof` track interpolation at given height. [More...](#)
- `const TVector3 & GetLinePoint ()`  
Get the origin of the `Tof` clusters linear fit. [More...](#)
- `const TVector3 & GetLineDirection ()`  
Get the direction of the `Tof` clusters linear fit. [More...](#)

**Public Attributes**

- `TofBetaVariable< float > Beta`  
Beta measurement for each type. See `Tof::BetaType` for list of types. [More...](#)
- `TofChargeVariable< float > Charge`  
Charge measurement for each type. See `Tof::ChargeType` for list of types. [More...](#)

**Private Attributes**

- `std::vector< TVector3 > m_pointAndDir`
- `bool _isMC = false`

**Detailed Description**

Container class for base `Tof` info.



# CONTAINER CLASSES

A container class is meant to just "hold data"

Each subdetector /element has one or more associated container classes

"Base" container classes hold the variables used most often, "Plus" container classes hold variables that won't be needed by everyone, or may be needed less frequently.

The screenshot shows a web browser displaying a Doxygen-generated class reference page for `NAIA::TofPlusData`. The page title is "NAIA" and the breadcrumb is "NAIA > TofPlusData". The page content includes:

- NAIA::TofPlusData Class Reference**: A container class for additional `Tof` info.
- `#include <Containers/Tof.h>`
- Inheritance diagram for NAIA::TofPlusData:**

```
graph BT; TofPlusData[NAIA::TofPlusData] --> TObject[TObject]; TofPlusPlus[NAIA::TofPlus] --> TofPlusData; TofPlusStandalone[NAIA::TofPlusStandalone] --> TofPlusData;
```
- Public Member Functions:**
  - `void Clear ()`: Clear container content.
  - `void Dump ()`: Dump on screen container content.
- Public Attributes:**
  - `float Chi2Coo`: spatial reconstruction chi-square.
  - `float Chi2Time`: temporal reconstruction chi-square.
  - `short NTrkClusters`: Total number of clusters matching Tracker track.
  - `short NBetaClusters`: Total number of clusters used for beta estimation.
  - `short NChargeClusters`: Total number of clusters used for charge estimation.
  - `float ChargeLikelihood`: Likelihood of charge estimation.
  - `TofBetaVariable< short > BetaPattern`: Pattern used for beta estimation for each type.
  - `std::vector< bool > LayerGoodPathI`: Pathlength check for each tof layer.
  - `std::vector< float > LayerCharge`: Charge estimation for each tof layer.

# READ ON DEMAND

The "read on demand" capability is delegated to a generic class, called `OnDemandContainer`

```
template <class T> class OnDemandContainer {
public:
    OnDemandContainer() = default;

    void LoadEvent() {
        if (!cacheIsValid) {
            m_branch->GetEntry(m_treeEntry, true);
            cacheIsValid = true;
        }
    }

    void SetTreeEntry(unsigned long long treeEntry) {
        if (treeEntry == m_treeEntry)
            return;
        m_treeEntry = treeEntry;
        cacheIsValid = false;
    }

    void Branch(TTree *tree) { tree->Branch(T::BranchName.c_str(), static_cast<T *>(this)); }

    void SetBranchAddress(TTree *tree) {
        // ROOT needs the address of the pointer to the buffer object.
        myPtrAddress = static_cast<T *>(this);
        tree->SetBranchAddress(T::BranchName.c_str(), &myPtrAddress);
        m_branch = tree->GetBranch(T::BranchName.c_str());
    }

    T *operator->() {
        LoadEvent();
        return static_cast<T *>(this);
    }

private:
    T *myPtrAddress; //!
    TBranch *m_branch = nullptr; //!
    unsigned long long m_treeEntry = std::numeric_limits<unsigned long long>::max(); //!

    mutable bool cacheIsValid = false; //!
};
```



## READ ON DEMAND

The "read on demand" capability is delegated to a generic class, called `OnDemandContainer`

- ▶ Given a `TTree` it handles the creation/reading of the branch for the corresponding container class

```
template <class T> class OnDemandContainer {
public:
    OnDemandContainer() = default;

    void LoadEvent() {
        if (!cacheIsValid) {
            m_branch->GetEntry(m_treeEntry, true);
            cacheIsValid = true;
        }
    }

    void SetTreeEntry(unsigned long long treeEntry) {
        if (treeEntry == m_treeEntry)
            return;
        m_treeEntry = treeEntry;
        cacheIsValid = false;
    }

    void Branch(TTree *tree) { tree->Branch(T::BranchName.c_str(), static_cast<T *>(this)); }

    void SetBranchAddress(TTree *tree) {
        // ROOT needs the address of the pointer to the buffer object.
        myPtrAddress = static_cast<T *>(this);
        tree->SetBranchAddress(T::BranchName.c_str(), &myPtrAddress);
        m_branch = tree->GetBranch(T::BranchName.c_str());
    }

    T *operator->() {
        LoadEvent();
        return static_cast<T *>(this);
    }

private:
    T *myPtrAddress; //!
    TBranch *m_branch = nullptr; //!
    unsigned long long m_treeEntry = std::numeric_limits<unsigned long long>::max(); //!

    mutable bool cacheIsValid = false; //!
};
```

## READ ON DEMAND

The "read on demand" capability is delegated to a generic class, called `OnDemandContainer`

- ▶ Given a `TTree` it handles the creation/reading of the branch for the corresponding container class
- ▶ Caches the call to `TBranch::GetEntry`, as long as the underlying event didn't change.
- ▶ Exposes member variables of the container class via the `->` operator, and adds the caching behavior.

```

template <class T> class OnDemandContainer {
public:
    OnDemandContainer() = default;

    void LoadEvent() {
        if (!cacheIsValid) {
            m_branch->GetEntry(m_treeEntry, true);
            cacheIsValid = true;
        }
    }

    void SetTreeEntry(unsigned long long treeEntry) {
        if (treeEntry == m_treeEntry)
            return;
        m_treeEntry = treeEntry;
        cacheIsValid = false;
    }

    void Branch(TTree *tree) { tree->Branch(T::BranchName.c_str(), static_cast<T *>(this)); }

    void SetBranchAddress(TTree *tree) {
        // ROOT needs the address of the pointer to the buffer object.
        myPtrAddress = static_cast<T *>(this);
        tree->SetBranchAddress(T::BranchName.c_str(), &myPtrAddress);
        m_branch = tree->GetBranch(T::BranchName.c_str());
    }

    T *operator->() {
        LoadEvent();
        return static_cast<T *>(this);
    }

private:
    T *myPtrAddress; //!
    TBranch *m_branch = nullptr; //!
    unsigned long long m_treeEntry = std::numeric_limits<unsigned long long>::max(); //!

    mutable bool cacheIsValid = false; //!
};

```

## READ ON DEMAND

The only thing left is to attach the "read on demand" capability to each container class, and this is done via inheritance.

The nice part of the trick is that we only need to write and take care of container classes, the "read on demand" part comes almost for free.

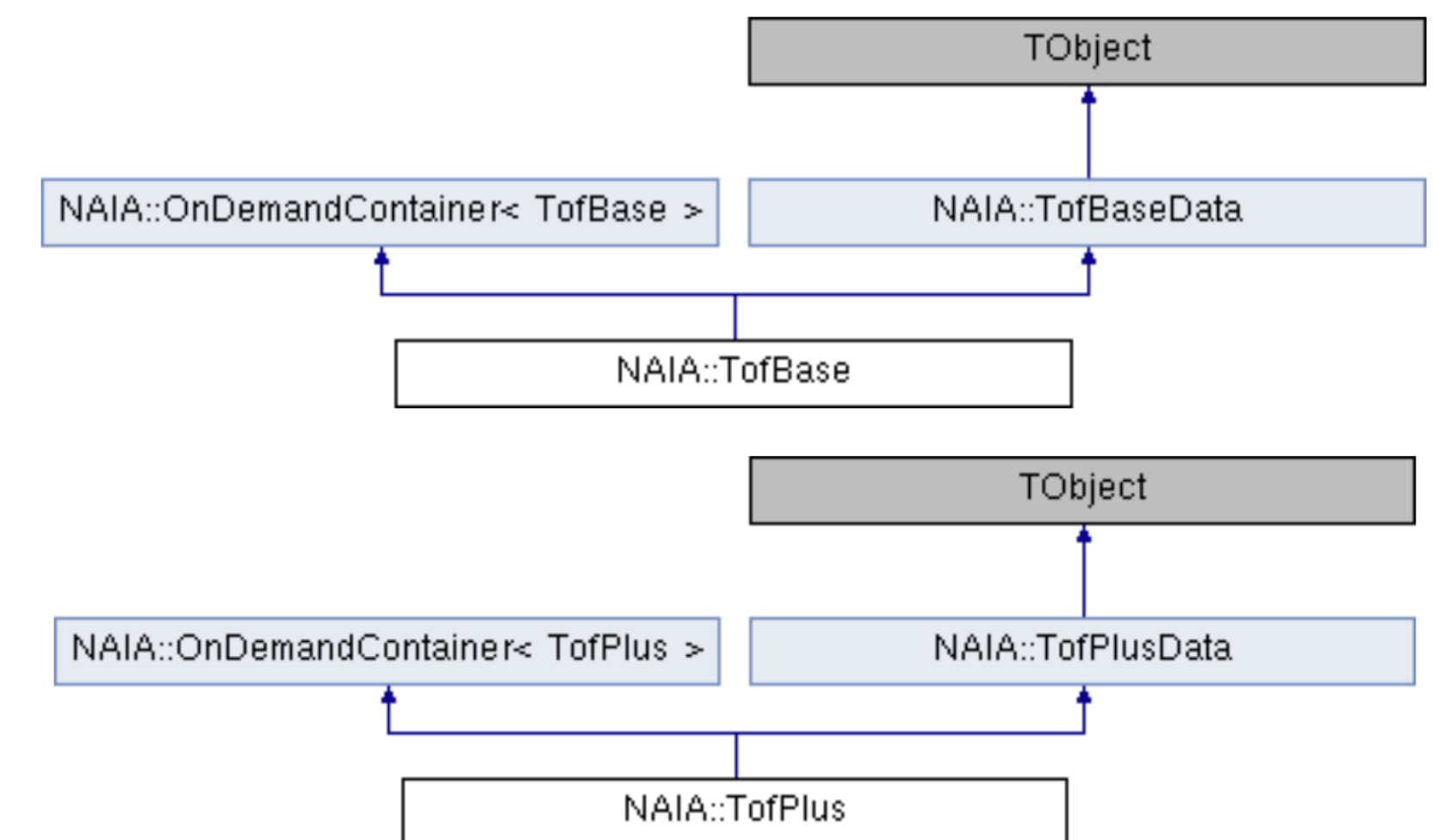
*(as long as we remember to define the corresponding branch name)*

```
class TofBase : public OnDemandContainer<TofBase>, public TofBaseData {
public:
    static const std::string BranchName;

    TofBase() : TofBaseData(), OnDemandContainer() {}
};

class TofPlus : public OnDemandContainer<TofPlus>, public TofPlusData {
public:
    static const std::string BranchName;

    TofPlus() : TofPlusData(), OnDemandContainer() {}
};
```



## SAVING SPACE

- ▶ Should be as light as possible (store only what's available)
  - ▶ We don't want to write missing info.  
(e.g. if no hit on L1, don't write 0 or -9999 or whatever value to keep track of this. We don't want to write anything at all)

This can be achieved by using associative containers (as `std::map`) and trying to find patterns in the requested variables

## SAVING SPACE

- ▶ Should be as light as possible (store only what's available)
  - ▶ We don't want to write missing info.  
(e.g. if no hit on L1, don't write 0 or -9999 or whatever value to keep track of this. We don't want to write *anything at all*)

This can be achieved by using associative containers (as `std::map`) and trying to find patterns in the requested variables:

```
// one number per "layer"
template <class T> using LayerVariable = std::map<unsigned int, T>;
```

you can still do

```
float L1Xpos = event.trTrackPlus->TrTrackHitPosX[0];
```

(but you generally want to check that the required layer info actually exists)

```
// if there is x there is always also y
if (event.trTrackPlus->TrTrackHitPosX.find(0) != end(event.trTrackPlus->TrTrackHitPosX)) {
    L1occupancy->Fill(event.trTrackPlus->TrTrackHitPosX[0], event.trTrackPlus->TrTrackHitPosY[0]);
}
```



## SAVING SPACE

You will find several of these "one value for each  $X$  type" variables, where  $X$  could be charge reconstruction method, track fitting algorithm, ECAL BDT estimator, and so on...

(example for `TrTrack`)

```
// one number per charge reconstruction type
template <class T> using TrackChargeVariable = std::map<TrTrack::ChargeRecoType, T>;
// one number per span type, for each fit type. (Use as Rigidity[fit][span])
template <class T> using TrackFitVariable = std::map<TrTrack::Fit, std::map<TrTrack::Span, T>>;
// one number per for each fit height position. (Use as TrTrackFitPos[heightPos]...)
template <class T> using TrackFitPosVariable = std::map<TrTrack::FitPositionHeight, T>;
```

But using plain numbers to access these values leads often to confusion and butchers readability by other people.

For this reason there are specific enums designed to help with this issue, so that you can write

```
float rig = event.trTrackBase->RigidityCorr[TrTrack::Fit::Kalman][TrTrack::Span::InnerL1];
```

and it's immediately clear which rigidity you are using.

(Of course, we will provide a doxygen page where all this info will be available.)

## AVAILABLE CONTAINERS

- |                     |   |  |                      |   |                                   |   |                |
|---------------------|---|--|----------------------|---|-----------------------------------|---|----------------|
| ▶ Header            | } | Lightweight info: meant to decide if event is interesting or not               | ▶ TrTrackBase        | } | Tracker track variables           |   |                |
| ▶ EventSummary      |   |  | ▶ TrTrackPlus        |   |                                   |   |                |
| ▶ DAQ               |   |  | ▶ TrdKBase           | } | TRD variables                     |   |                |
| ▶ TofBase           | } | Tof variables ("standalone", means <i>reconstructed without tracker info</i> ) | ▶ TrdKBaseStandalone |   |                                   |   |                |
| ▶ TofPlus           |   |  |                      |   | ▶ RichBase                        | } | Rich variables |
| ▶ TofBaseStandalone |   |  |                      |   | ▶ RichPlus                        |   |                |
| ▶ TofPlusStandalone |   |  |                      |   |                                   |   |                |
| ▶ EcalBase          | } | Ecal variables   | ▶ UnbExtHitBase      |   | External layers hits (standalone) |   |                |
| ▶ EcalPlus          |   |  |                      |   |                                   |   |                |
|                     |   |  | ▶ MCTruthBase        | } | MC Truth variables                |   |                |
|                     |   |  | ▶ MCTruthPlus        |   |                                   |   |                |

# EVENT LOOPING

Taking inspiration from gbatch we introduce a **Event** class, but in our case the event is just a collection of containers and nothing more.

The screenshot shows a C++ IDE window titled "NAIA: NAIA::Event Class Refere". The code defines an `Event` class with two methods and two sets of attributes.

```

void SetMaskCategory (Category cat)
    Set the given gategory bit to 1 in the event mask. More...

void SetMC (bool isMC)
    Set wether this is a MC event or not. More...

Public Attributes
    Header header
    EventSummary evSummary
    DAQ daq
    TofBase tofBase
    TofPlus tofPlus
    TofBaseStandalone tofBaseSt
    TofPlusStandalone tofPlusSt
    EcalBase ecalBase
    EcalPlus ecalPlus
    TrTrackBase trTrackBase
    TrTrackPlus trTrackPlus
    TrdKBase trdKBase
    TrdKBaseStandalone trdKBaseSt
    RichBase richBase
    RichPlus richPlus
    UnbExtHitBase extHitBase
    MCTruthBase mcTruthBase
    MCTruthPlus mcTruthPlus

Private Attributes
    bool m_isMC = false
  
```

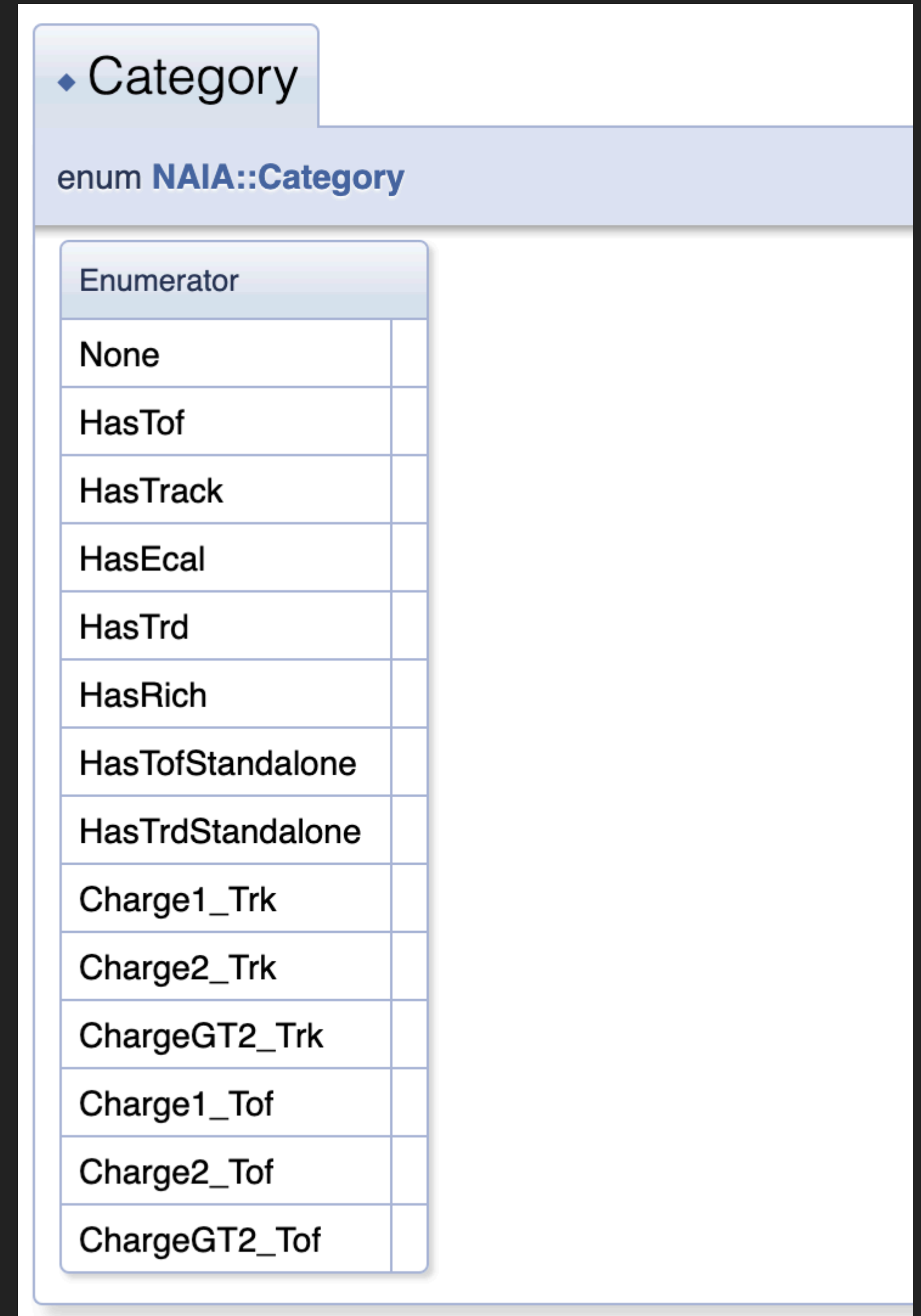
The IDE interface includes a sidebar with navigation icons (Home, Star, Home, Messages, WhatsApp, Instagram, Twitter, Play, Arrow, Heart, Clock, Gear, Lightbulb) and a top toolbar with various icons and a search bar. The browser-like address bar shows the file path: `file:///Volumes/AMS_Disk/`.

## EVENT LOOPING

Taking inspiration from gbatch we introduce a `Event` class, but in our case the event is just a collection of containers and nothing more.

The `Header` container also stores a bit-mask that will encode whether the event belongs or not in a given set of categories. This will be extremely helpful to skip uninteresting events without having to read anything else.

*(full list of categories to be finalized)*



The screenshot shows a software interface with a tab labeled "Category". Below the tab, the text "enum NAIA::Category" is displayed. A table with the following structure is shown:

Enumerator	
None	
HasTof	
HasTrack	
HasEcal	
HasTrd	
HasRich	
HasTofStandalone	
HasTrdStandalone	
Charge1_Trk	
Charge2_Trk	
ChargeGT2_Trk	
Charge1_Tof	
Charge2_Tof	
ChargeGT2_Tof	

## THE CHAIN

The `Event` object is always provided by a `Chain` class.

Also here the approach is derived from `gbatch`. The idea is that you just have to:

- ▶ Declare a chain
- ▶ Add files to it
- ▶ Loop and get each event -> do analysis on it

```
using namespace NAIA;

int main(int argc, char const *argv[]) {
    NAIAChain chain(NAIA::SingleTreeChain::AccessMode::Read);
    chain.Add("test.root");

    chain.SetupBranches();

    unsigned long long nEntries = chain.GetEntries();
    spdlog::info("{} entries in the chain", nEntries);

    for (unsigned long long iEv = 0; iEv < nEntries; iEv++) {
        spdlog::info("Entry {}", iEv);

        auto event = chain.GetEvent(iEv);

        chain.GetEventFileInfo().Dump();
        chain.GetEventRTIInfo().Dump();

        fmt::print("Mask: {}\n", to_string_binary<32>(event.header->Mask()));
        event.header->Dump();
    }

    return 0;
}
```



## THE CHAIN

In addition to the event tree we also store two smaller trees with the RTI info for each run, and the info of the original file that was processed

You can either get the RTI/File info for the current event, or get the whole RTI/File info tree and loop on it yourself (avoiding a loop on the events)

```
using namespace NAIA;

int main(int argc, char const *argv[]) {
    NAIAChain chain(NAIA::SingleTreeChain::AccessMode::Read);
    chain.Add("test.root");

    chain.SetupBranches();

    unsigned long long nEntries = chain.GetEntries();
    spdlog::info("{} entries in the chain", nEntries);

    for (unsigned long long iEv = 0; iEv < nEntries; iEv++) {
        spdlog::info("Entry {}", iEv);

        auto event = chain.GetEvent(iEv);

        chain.GetEventFileInfo().Dump();
        chain.GetEventRTIInfo().Dump();

        fmt::print("Mask: {}\n", to_string_binary<32>(event.header->Mask()));
        event.header->Dump();
    }

    return 0;
}
```

# SUPPORT INFRASTRUCTURE

Lastly, the code is hosted on the CERN gitlab instance, under the AMS-Italy group name.

The screenshot shows the GitLab web interface for the NAIA project. The browser address bar displays `gitlab.cern.ch/ams-italy/naia`. The page header includes navigation links for Projects, Groups, and More. The main content area shows the project details for NAIA (Project ID: 86658), including statistics for 110 Commits, 2 Branches, 0 Tags, 512 KB Files, and 1.3 MB Storage. A recent merge commit is visible: 'Merge branch '17-test-usability-by-external-projects' into 'master'' by Valerio Formato, authored 1 week ago. Below the merge, there are buttons for adding CI/CD configuration, README, LICENSE, CHANGELOG, CONTRIBUTING, and a Kubernetes cluster. A table at the bottom lists the project's files and their last commit details.

Name	Last commit	Last update
Modules	[Core] First commit. Early prot...	1 year ago
doc	[Build] Remove usage of bare ...	1 week ago
include	Merge remote-tracking branc...	1 week ago
setenvs	Bump cmake version in setenv...	11 months ago

## SUPPORT INFRASTRUCTURE

Lastly, the code is hosted on the CERN gitlab instance, under the AMS-Italy group name.

Compilation is tested for each commit on a generic CentOS7 environment (gcc 7.3 + ROOT 6.18/04)

A setenv is available for this environment, gcc and ROOT provided on cvmfs

Plan to create a dedicated AMS-Italy cvmfs at CERN

The screenshot displays the GitLab Pipelines interface for the project 'AMS-Italy / NAIA'. The page shows a list of pipeline runs with the following columns: Status, Pipeline, Triggerer, Commit, and Stages. The status of each run is indicated by a green checkmark for 'passed' or a red 'x' for 'failed'. The pipeline runs are as follows:

Status	Pipeline	Triggerer	Commit	Stages
passed	#2349006 latest	[Avatar]	master -> 09b81ea9 Merge branch '17-tes...	00:02:31 1 week ago
passed	#2348993	[Avatar]	17-test-usa... -> dd570c08 [Build] Permanently fi...	00:02:25 1 week ago
failed	#2348951	[Avatar]	17-test-usa... -> 1d9c6ab4 [Build] Fix compilatio...	00:01:25 1 week ago
failed	#2348833	[Avatar]	17-test-usa... -> a5950f62 [Build] Fix installation...	00:01:39 1 week ago
failed	#2348366	[Avatar]	17-test-usa... -> dc69c3f5 [Build] Export targets...	1 week ago
passed	#2347812	[Avatar]	17-test-usa... -> eee1758e [Build] Remove usage...	00:02:56 1 week ago
passed	#2347428	[Avatar]	17-test-usa... -> bc1c21e8 [Build] Don't compile ...	00:01:45 1 week ago
failed	#2347411	[Avatar]	17-test-usa... -> cfd38d4e	00:00:55 1 week ago

## SIMPLE SETUP

Everything is provided by two libraries:

- ▶ `libNAIACChain.so`
- ▶ `libNAIACContainers.so`

Link/load those and that's it.

If you plan on using CMake the setup consists of just three lines:

```
find_package(NAIA REQUIRED)

add_executable(testReadNtp src/testReadNtp.cpp)
target_link_libraries(testReadNtp PUBLIC NAIA::NAIACChain)
```



## BONUS TRACK: AMS-ITALY DISCORD SERVER

We had this setup for the event on Parmitano's reentry from the ISS.

On this server there is a channel dedicated to the DST topic. That is the recommended communication channel for questions/help with the DST.

(We can (should) use this server also for all other kinds of AMS-Italy related communications)



The screenshot displays the Discord interface for the AMS-Italia server. The left sidebar shows the server's structure with categories: CANALI TESTUALI (containing #generale and #commentiamo\_il\_gm), SOFTWARE (containing #ntuple-management), ANALYSIS, and CANALI VOCALI (containing Generale). The main window shows the #ntuple-management channel with a message history. The messages are from Valerio Vagelli and Valerio Formato, dated 07/07/2020 and 02/25/2021. The messages discuss the DST topic, mentioning the beta version of ntuple and the need for a beta version. The right sidebar shows the server's member list, including ADMIN-2 (bozzochet, Valerio Formato), ONLINE-3 (Federico, Maura, Valerio Vagelli), and OFFLINE-11 (BrunaB, Francesco Dimiccoli, Giorgione, Giuseppe La Vacca, Glenda, Lorenzo Mussolin, maurotacconi, Paolo Zuccon, Stefano, Valeria Di Felice).



## NEXT STEPS

Project is now ready for beta testing. All the functionalities requested initially have been implemented.

5000 random pass7 runs processed at CNAF

I would like to call a kickoff meeting with at least one designated tester from each analysis group to:

- ▶ Go in detail on the (almost nonexistent) setup required to get up and running on the new data
- ▶ Collect feedback / feature requests after each tester tries to reproduce a small portion of his analysis on the new DST

After this testing phase the first release will be tagged and full production of pass7 and MC samples will begin.

