Introduction to the SHOE framework

Yunsheng Dong

11/01/2025

How to start

- The software is in a git repository hosted on balgit.infn.it

Instructions about the installation of SHOE are provided in the wiki: https://baltig.infn.it/asarti/shoe/-/wikis/SHOE, the main passages are:

- Register on the git lab portal (baltig).
- Ask to be added by a shoe maintainer (just send an email to yunsheng.dong@mi.infn.it or roberto.zarrella@bo.infn.it)
- Fulfill the prerequisites: gcc>=8, properly installed and working ROOT
- For some old ROOT versions <6.28 there could be problems with specific packages. Check the output of "root-config -features" and check that "eve" and "minuit2" libraries are installed and c++17 standard is enabled.
  If not, you can enable them in the root builddir with:
  "cmake pathtoroot -Drpath=ON -Dminuit2=ON -DCMAKE_CXX_STANDARD=17 -DCMAKE_BUILD_TYPE=Debug -Dbuiltin_vdt=ON"
- Download the code with the following command:
  git clone https://baltig.infn.it/asarti/shoe.git
- Compile with:
  mkdir build
  cd build
  cmake path_to_shoe -FILECOPY=ON
  make

How to start

- There is the possibility to use tier1 resources to compile and run SHOE, instructions are in the wiki: https://baltig.infn.it/asarti/shoe/-/wikis/Access%20to%20Tier%201

- You can scp or mount tier1 folder to access to both simulation and data, instructions are in the wiki: https://baltig.infn.it/asarti/shoe/-/wikis/Data%20location%20on%20Tier%201

- Once SHOE is compiled, you can try to run the code on a MC sample and/or on data:
  eg.: “../bin/DecodeGlb -in inputrawfilename -exp experimentname -run runnumber”
  (N.B.: if you are using a MC file, a “-mc" flag must be added, other useful flags are explained with “../bin/DecodeGlb -help” command)
  If everything worked… Enjoy your SHOE reconstructed file!

- You can grab a MC simulation or raw data file from tier1… guess where are the instructions…
  (https://baltig.infn.it/asarti/shoe/-/wikis/Data%20location%20on%20Tier%201)

Execution parameters

- Before the execution of the code, one need to define different parameters/flags

- There are parameters defined with the execution command (../bin/DecodeGlb -help)

```
ymac@mactiger Reconstruction % ./DecodeGlb -help
 Decoder help:
 Ex: Decoder [opts]
 possible opts are:
      -in path/file   : [def=] raw input file
      -out path/file : [def=*_Out.root] Root output file
      -nev value      : [def=10^7] Numbers of events to process
      -nsk value      : [def=0] Skip number of events
      -run value      : [def=-1] Run number
      -exp name       : [def=] experient name for config/geomap extention
      -mc             : reco from MC local reco tree
      -inmc           : MC file name ONLY FOR TOE
      -subfile        : [def=false] when true disable the processing of the chain of all the sub file related to a given run: only the subfile related to the input file is processed
      -mth            : enable multi threading (for clustering)
```

- = mandatory

- All the FOOT experiment names and run numbers can be found in builddir/Reconstruction/cammaps/FOOT.cam, each campaign has his own campaign file in which all the subdetector config/calib/map files are defined
more details here: https://agenda.infn.it/event/23332/contributions/116692/attachments/73616/93274/CampManager.pdf

-

FOOT.cam:

```
CamNumber:    42
CamName:     "12CFull24_MC"
CamDataMC:    1
CamDate: "September 2024"
CamSum: "Simulation data for a full detector with BM+ST+VT+IT+MSD+TW+CA, Magnets and pas

CamNumber:    43
CamName:     "CNAO2024"
CamDataMC:    0
CamDate: "November 2024"
CamSum: "Test experiment at CNAO with ST+VT+IT+MSD+TW+CAL+Magnets @ 200,400 MeV/u"
```

12Full24_MC.cam:

```
// Campaign file
CamName: "12CFull24_MC"
RunNumber: 200;201;202;400;401;402
NumberDevices: 10

DetectorName: "FOOT"
NumberFiles: 2
"./geomaps/12CFull24_MC/FOOT.geo": 200;201;202;400;401;402
"./geomaps/12CFull24_MC/FOOT.reg": -1

DetectorName: "DI"
NumberFiles: 1
"./geomaps/12CFull24_MC/TADIdetector.geo": -1
```

Execution parameters

```
Debug:  0

###########    MC selection Control Parameters ###############
MC Particle Types: H1 H2 H3 He3 He4 He6 He8  Li6 Li7 Li8 Li9  Be7 Be9 Be10

Genfit Event Display ON: n


###########   Global Reconstruction Parameters   ###############

IncludeKalman:   y
IncludeTOE:      n
IncludeStraight: n
FromLocalReco:   n

###########   Kalman Filter Control Parameters   ###############

Kalman Mode:       ref
Tracking Systems Considered:    all
#Tracking Systems Considered:    VT IT MSD TW
Reverse Tracking:       false

Kalman preselection strategy: Standard

Chi2 cut: -1
N measure in global tracking: 9

Kalman Particle Types:  C

###########    END - Kalman Filter Control Parameters  ###############
```

Legenda: red: usually not to be changed
BOLD: something you should check carefully

Other "global" parameters are set in builddir/Reconstruction/config/<expname>/FootGlobal.par

- Debug flag with values [0,4], higher values=more debug messages
- MC particle types: for global tracking efficiency and other calculations
- Genfit Event Display ON: if y, it will open the genfit event display. N.B.: there is a foot ROOT based event display available with DisplayFOOT.C and DisplayMcFOOT.C
- **IncludeKalman**: enable the global track reconstruction based on the genfit kaman filter (developed and in an optimisation phase)
- IncludeTOE: enable the global track reconstruction based on a foot Kaman filter (development stalled few years ago)
- IncludeStraight: enable the global track reconstruction based on a simple extrapolation of the vtx tracks in Z towards the TW and attachment of all the IT/MSD/TW hits close to the track (developed, cannot be used for events with magnetic field)

Execution parameters

```
Debug:  0

###########    MC selection Control Parameters ################
MC Particle Types: H1 H2 H3 He3 He4 He6 He8  Li6 Li7 Li8 Li9  Be7 Be9 Be10

Genfit Event Display ON: n


###########   Global Reconstruction Parameters   ################

IncludeKalman:   y
IncludeTOE:      n
IncludeStraight: n
FromLocalReco:   n

###########   Kalman Filter Control Parameters   ################

Kalman Mode:      ref
Tracking Systems Considered:    all
#Tracking Systems Considered:    VT IT MSD TW
Reverse Tracking:       false

Kalman preselection strategy: Standard

Chi2 cut: -1
N measure in global tracking: 9

Kalman Particle Types:  C

###########    END - Kalman Filter Control Parameters  ################
```

Legenda: red: usually not to be changed
BOLD: something you should check carefully

- FromLocalReco: to reconstruct global tracks starting with an already shoe processed file with local reconstructed quantities (known issue: it's under fix)
- Kalman mode: "off", "on", "ref", "daf", "dafsimple": these are the kaman mode available in Genfit, the standard one is "ref"
- Tracking systems considered: all, or write each single detector that should be considered by the global reco algorithm
- Reverse tracking: activate the reverse tracking
- Kalman preselection strategy: **"TrueParticle"**, **"Standard"**, "Linear", "OutsideIn", "Backtracking": strategy adopted to separate the hits of different tracks. (TrueParticle means use MC info and perform a perfect separation, Standard is the method based on the extrapolation of the VTX tracks towards the other detectors and associate the hits close to each track)
- Chi2 cut: cut on the global track chi2 value, -1=not used
- **N measure in global tracking:** minimum number of hits required for a track to be considered valid

Execution parameters

```
###########   TOE Control Parameters   ################

TGT Tag:        C
VTX Tag Cuts:  15  21
IT Tag Cuts:   73  51  77  61
MSD Tag Cuts:   9   6   6   4  4  2
MSD2 Tag Cuts: 25
TOF Tag Cuts:  12  15

###########   END - TOE Control Parameters   ################

###########   Options for reconstruction   ################

EnableTree:          y
EnableFlatTree:      n
EnableHisto:         y
EnableTracking:      y

EnableSaveHits:      n
EnableRootObject:    y
EnableRegionMc:      y
EnableElecNoiseMc:   y

###########   END - Options for reconstruction   ################

IncludeDI:                        y
IncludeST:                        y
IncludeBM:                        y
IncludeTG:                        y
IncludeVT:                        y
IncludeIT:                        y
IncludeMSD:                       y
IncludeTW:                        y
IncludeCA:                        y

FLUKA version: pro
```

- TOE: is the foot “home made” kalman reconstruction software developed few years ago in parallel with the genfit kalman code. At the moment, we suggest to use the genfit based Kalman filter.
- **EnableTree:** write in the output file a ttree containing the shoe ntuple necessary for further analysis (recommended)
- EnableFlatTree: write in the output file a ttree containing only “simple” variables (double, int, tvector3) representing some of the relevant reconstructed quantities (e.g.: track parameters) **NOT recommended**
- **Enable histo:** write in the output file all the histograms created in the the “action” classes (recommended)
- **EnableTracking:** activate the tracking. N.B.: detector local tracking flags are in config/<expname>/TA*detector.cfg
- **EnableSaveHits:** write in the output file all the hits ntuple, otherwise only tracks and points will be saved (recommended)
- EnableRootObject: only for MC files, it gives the possibility to read also fluka structure files (if “n”). At present, all the simulation are provided as files of root object (so always set on “y”)

Legenda: red: usually not to be changed
BOLD: something you should check carefully

Execution parameters

```
###########   TOE Control Parameters   ################

TGT Tag:        C
VTX Tag Cuts:  15  21
IT Tag Cuts:   73  51  77  61
MSD Tag Cuts:   9   6   6   4  4  2
MSD2 Tag Cuts: 25
TOF Tag Cuts:  12  15

###########   END - TOE Control Parameters   ################

###########   Options for reconstruction   ################

EnableTree:          y
EnableFlatTree:      n
EnableHisto:         y
EnableTracking:      y

EnableSaveHits:      n
EnableRootObject:    y
EnableRegionMc:      y
EnableElecNoiseMc:   y

###########   END - Options for reconstruction   ################

IncludeDI:                          y
IncludeST:                          y
IncludeBM:                          y
IncludeTG:                          y
IncludeVT:                          y
IncludeIT:                          y
IncludeMSD:                         y
IncludeTW:                          y
IncludeCA:                          y

FLUKA version: pro
```

- **EnableRegionMC:** write in the output file also the MC crossing ntuple
- **EnableElectricNoiseMC:** add noise effect to the MC simulation
- **Include*:** include different detectors in the analysis, there is the possibility to skip the reconstruction of a detector even if it is present in the campaign to speed up the process.
  DI=magnetic dipole
  ST= Start counter
  BM= Beam Monitor
  TG= Target
  VT= Vertex
  IT= Inner tracker
  MSD= Microstrip silicon detector
  TW= TofWall
  CA= Calorimeter
- Fluka version: select the fluka version of the simulation (pro or dev). This is used only by who need to create a FLUKA simulation

Legenda: red: usually not to be changed
BOLD: something you should check carefully

General information

- The code is written in C++, based on ROOT
  you can find bash, fortran and python scripts for dedicated tasks not involved in the main workflow

The full event reconstruction proceeds in steps:

- Depending on the flags of the input command, the code will start to load the configuration/calibration/mapping/geometry files

- starting from the decoding of the data/MC 'raw' information the different subdetectors process the information and 'complex' detector specific objects are built in a sequence of actions
  E.g. for the Vertex detector (VT):
  – MC/data provide the number and position of pixels that are fired for a given event
  – pixels are combined in clusters
  – clusters are combined to reconstruct tracks
  – Tracks are extrapolated to the target Z position and combined into vertices if they are spatially close
  – The vertex that is closest to the BM track projection is selected as the BM-matched vertex

- If the global reconstruction Kalman flag is activated, each VT track that belong to a BM-matched vertex is extrapolated towards the other trackers and a global reconstruction algorithm will reconstruct the whole fragment track creating a global track object with charge estimate

Libraries

In shoe/libs/src (not builddir, in the downloaded folder!) there are all the libraries organised in folders:

-**TABMbase**: Beam Monitor (Yunsheng Dong)
-**TACAbase**: Calorimeter (Laura Buonincontri?)
-**TATWbase**: Tof-Wall (Marco Toppi)

-**TAVTbase**: Vertex (Christian Finck)
-**TAMSDbase**: MSD (Ilaria Mattei?)
-**TASTbase**: Start Counter (Giacomo Traini)
-**TAIRbase**: Interaction Region (Christian Finck)
-**TAITbase**: Inner tracker (Christian Finck)
-**TADIbase**: Magnetic field
-**TAEDbase**: Event display
-**TAGbase**: base libraries, a lot of other classes are inherited from classes declared here. In particular: GlobalPar: that contains all the global parameters and TAGcampaignManager class that handle all the campaign parameters
-**TAGdaq/TAGdaqApi**: Daq libraries to read data
-**TAGfoot**: general foot libraries used to manage the main event loop

-**TAMCbase**: MC libraries related to the MC simulation

-**TOE**: TOE global reconstruction libraries
-**TAGFbase**: global reconstruction based on genfit libraries
-**GenFit**: Genfit libraries copied from (https://github.com/GenFit/GenFit)

Libraries

- **Analysis libraries are in shoe/libs/Analysis**: We (mainly G.Ubaldi and C. Finck) created this libraries few months ago, so it is in a development and testing phase
- There is a folder with libraries dedicated to ancillary detector studies in shoe/Ancillary

Other relevant files are in shoe/Reconstruction folder

- The executable codes are saved in shoe/Reconstruction (e.g.: DecodeGlb.cc, Calibrate* etc.)
- In macros/ pyscripts/ scripts/ folders you can find different macros/python_scripts/bash_scripts useful for analysis/ calibration etc.
- There are different geometry, calibration, configuration folders with all the parameter files necessary to execute shoe
- When you execute “cmake /path/to/shoe -D filecopy=ON” in your builddir, all the geo/config/cal/macros/scripts that are in the shoe/Reconstruction folder will be copied in your builddir/Reconstruction folder
  N.B.: if you have modified one of the previous file, it will be overwritten!

Global workflow

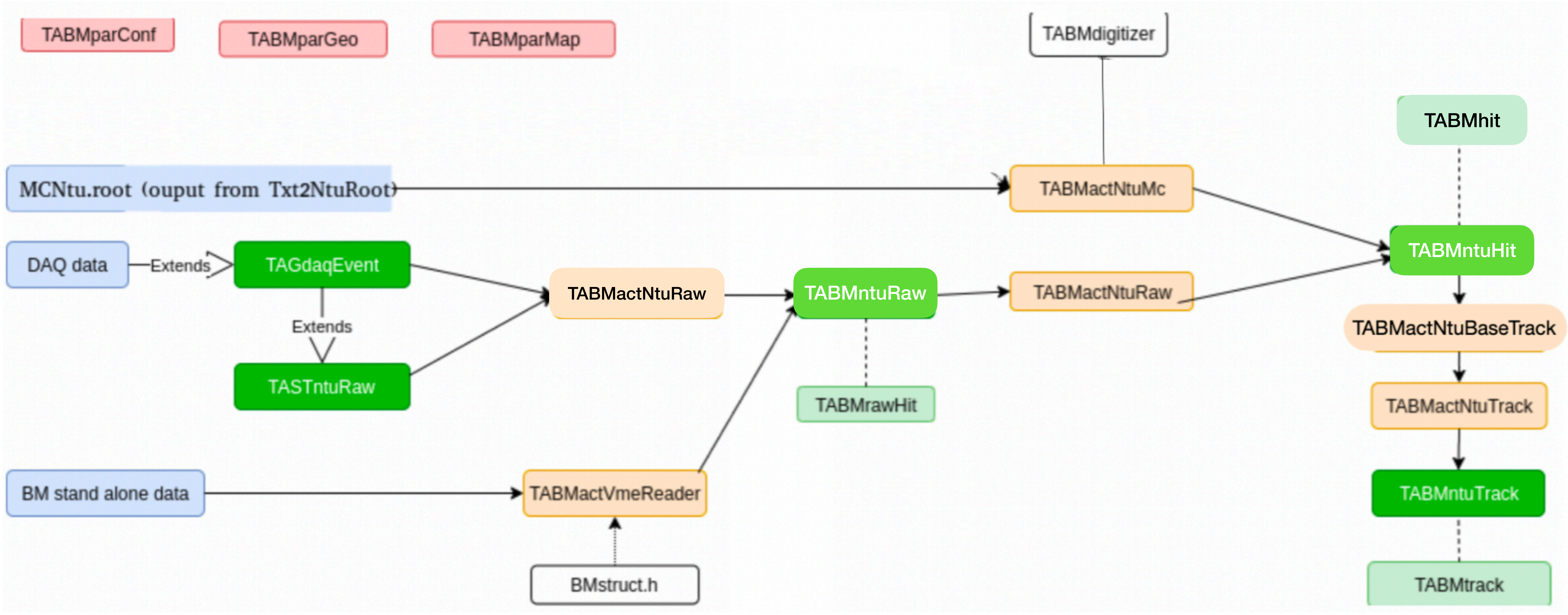
```
//! Actions before loop event
void BaseReco::BeforeEventLoop()
{
   GlobalSettings();

   ReadParFiles();

   CampaignChecks();

   CreateRawAction();
   CreateRecAction();

   AddRequiredItem();

   OpenFileIn();

   GlobalChecks();

   if (fFlagOut)
      OpenFileOut();

   fTAGroot->BeginEventLoop();
   fTAGroot->Print();
}
```

- The global reconstruction main code is in Reconstruction/DecodeGlb.cc
- it relies on libs/src/TAGfoot/**RecoMC**, **RecoRaw** for local reco, and on **GlobalToeReco, GlobalReco** for global track reconstruction
- They are all inherited from libs/src/TAGfoot/**BaseReco**
  The workflow is composed only of these three lines:

```
glbRec->BeforeEventLoop();
glbRec->LoopEvent(nTotEv);
glbRec->AfterEventLoop();
```

- These methods are developed in RecoMC/Raw, Global*Reco and BaseReco, depending if they are data/MC dependent or not.
- **BaseReco::BeforeEventLoop()** → read the campaign file and the detector parameter filesloads the geometry, calibration, configuration files.. and creates all the objects needed during the loop and the related actions. It also handles the 'reading' of the input file, create the "raw" and "rec" actions and begin the event loop. Here the methods are developed directly in BaseReco or in the Reco*/Global*Reco libraries
  (e.g.: ReadParFiles and CreateRecAction in BaseReco, CreateRawAction in Reco*/Global*reco).
- **BaseReco::LoopEvent()** → it's just a simple loop developed in BaseReco, It 'only' calls 'NextEvent()' the method that triggers the sequential running of all the Actions() defined in before EL
- **BaseReco::AfterEventLoop()** → end function, developed in BaseReco, save the output and close files
- **N.B.: BaseReco is in common both for MC and data processing. Be careful in the modification of this library! You need to grant the compatibility with both MC and Data processing (also back compatibility with old data takings) and foresee all the possible cases that a user can provide with the execution parameters (of course, be careful also in the modification of other libraries)**

An example: the BM reco workflow



Input file Parameters actions Ntuples Others

Parameters

Parameter files: derived from **TAGparaDSC and TAGparTools**: they are the parameter files used in a lot of other classes and methods
Each TAGparaDSC class reads an input file to charge the values. **The input file position can depends on the -exp flag, according to the campaign manager** (link1, link2).

TABMparGeo: to manage the geometry

- FromFile method to read the materials and the geometry parameters written in TABMdetector.geo file placed by default in builddir/Reconstruction/geomaps/expname/.
- InitGeo() called after FromFile and before the event loop to define the detector geometry
- BuildBeamMonitor() method used to create the BM in the FOOT sys. of reference.
  For other detector the method is Build*()
- There are different PrintBodies(), PrintRotations(), PrintRegions(), PrintAssignMaterials() Print* that are used in Simulation/Makegeo.cxx to write the FLUKA .geo file for the simulations
- Each detector has a own **local** system of reference centred in (0.,0.,0). All the hits/tracks related to a specific detector is defined in the detector local frame.
- Each detector has different components (e.g.: sensor, wire etc.) and there are different methods (e.g.: Detector2Wire, Wire2Detector, Detector2WireVect et.) to retrieve the component position, to pass from the detector local frame to the component frame and vice-versa
- Then, each detector is placed in the FOOT global frame. The method to retrieve the detector position and to convert the local coordinate and vectors into the **global** frame are in TAGbase/TAGgeoTrafo (VecFromBMLocalToGlobal, FromBMLocalToGlobal etc.)

Parameters

TABMparConf: manage the configuration parameters

- There are different methods to set and retrieve the track reconstruction parameters (e.g.: tolerance, fit stepsize), MC threshold, smearing etc.
- It read the input parameters from TABMdetector.cfg with the method FromFile. The input file default position is in builddir/Reconstruction/config/expname/

TABMparMap: manage the mapping of data to the detector parameters (usually not used for MC)

- In the case of the BM, TABMparMap is used to map the TDC channels into the BM detector internal channel code. In addition, it is used to define the ADC and Scaler parameters for the stand alone data processing.
- The default file name is TABMdetector.map and the default file position is: builddir/Reconstruction/config/expname

TABMparCal: The TABMparCal is used for the calibration parameters related to each detector and each campaign/run

- For the BM, it is used to set and retrieve the BM space-time relations, T0 values, resolution function
- The default file position is in: builddir/Reconstruction/calib/expname and the default file name is: TABM_T0_Calibration.cal.
  For the BM this file can be run dependant, so the typical file name is TABM_T0_Calibration_runnumber.cal

Ntuples or TAGdata

TAGdata: They are objects used to represent a detector hit, track etc.. They can be divided in:

- **Containers:** (TABMntuRaw, TABMntuHit, TABMntuTrack) in which there is a **TClonesArray*** used to store all the hits/tracks and the methods to add/retrieve the single hit/track element. In addition there are methods related to all the hits/tracks sample (eg.: efficiency calculation, retrieve number of hits etc.).
- **Hit/tracks**: (TABMrawHit, TABMntuHit, TABMtrack) Here each hit /track is defined.
  TABMntuRaw—>TABMrawHit is the hit taken from the TDC without any cut or calculation, just raw time and cell infos. This class is constructed only if real data are processed.
  TABMntuHit—>TABMhit is the BM hit. Here the drift distance,resolution, chi2 are saved. From this class, there are no differences between MC and data.
  TABMntuTrack—>TABMtrack is a BM single track. Here all the track parameters can be retrieved (direction, chi2 etc.) Also in this case, no difference between MC and data.
- They are created by a specific TAGaction class
- The convention is: TA*actSomethingNtu* create an element with similar name TA*SomethingNtu*
- **In a global analysis framework, these are the elements that are useful for analysis**

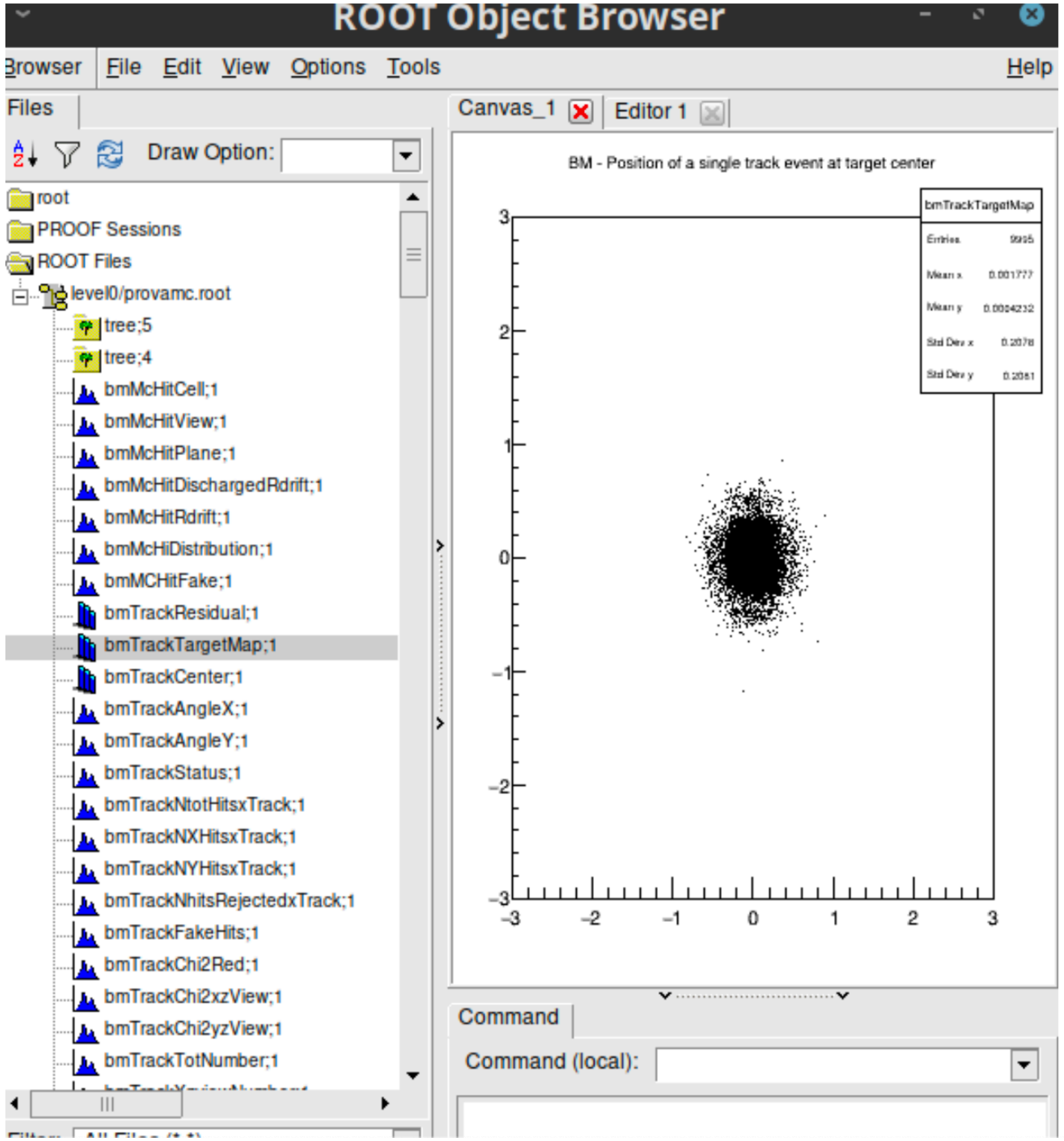
Actions

```
//------------------------------------------------------------------------------
//! Default constructor.
//!
//! \param[in] name action name
//! \param[out] dscntutrk track output container descriptor
//! \param[in] dscnturaw raw data  intput container descriptor
//! \param[in] dscbmgeo geometry parameter descriptor
//! \param[in] dscbmcon configuration parameter descriptor
//! \param[in] dscbmcal calibration parameter descriptor
TABMactBaseNtuTrack::TABMactBaseNtuTrack(const char* name,
                                         TAGdataDsc* dscntutrk,
                                         TAGdataDsc* dscnturaw,
                                         TAGparaDsc* dscbmgeo,
                                         TAGparaDsc* dscbmcon,
                                         TAGparaDsc* dscbmcal)
  : TAGaction(name, "TABMactBaseNtuTrack - Track finder for BM with Legendre method"),
    fpNtuTrk(dscntutrk),
    fpNtuHit(dscnturaw),
    fpParGeo(dscbmgeo),
    fpParCon(dscbmcon),
    fpParCal(dscbmcal)      Yunsheng Dong, 19 months ago • update BM tracking
{
  if (FootDebugLevel(1))
   cout<<"TABMactBaseNtuTrack::default constructor::Creating the Beam Monitor Track ntuplizer"<<endl;

  AddDataIn(fpNtuHit,  "TABMntuHit");
  AddDataOut(fpNtuTrk, "TABMntuTrack");
  AddPara(fpParGeo,  "TABMparGeo");
  AddPara(fpParCon,  "TABMparConf");
  AddPara(fpParCal,  "TABMparCal");
```

TAGaction (TABMactNtuHitMC, TABMactNtuMC, TABMactDatRaw, TABMactNtuRaw, TABMactNtuTrack): -

- Each TAGdata element is created by a TAGaction class.
- Each TAGaction contains an Action() method that start from a TAGdata and it creates a new TAGdata.
  (e.g.: TABMactNtuTrack take as input the TABMntuHit contained in TABMntuRaw and creates TABMtrack contained in TABMntuTrack)
- The Action() is called at each Next event iteration (Remember the Eventloop in BaseReco?)
- the input and output parameters are usually listed in the constructor.
  e.g.: TABMactBaseNtuTrack:
  -AddDataIn(fpNtuHit,"TABMntuHit");
  -AddDataOut(fpNtuTrk,"TABMntuTrack");
  -AddPara(...) are the parameters used by this class. In this case, it needs the geometry and the configuration classes.
- In addition to Action(), usually there is a CreateHistogram() method in which all the histogram of interest are created. They are filled inside a if(ValidHistogram()) structures.
  you'll see all these histograms in the output file if the EnableHisto flag is activated

Output file



- If the EnableHisto flag is activated, you should find in the output file different **histograms** defined in the TAGaction classes.
- Here, one can check if everything worked properly or if there is something wrong/strange
- These histos can be directly read by root without any requirements
- If the EnableTree flag is activated, you should also find also a root tree useful for further analysis.
- In order to read correctly the ttree, ROOT need to know the SHOE libraries that are charged with the rootlogon.C file present in builddir/Reconstruction

```
*******************************************************************
                    Messages from rootlogon.C
*******************************************************************
                    Load libraries for Reconstruction
*******************************************************************
- Add /Users/ymac/myFOOT/shoes/shoe_main/builddir/Reconstruction/ to the macros path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAGbase to the include path
- Add  /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAMCbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TADIbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAGdaq to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAGdaqApi to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TASTbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAIRbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TABMbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAVTbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAITbase to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAMSDbase to the include path
- Add  /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TATWbase to the include path
- Add  /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TACAbase to the include path
- Add  /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAEDbase to the include path
- Add GenFit to the include path
- Add /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAGFbase to the include path
- Add  /Users/ymac/myFOOT/shoes/shoe_main/Libraries/TAGfoot to the include path
root [0]
```

Analysis macro

- You can use a macro to read the shoe DecodeGlb output ttree.
  In this way you can use the shoe reconstructed objects to perform your analysis, without the need to re-execute the whole reconstruction process each time.
  Analysis macro are recommended to perform short, specific analysis tasks:
  Eg.: Evaluate detector performances or calculate detector parameters.
- if the task gets too complicated.. best to go for executables that inherit from BaseReco (see in next slides)
- **N.B.: be careful!** you'll develop your macro in your builddir/Reconstruction/ folder, but if you want to add the macro in the online repository, you have to copy this macro in the shoe/Reconstruction/macros folder, add the macro explicitly in git and then push.
- **N.B.: be careful 2!** if in the initial cmake command you add the -D FILECOPY=ON flag, then the make command will copy all the macros contained in the online repository in your builddir/Reconstrcution/folder.
  your changes will be overwritten
- There are different macros that are already in the online repository.
  E.g.1: AlignFOOTFunc.h and AlignFOOTMain.C are used to align the FOOT sub detectors
  it can be executed with the following command:
  root -l -b 'AlignFOOTMain.C+("reco4314Full_21nov_TWBarCalib.root",0,0,true,true,true,true)'

  E.g.2: there are different BM related macros that are used to calibrate the space time relations, to evaluate T0, spatial resolution etc.

Analysis macro template

```cpp
#include "ReadShoeTreeFunc.h"

// nameFile=Input file name
// entries: number of events to be processed (use 0 to process the whole file)
// printFile: redirect cout to an external txt file
void ReadShoeTreeMain(TString nameFile = "", Int_t nentries = 0, Int_t debug_in=0)
{

  debug=debug_in;
  if(OpenInputFile(nameFile))
    return;
  if(ChargeCampaignParameters())
    return;
  ChargeParFiles(nentries);
  SetOutputFiles(nameFile);
  BookHisto();

  cout<<"input file="<<nameFile.Data()<<endl;
  cout<<"I'll process "<<maxentries<<" events. The input tree contains a total of "<<tree->GetEntries()<<" events."<<endl;

  //event loop
  for (evnum = 0; evnum < maxentries; ++evnum) {

    if(evnum%100==0)
      printf("Processed Events: %d\n", evnum);

    tree->GetEntry(evnum);

    if(IncludeVT)
      Vertex();
    if(IncludeGLB)
      GLBTRKstudies();
    if(IncludeDAQ)
      DataAcquisition();
    if(IncludeMC)
      MonteCarlo();

      CaloTest();
  }

  //write and close the input/output files
  inputFile->Close();
  outputFile->Write();
  outputFile->Close();

  cout<<"program executed; output file= "<<outputFile->GetName()<<endl;
}
```

- There is a macro template you can start from: ReadShoeTreeMain.C and ReadShoeTreeFunc.h
- The main is “simple”:
  -There are different beforeeventloop methods that are called at the beginning.
  -There is an eventloop in which the input file ttree is read and different analysis method are called
  -After the event loop, it closes the files
- You can execute the macro with the following command:
  root -l -b 'ReadShoeTreeMain.C+("recofile.root",0,0)'
  N.B.: there is a “+” after the macro name
- N.B.: to execute this macro, you just need to give an input file without any information about exp name or runnumber.
  This is because the shoe output contain a runinfo object that is stored in the output file.
  Try to type
  runinfo->Print()
  and check what is happening

Analysis macro template

```
static int  IncludeMSD;
static int  IncludeTW;
static int  IncludeCA;
static int  IncludeDAQ;

//global variables
static int debug;              //debug flag
static int  evnum;              //current event number
static int  maxentries;         //max number of events in ttree or set by
static TAGrunInfo*   runinfo;  //runinfo from the input file
static TAGgeoTrafo*  geoTrafo; //useful geometry class that handle all t
static TAGparGeo* parGeo;      //beam and target info
static TAGcampaignManager* campManager; //campaign manager
static  TTree *tree;                     //input tree
static  TFile *inputFile;
static  TFile *outputFile;
TAGroot *gTAGroot;

//detector and MC variables
static TAMCntuHit *scNtuMc;
static TAMCntuHit *bmNtuMc;
static TAMCntuHit *vtNtuMc;
static TAMCntuHit *itNtuMc;
static TAMCntuHit *msdNtuMc;
static TAMCntuHit *twNtuMc;
static TAMCntuHit *caNtuMc;
static TASTntuHit *stNtuHit;
```

```
//methods to fill plots
//N.B.!!!! these methods are "safe", but SLOWER than the usual ((TH1D*)(
//use myfill("plotname",variabletobefilled) or myfill("plotname",firstva
//use myweightfill("plotname",variabletobefilled, weight) to fill h1 and
template <class t>
void myfill(const char *graphname, t x){
  if(gDirectory->Get(graphname)!=nullptr){
    ((TH1D*)(gDirectory->Get(graphname)))->Fill(x);
  }else{
    cout<<"ERROR!!!: "<<graphname<<" cannot be filled because it does no
  }
return;
}

template <class t, class u>
void myfill(const char *graphname, t x, u y){
  if(gDirectory->Get(graphname)!=nullptr){
    ((TH2D*)(gDirectory->Get(graphname)))->Fill(x,y);
  }else{
    cout<<"ERROR!!!: "<<graphname<<" cannot be filled because it does no
  }
return;
}
```

```
//****************************************************************************** be aware of what you change from here on out ******************************
//the following methods are used to open the input file, charge the parameters etc., they should be changed only if really needed

//open the input files
Int_t OpenInputFile(TString nameFile){
  inputFile = new TFile(nameFile.Data());
  if(inputFile->IsOpen()==false){
    cout<<"FATAL ERROR: I cannot open the input file"<<endl;
    return -1;
  }
  tree = (TTree*)inputFile->Get("tree");
  if(tree==nullptr){
    tree = (TTree*)inputFile->Get("EventTree");
    if(tree==nullptr){
      cout<<"FATAL ERROR: your input file do not contains a tree"<<endl;
      return -1;
    }
  }
return 0;
}


  //define and charge varaibles and geometrical parameters from the campmanager
Int_t ChargeCampaignParameters(){

  if(debug)
    cout<<"ChargeCampaignParameters start"<<endl;
```

- In the ReadShoeTreeFunc.h there are different static variables and functions defined.
- Variables and functions are commented
- There are different functions to fill plots, a BookHisto function to define the plots, and different detector analysis functions
  There are few examples about VTX plots using reconstructed and MC quantities
- There are different methods that are used to read the input file, extract exp and run names, load configuration files etc.
  Typically, you can leave these methods and add or modify your own analysis functions
- **You can copy and modify this macro for you own analysis**
- Exercise: understand the examples and modify the macro to add more analysis evaluation
  e.g.: evaluate the VTX track reconstruction efficiency without and with the electric noise,
  e.g. 2: evaluate the residual plot between the BM track and the VTX vertex position

Other analysis methods

New analysis framework

- There is a new analysis framework dedicated to the complex analysis studies (e.g.: Cross section evaluation)
- This has been developed few months ago and it is in a development and test phase (no tutorial will be given today)
- You can find a presentation here: https://agenda.infn.it/event/40055/contributions/233761/attachments/122250/178733/AnaFrame_250624.pdf

Executables

- If necessary, you can develop your own executable to perform specific tasks that are too complicated to be done with a macro
- E.g.: we created a DecodeFast executable that had been used in CNAO2024 data taking to reconstruct the BM and VT detector related quatitites and to evaluate the VT performances as fast as possible.
- There are other executables created to calibrate detectors (e.g.: DecodeCA, CalibBM)
- N.B.: The FOOT reconstruction executable is DecodeGlb.
  Everything done in other executables can have an impact in DecodeGlb (e.g.: be careful changing BaseReco or other shared libraries)
  Not everything done in other executables are directly propagated in DecodeGlb (e.g.: ensure that methods/classes/functions that can improve the reconstruction performances are propagated also in DecodeGlb)

Software development

- The shoe software is in constant evolution! New version improvements are documented here: https://baltig.infn.it/asarti/shoe/-/wikis/Releases

- We use git to maintain the code. The baltig interface from INFN provides a web tool to 'navigate' the code and its changes. Otherwise you can use plain 'git' from command line to keep your code up to date

- How to use git? GIYF (google is your friend)
  git is one (probably the most) used software development platform

- The main branch of SHOE is called "main" and it is the default branch

How to start my work in FOOT?

- **I'm a detector expert (or I'll be a detector expert):** I need to develop some methods for the reconstruction of a specific detector? Fantastic!
  -Create your own branch from the main branch
  -Develop your methods/class in my branch
  -once my work is done (or partially done), check that everything is working both with MC and data, then ask for a merge request (https://baltig.infn.it/asarti/shoe/-/merge_requests —> New merge request)
  -**Update the wiki!** Don't keep secrets to yourself. If the wiki is not updated, people will continue to bother you

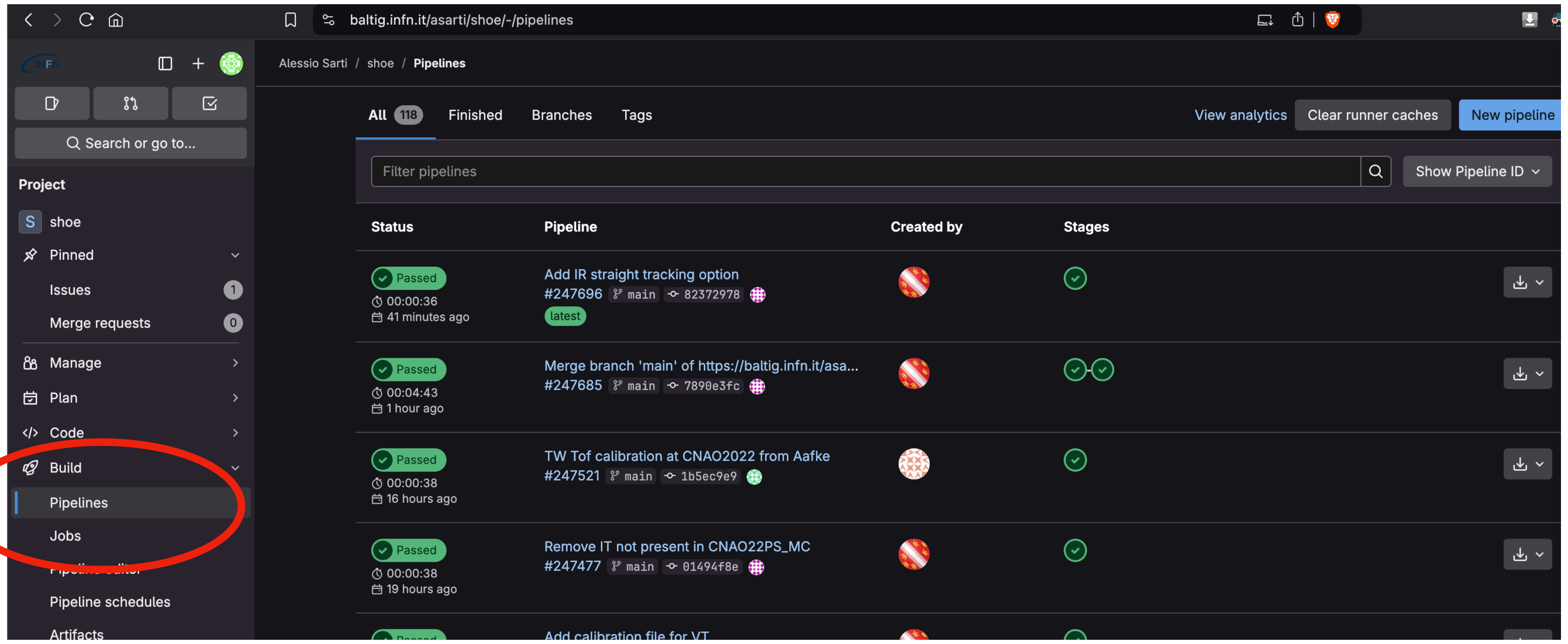
Software development

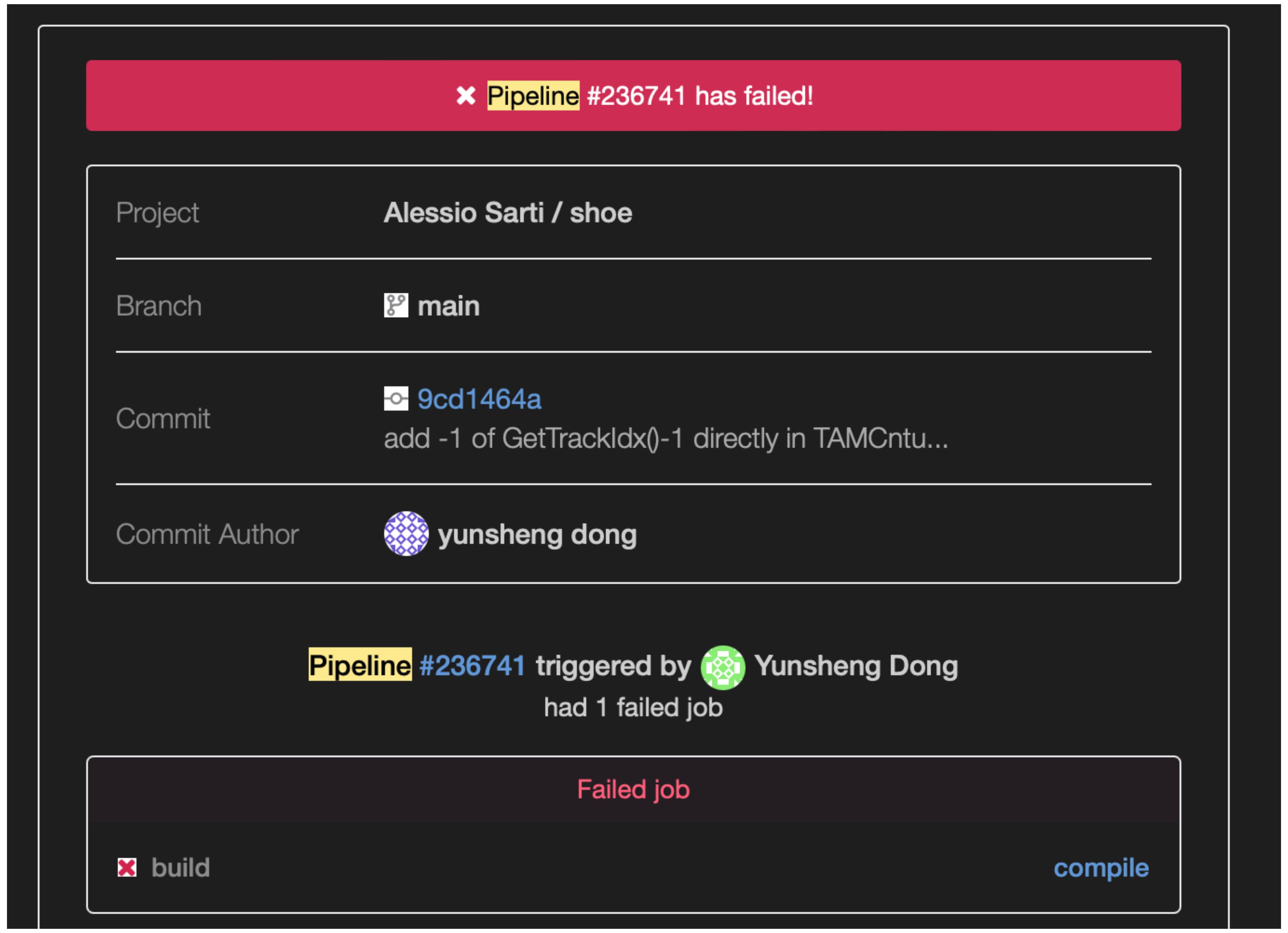
- **I'm an analysis guy:** I want to do some analysis with the FOOT data
  -Contact the Analysis coordinator (Marco Toppi)
  -decide to develop a macro or contribute to the analysis framework
  -In the first case you can just develop your own macro and share your results in the analysis meetings
  -in the second case you will collaborate with the people who already are working on this topic (e.g.: Ubaldi)
  And share your results in the analysis meeting

- In general:
  -**Keep your branch up-to-date with the main one**: If you are a software developer and the distance btw your branch and the master becomes too large, merging the algorithms will become a huge pain in the end.
  If you are an analysis guy, you can miss detector improvements, calibration file updates etc if your local branch is not up-to-date with the main one
  -**Observe the coding conventions as much as possible.** (Check what is done in other classes eg.: TAVTbase). This will ease the merging of your code within shoe!
  -**Remember to add comments in general and add doxygen comments when new variables or methods are implemented.** This can greatly ease the life of future newcomers
  -**Do not reinvent the wheel**! we have tons of lines of code and examples. Just ask!

- Contact Marco or Alessio and ask to be inserted in the following foot mailing lists (if you want):
  foot-analysis-and-reco
  foot-software-develop

Git pipeline





What is a CICD pipeline?

A pipeline is the lead component of continuous integration, delivery, and deployment. It drives software development through building, testing and deploying code in stages. Pipelines are comprised of jobs, which define what will be done, such as compiling or testing code, as well as stages that spell out when to run the jobs. An example would be running tests after stages that compile the code.

A CI/CD pipeline automates steps in the SDLC such as builds, tests, and deployments. When a team takes advantage of automated pipelines, they simplify the handoff process and decrease the chance of human error, creating faster iterations and better quality code. Everyone can see where code is in the process and identify problems long before they make it to production.

- There is a pipeline that, **at each push of any shoe branch**, build the whole code from scratch on a linux based machine.

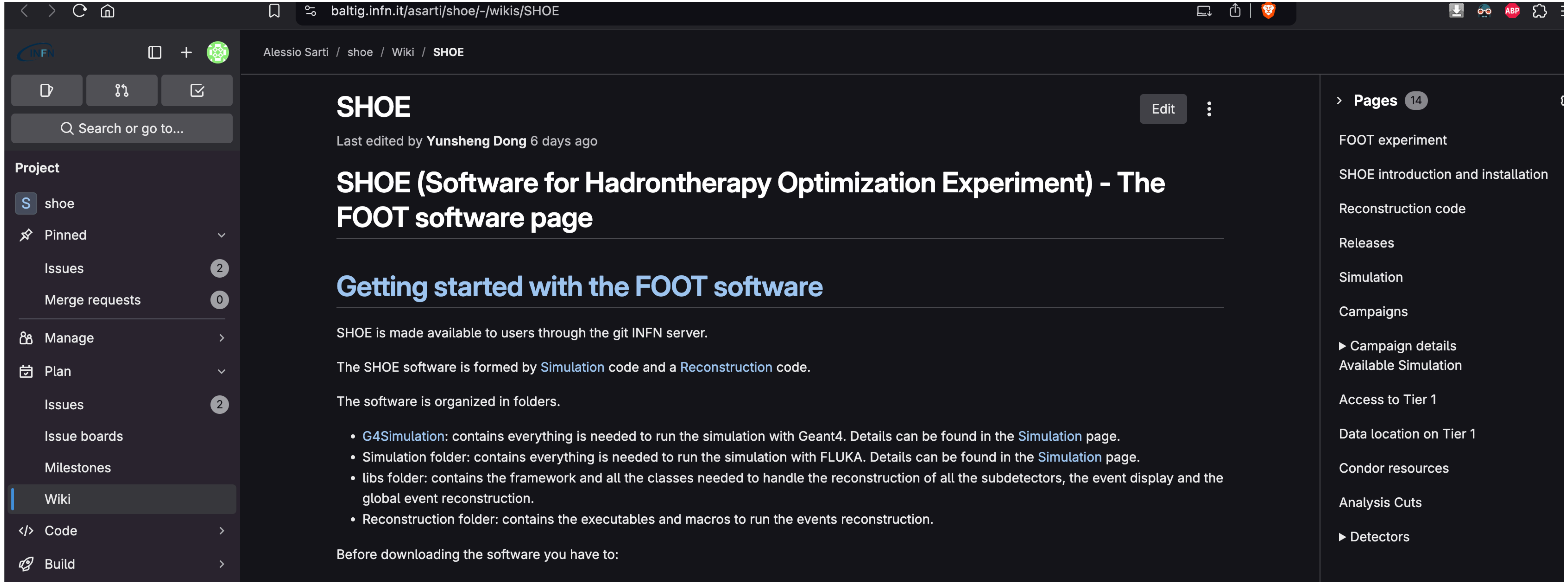
- If the pipeline fails, an email message will be sent.
  You can click on the Pipeline# link and check the error
  (N.B.: sometime there can be false errors due to out-of-memory issues on baltig)

- **Pipeline is a useful tool to check the compilation of the code. However, it is good practice to check and test the code before pushing**

- There is also a “deploy” stage of the pipeline that update the doxygen libraries at each push on the main branch

Wiki

- There is a wiki page of SHOE on the git repository: https://baltig.infn.it/asarti/shoe/-/wikis/SHOE
- The wiki is used both to document the software status and to provide information about data takings, analysis etc.
- The wiki is accessible only by the people in the collaboration and it can be modified by whoever can access to shoe
- **Since both the software and the analysis status are in evolution, we need to update the wiki constantly**



Wiki

- Each wiki page can be edited directly online (just press the edit button on the top right corner).
  There is also a preview function.
  Remember to save after the changes

How to add images/presentations or modify the page with your own text editor locally:

- the shoe wiki is a repository that can be downloaded and used as a git repository, it's name is shoe.wiki.git instead of shoe.git

- git clone https://baltig.infn.it/asarti/shoe.wiki.git
  Will download the whole wiki page, you will find different *.md files that are the wiki pages and there is an "uploads" folder in which you can add all the images, presentations etc. that you want to add to the wiki page.
  e.g.: I added a 3dnewcut.png file in uploads/AvailableSimulation folder

- Add the reference to the image in the page
  e.g.: add the following line in the available-simulation wiki page
  <p></p>

Doxygen

- Doxygen documentation of SHOE is available here: https://asarti.baltig-pages.infn.it/shoe/annotated.html
- We (mainly Chris, Rob and few other people) spent a lot of time to add the doxygen comments on almost all the SHOE code
- **Please: add doxygen comments if you modify or develop new classes in SHOE**
  It's an easy task and it take just few seconds to add a comment. The revision of the code to add comments is a much more time consuming (and boring) work
- You can find instructions here: https://agenda.infn.it/event/29336/contributions/149012/attachments/87696/117259/Doxygen_ChF.pdf
- You can find different examples in the other SHOE classes

Communication channels

- **Usually, we use this email list to communicate and spread information about software: foot-software-develop@lists.infn.it**

- Sometime (very rarely) we have software meetings

Rocket chat

- If you can access to the INFN resources, you'll have also access to the INFN rocket.chat interface: https://chat.infn.it/

- This is a team chat based on HTML5 (real time) and it can be used with a desktop app and/or directly on the web page

- We have different channels dedicated to FOOT

- The channel dedicated to SHOE is called "software" and you can join the channel with the following link:
https://go.rocket.chat/invite?host=chat.infn.it&path=invite%2FFYJkBE

Issues and troubleshoot

- If you have issues, first of all try to resolve yourself
  N.B.: sometime the problem is that people are used to copy from a pdf and paste on the terminal…
  No, try to type the full command by yourself!

- GIYF (Google is your friend)

- Search on the wiki and check the known issues page: https://baltig.infn.it/asarti/shoe/-/issues

- If the issue is something relevant/“long to be fixed”
  add/update the wiki issue page

- If the issue is something simple/easily solvable and if you need a fast response (e.g.: my code doesn’t compile, the ./DecodeGlb crashes unexpectedly):
  Send an email to foot-software-develop@lists.infn.it or send a message to the rocket chat channel