ECCE Simulation Workflow

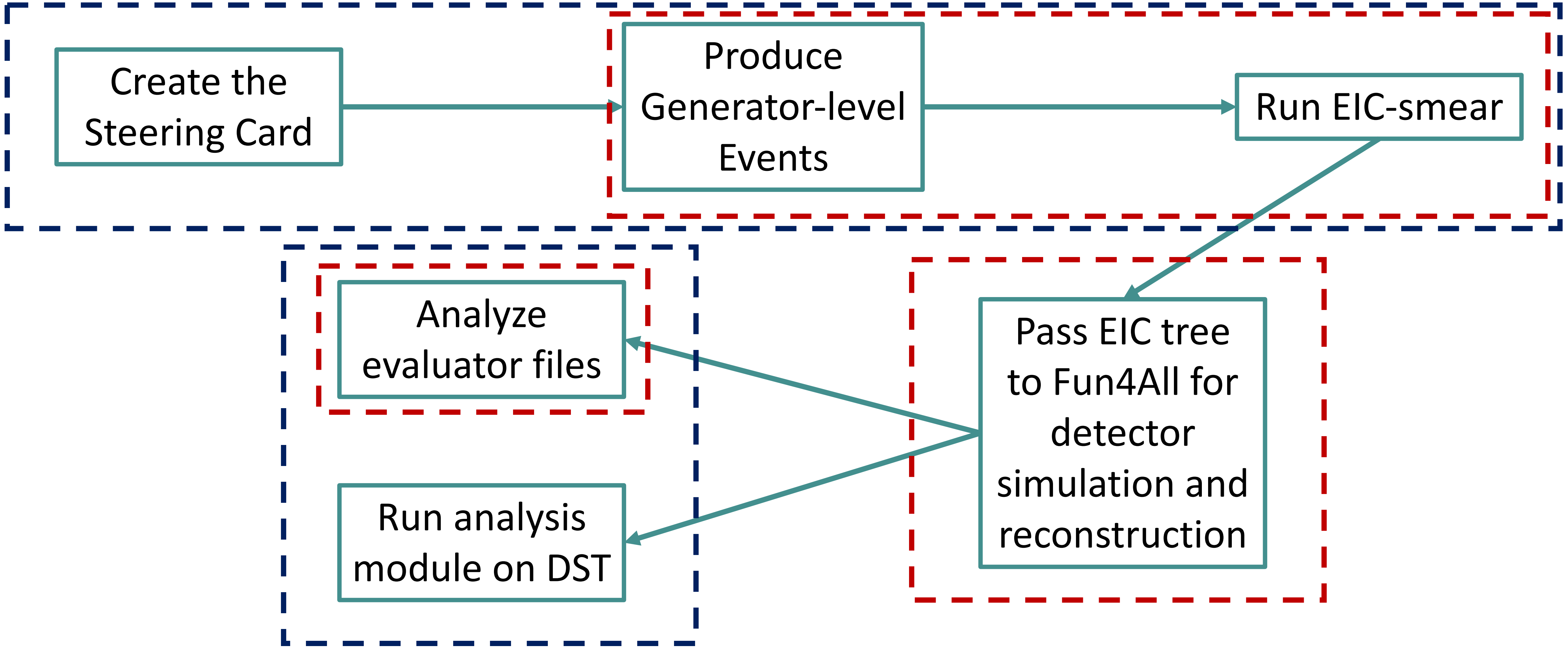
From MC input samples to output DSTs

Cameron Dean

05/21/2021

2nd ECCE Simulations Workshop

Overview


Create the Steering Card
Produce Generator-level Events
Run EIC-smear
Pass EIC tree to Fun4All for detector simulation and reconstruction
Analyze evaluator files
Run analysis module on DST

🟦 In the realm of PWGs

🟥 In the realm of Simulations

Part 1

Generation

Overview

- Not every PWG will use the same generators or steering cards
- This is typically fast, low resource and small storage (relative)
- Can be handled directly by PWGs
- List of generators and instructions: https://eic.github.io/software/mcgen.html
- This example will use pythia6, available at https://gitlab.com/eic/mceg/PYTHIA-RAD-CORR
- I will use the input file:

`PYTHIA-RAD-CORR/STEER-FILES-Official/official.pythia6.18x275.noRC.eic`

- Please note that our singularity container comes with “pythiaeRHIC” installed already:

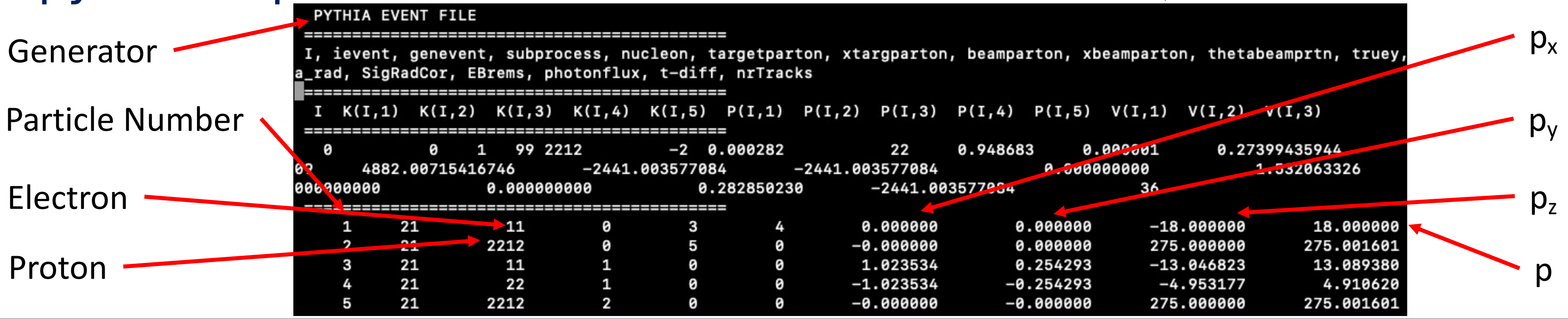
```
 Singularity> which pythiaeRHIC
/cvmfs/eic.opensciencegrid.org/ecce/gcc-8.3/release/release_new/new.2/bin/pythiaeRHIC
```

Running pythia6

- This pythia6 distro. is built using cmake and then running “`make install`”
- If you use our pythia6 build, change directory to:
  `cd PYTHIA-RAD-CORR/STEER-FILES-Official`
- Now run this command on one line

```
./pythiaeRHIC < official.pythia6.18x275.noRC.eic > log.txt
```

- You will now have an output ASCII file with all your generated events:
  `pythia.ep.18x275.5kevents.1.RadCor\=0.Q2\=1.0-20000.txt`



```
 PYTHIA EVENT FILE
 ============================================
 I, ievent, genevent, subprocess, nucleon, targetparton, xtargparton, beamparton, xbeamparton, thetabeamprtn, truey,
a_rad, SigRadCor, EBrems, photonflux, t-diff, nrTracks
 ============================================
  I  K(I,1)  K(I,2)  K(I,3)  K(I,4)  K(I,5)  P(I,1)  P(I,2)  P(I,3)  P(I,4)  P(I,5)  V(I,1)  V(I,2)  V(I,3)
 ============================================
   0          0    1   99 2212          -2  0.000282            22      0.948683      0.000001       0.27399435944
02      4882.00715416746        -2441.003577084         -2441.003577084                0.000000000             1.532063326
000000000             0.000000000             0.282850230        -2441.003577084                36
 ============================================
     1     21        11         0         3         4     0.000000      0.000000    -18.000000     18.000000
     2     21      2212         0         5         0    -0.000000      0.000000    275.000000    275.001601
     3     21        11         1         0         0     1.023534      0.254293    -13.046823     13.089380
     4     21        22         1         0         0    -1.023534     -0.254293     -4.953177      4.910620
     5     21      2212         2         0         0    -0.000000     -0.000000    275.000000    275.001601
```

Running EIC-smear

- We can turn the text file into a TTree, readable by Fun4All with the following commands:

```
root -l
gSystem->Load("libeicsmear.so");
BuildTree("pythia.ep.18x275.5kevents.1.RadCor\=0.Q2\=1.0-20000.txt", ".", -1, "log.txt")
```

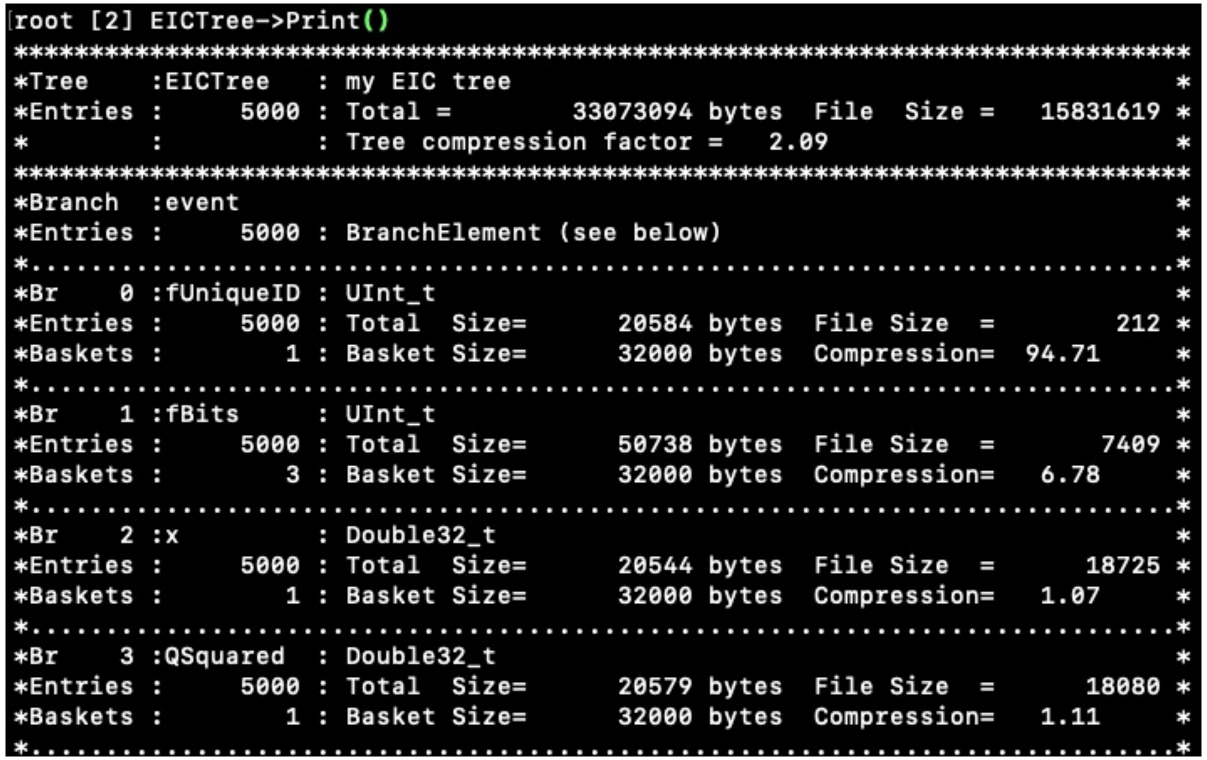
- The BuildTree options are:

```
BuildTree(“<inputFile>”, “<outputPath>”, <nEvents>, “<logFile>”)
```

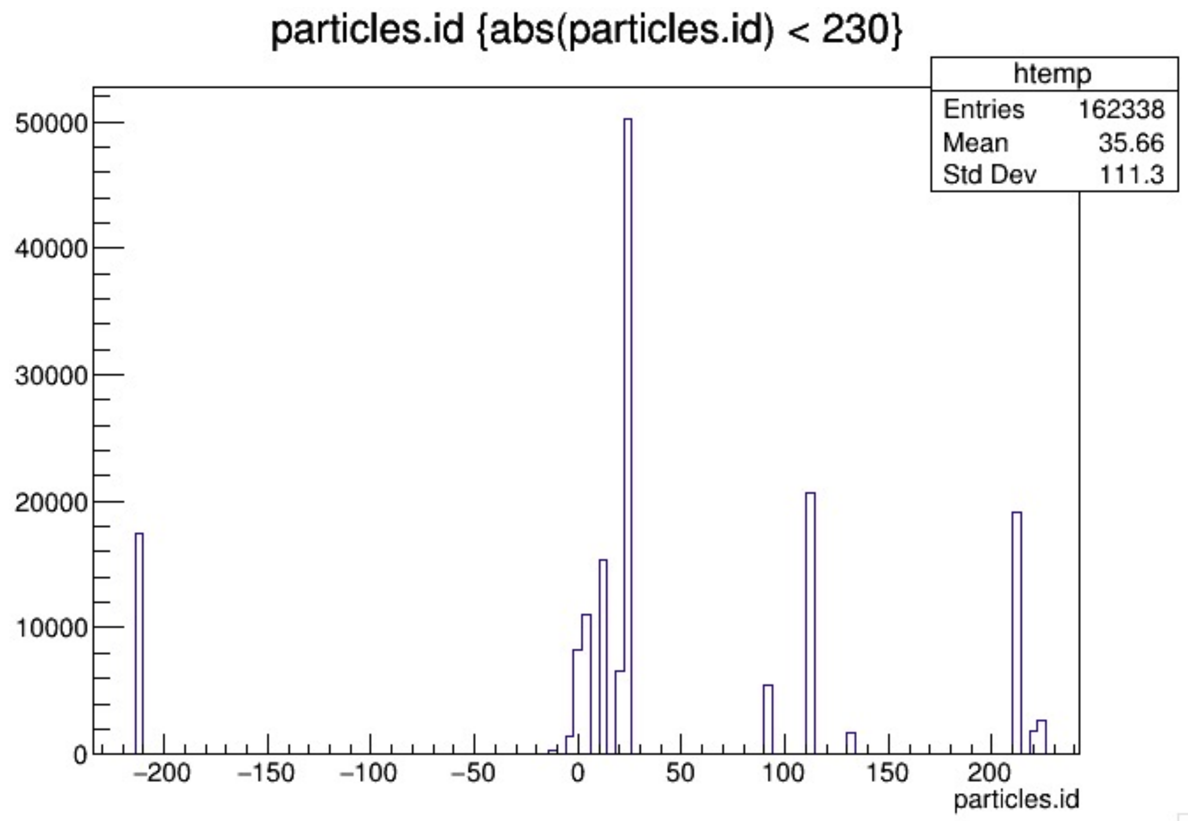
- You should now have an output tree that we can run the simulation on

```
Processed pythia.ep.18x275.5kevents.1.RadCor=0.Q2=1.0-20000.txt
TFile** ./pythia.ep.18x275.5kevents.1.RadCor=0.Q2=1.0-20000.root
 TFile* ./pythia.ep.18x275.5kevents.1.RadCor=0.Q2=1.0-20000.root
  OBJ: TTree EICTree my EIC tree : 0 at: 0x626aed0
  KEY: TProcessID ProcessID0;1 81566468-b9b4-11eb-a796-6330c782beef
  KEY: TTree EICTree;1 my EIC tree
Began on Thu May 20 17:44:16 2021
Ended on Thu May 20 17:44:19 2021
Processed 5000 events containing 203248 particles in 2.87108 seconds (0.000574216 sec/event)
```

EIC-smear output

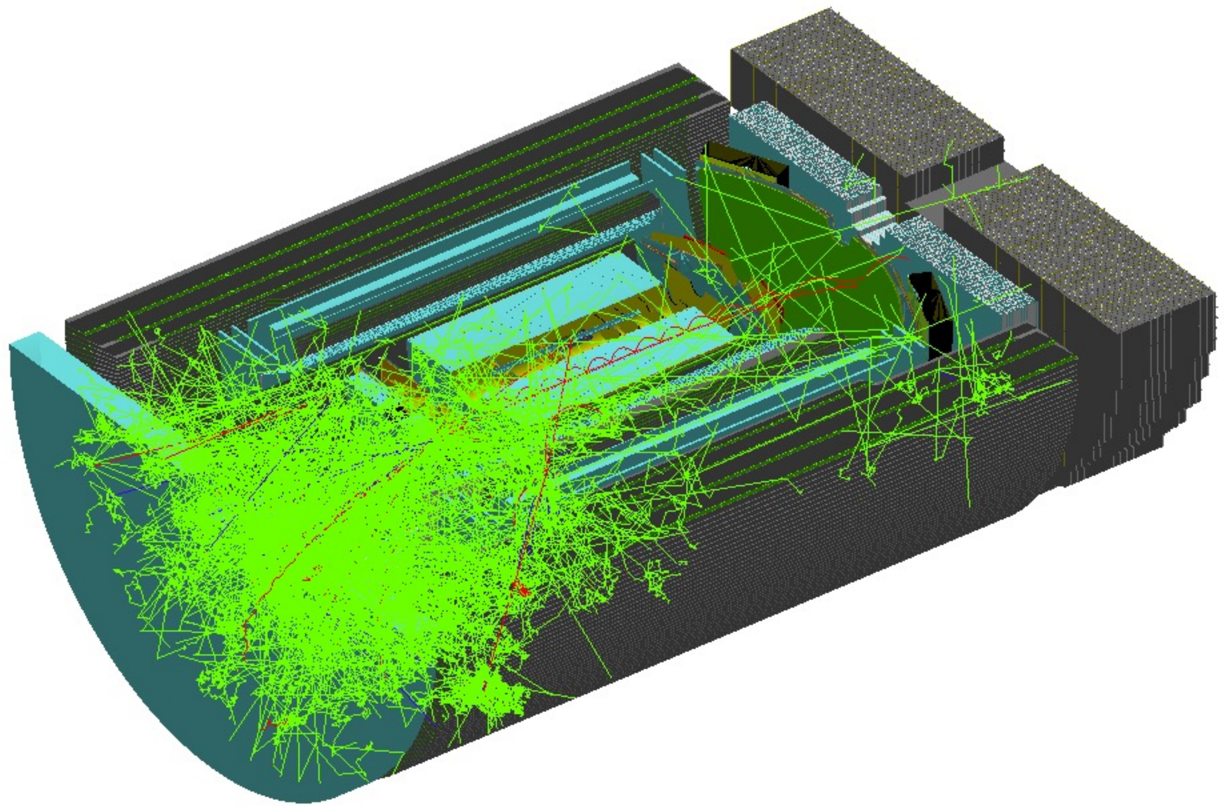


```
root [2] EICTree->Print()
******************************************************************************
*Tree    :EICTree   : my EIC tree                                            *
*Entries :     5000 : Total =        33073094 bytes  File  Size =   15831619 *
*        :          : Tree compression factor =   2.09                       *
******************************************************************************
*Branch  :event                                                              *
*Entries :     5000 : BranchElement (see below)                              *
*............................................................................*
*Br    0 :fUniqueID : UInt_t                                                 *
*Entries :     5000 : Total  Size=      20584 bytes  File Size  =        212 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  94.71     *
*............................................................................*
*Br    1 :fBits     : UInt_t                                                 *
*Entries :     5000 : Total  Size=      50738 bytes  File Size  =       7409 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   6.78     *
*............................................................................*
*Br    2 :x         : Double32_t                                             *
*Entries :     5000 : Total  Size=      20544 bytes  File Size  =      18725 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.07     *
*............................................................................*
*Br    3 :QSquared  : Double32_t                                             *
*Entries :     5000 : Total  Size=      20579 bytes  File Size  =      18080 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.11     *
*............................................................................*
```



Part 2

Simulation



Overview

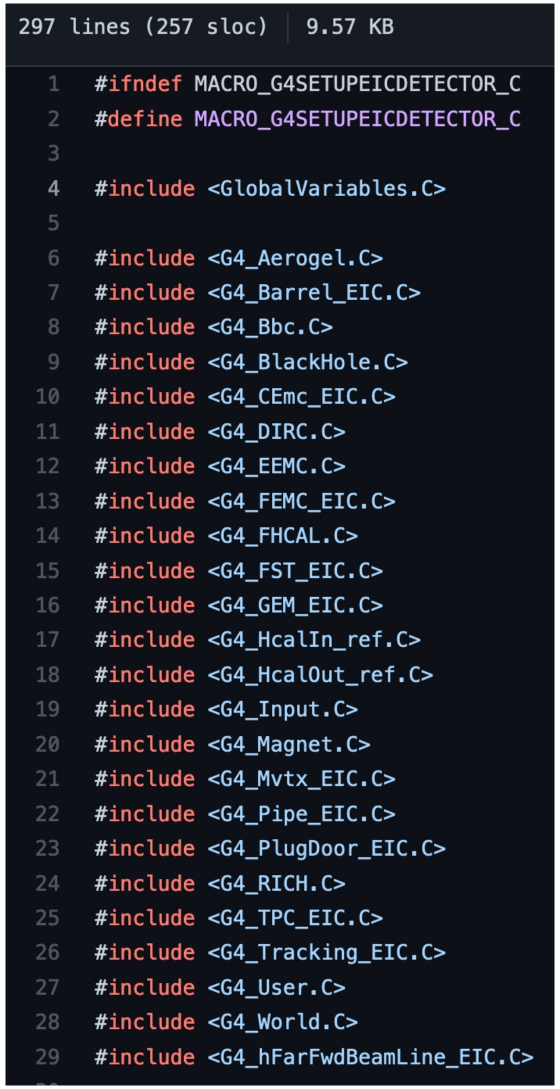
- Our detector simulation is located in `macros/detectors/EICDetector/Fun4All_G4_EICDetector.C`
- This macro can read in an EIC TTree, simulate detector responses, do the event reconstruction, write the event information to files (DST or TTree) and visualize the detector/event
- This is a top-level macro that runs smaller macros in

`macros/common` and `G4Setup_EICDetector.C`

- Default action is pion injection by setting `Input::Simple = true`
- Display is **disabled** by default
- DST writing is **disabled** by default
- Evaluators are **enabled** by default

Running a simulation

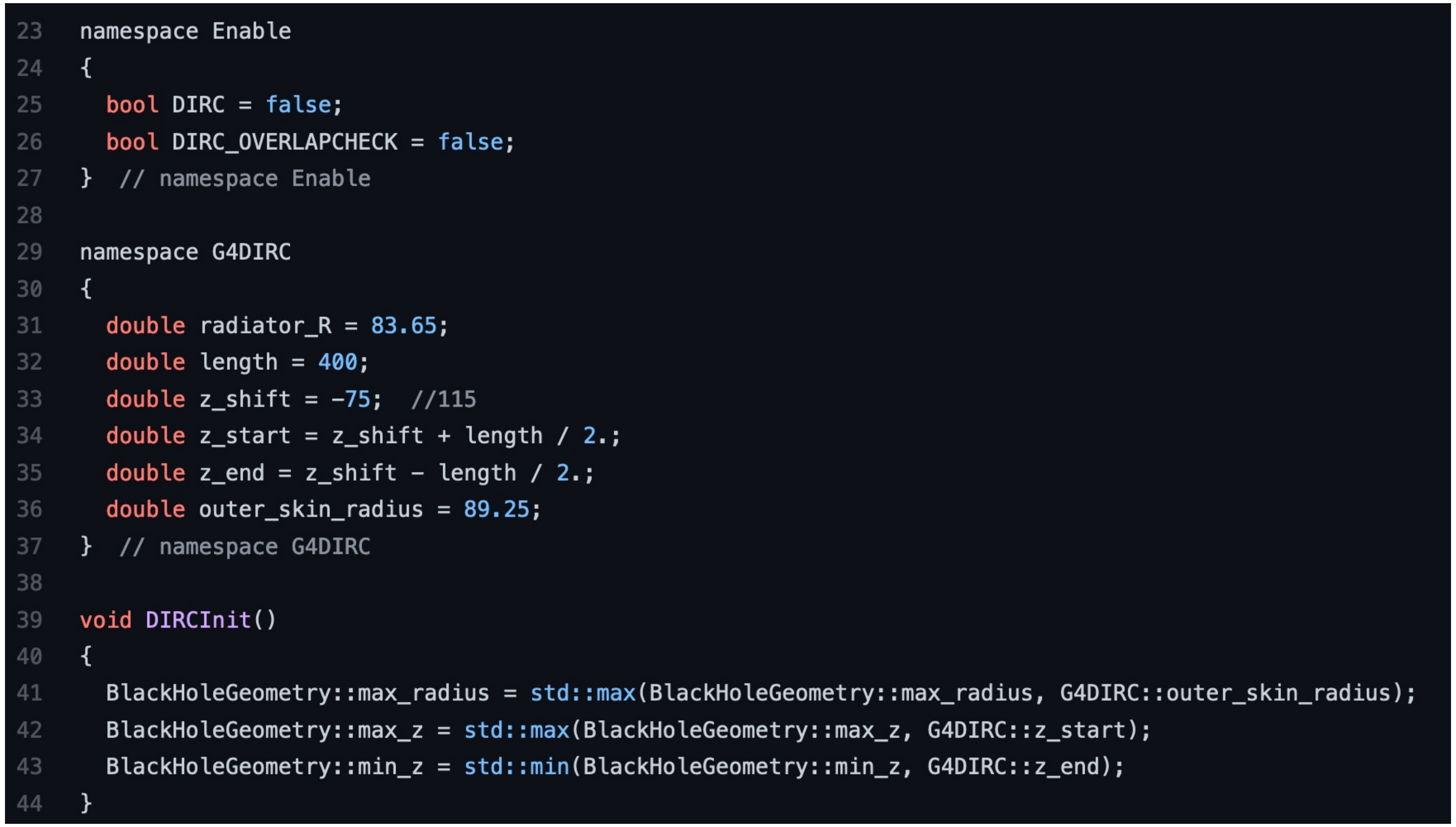
- The top-level macro interfaces with lower-level macros
- The lower-level macros typically interface with compiled classes
- This means you can write and run our macros without needing to compile classes
  (see Joe's talk on how to do this)
- Typically we read in an EIC TTree, simulate the detector response, perform reconstructions and then run any further analyses as required



```
297 lines (257 sloc)  9.57 KB

#ifndef MACRO_G4SETUPEICDETECTOR_C
#define MACRO_G4SETUPEICDETECTOR_C

#include <GlobalVariables.C>

#include <G4_Aerogel.C>
#include <G4_Barrel_EIC.C>
#include <G4_Bbc.C>
#include <G4_BlackHole.C>
#include <G4_CEmc_EIC.C>
#include <G4_DIRC.C>
#include <G4_EEMC.C>
#include <G4_FEMC_EIC.C>
#include <G4_FHCAL.C>
#include <G4_FST_EIC.C>
#include <G4_GEM_EIC.C>
#include <G4_HcalIn_ref.C>
#include <G4_HcalOut_ref.C>
#include <G4_Input.C>
#include <G4_Magnet.C>
#include <G4_Mvtx_EIC.C>
#include <G4_Pipe_EIC.C>
#include <G4_PlugDoor_EIC.C>
#include <G4_RICH.C>
#include <G4_TPC_EIC.C>
#include <G4_Tracking_EIC.C>
#include <G4_User.C>
#include <G4_World.C>
#include <G4_hFarFwdBeamLine_EIC.C>
```

Taking a closer look at a detector

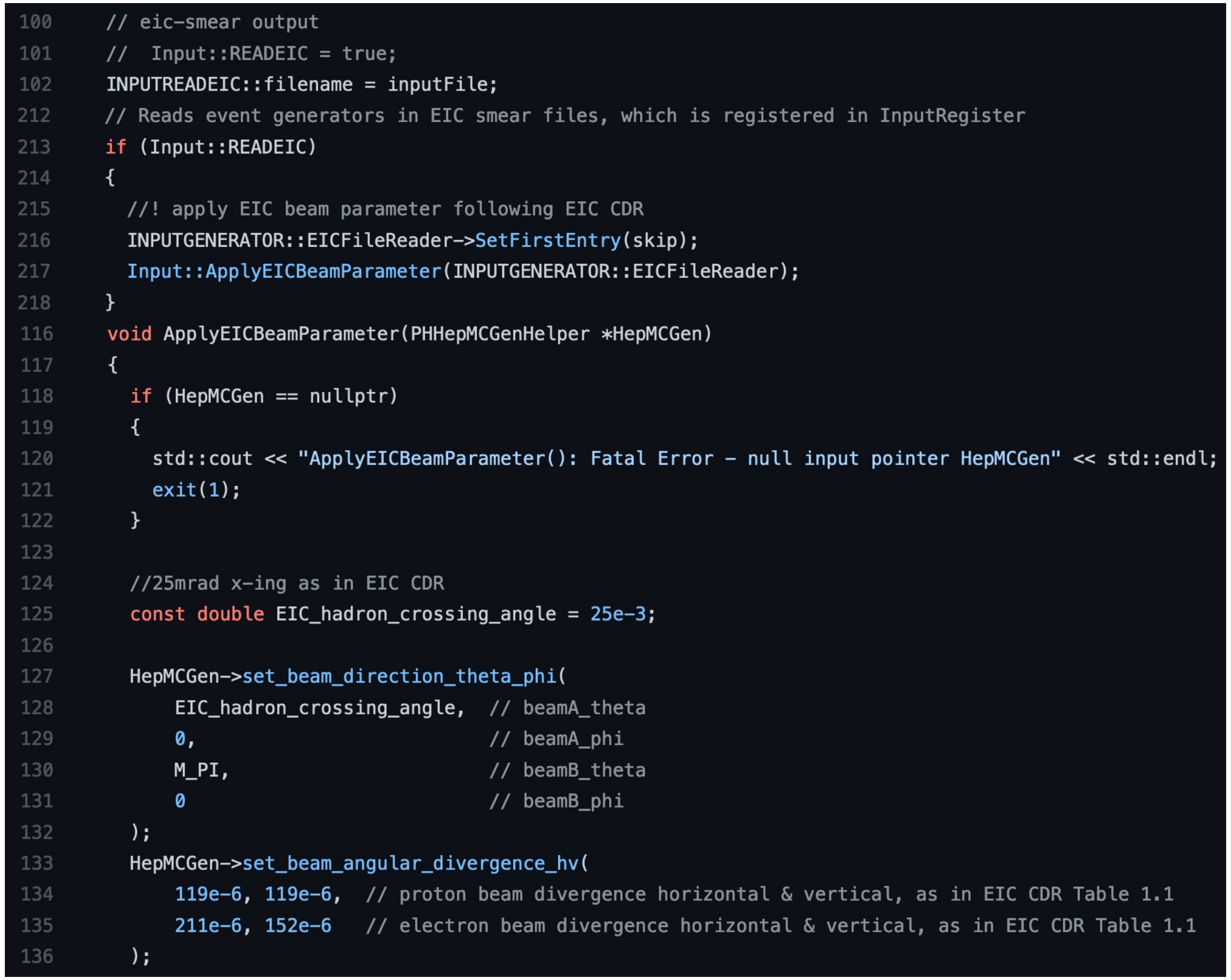
- This is macros/common/G4_DIRC.C
- Use the variables in namespaces to alter parameters in the top-level



```
namespace Enable
{
  bool DIRC = false;
  bool DIRC_OVERLAPCHECK = false;
}  // namespace Enable

namespace G4DIRC
{
  double radiator_R = 83.65;
  double length = 400;
  double z_shift = -75;  //115
  double z_start = z_shift + length / 2.;
  double z_end = z_shift - length / 2.;
  double outer_skin_radius = 89.25;
}  // namespace G4DIRC

void DIRCInit()
{
  BlackHoleGeometry::max_radius = std::max(BlackHoleGeometry::max_radius, G4DIRC::outer_skin_radius);
  BlackHoleGeometry::max_z = std::max(BlackHoleGeometry::max_z, G4DIRC::z_start);
  BlackHoleGeometry::min_z = std::min(BlackHoleGeometry::min_z, G4DIRC::z_end);
}
```

How do we read and setup an event

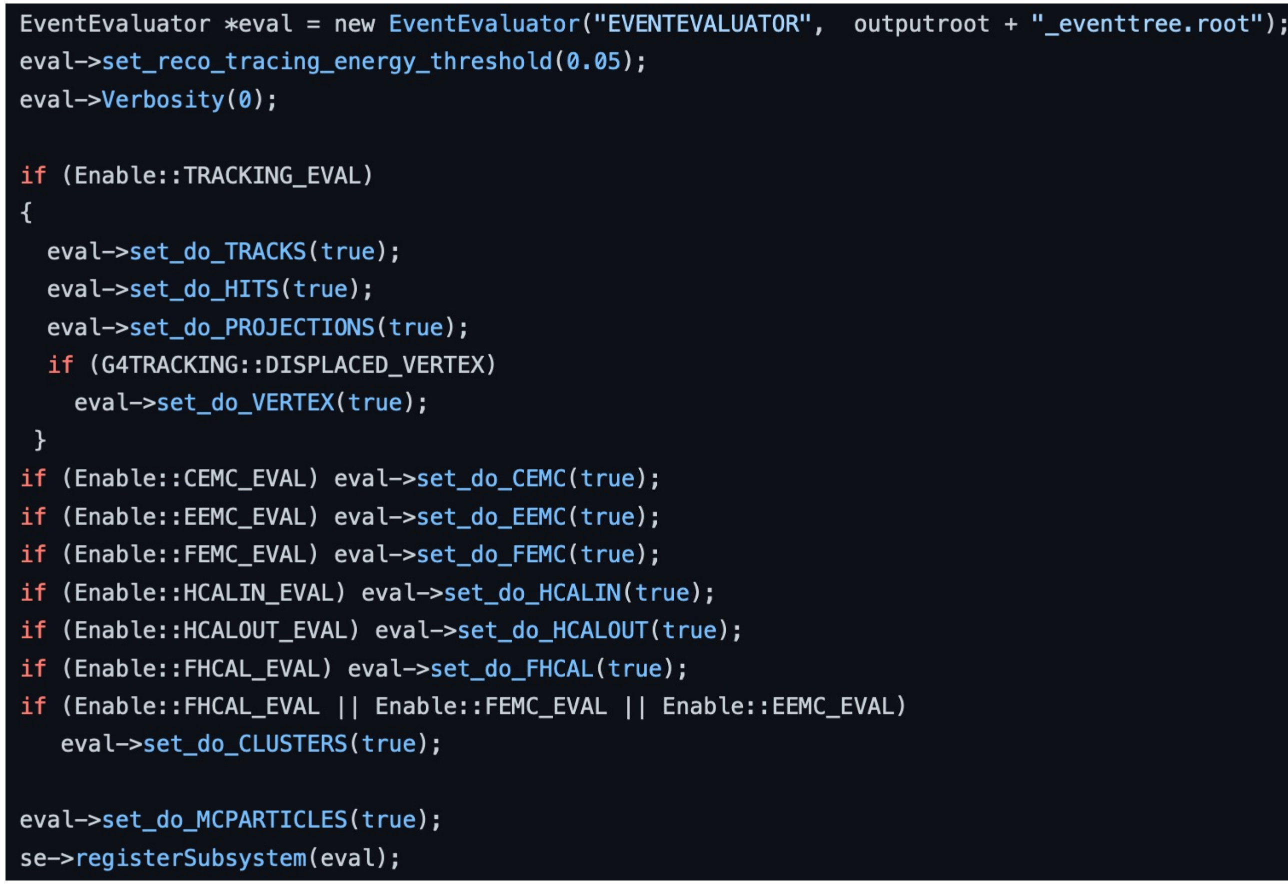
1. We enable the EIC reader
2. We set the input file, number of events to process, where to start and where to write to
3. We set the beam parameters
4. We then process all required events with what detectors and reconstruction we want



```
    // eic-smear output
    //  Input::READEIC = true;
    INPUTREADEIC::filename = inputFile;
    // Reads event generators in EIC smear files, which is registered in InputRegister
    if (Input::READEIC)
    {
      //! apply EIC beam parameter following EIC CDR
      INPUTGENERATOR::EICFileReader->SetFirstEntry(skip);
      Input::ApplyEICBeamParameter(INPUTGENERATOR::EICFileReader);
    }
    void ApplyEICBeamParameter(PHHepMCGenHelper *HepMCGen)
    {
      if (HepMCGen == nullptr)
      {
        std::cout << "ApplyEICBeamParameter(): Fatal Error - null input pointer HepMCGen" << std::endl;
        exit(1);
      }

      //25mrad x-ing as in EIC CDR
      const double EIC_hadron_crossing_angle = 25e-3;

      HepMCGen->set_beam_direction_theta_phi(
          EIC_hadron_crossing_angle,  // beamA_theta
          0,                          // beamA_phi
          M_PI,                       // beamB_theta
          0                           // beamB_phi
      );
      HepMCGen->set_beam_angular_divergence_hv(
          119e-6, 119e-6,  // proton beam divergence horizontal & vertical, as in EIC CDR Table 1.1
          211e-6, 152e-6   // electron beam divergence horizontal & vertical, as in EIC CDR Table 1.1
      );
```

What's next

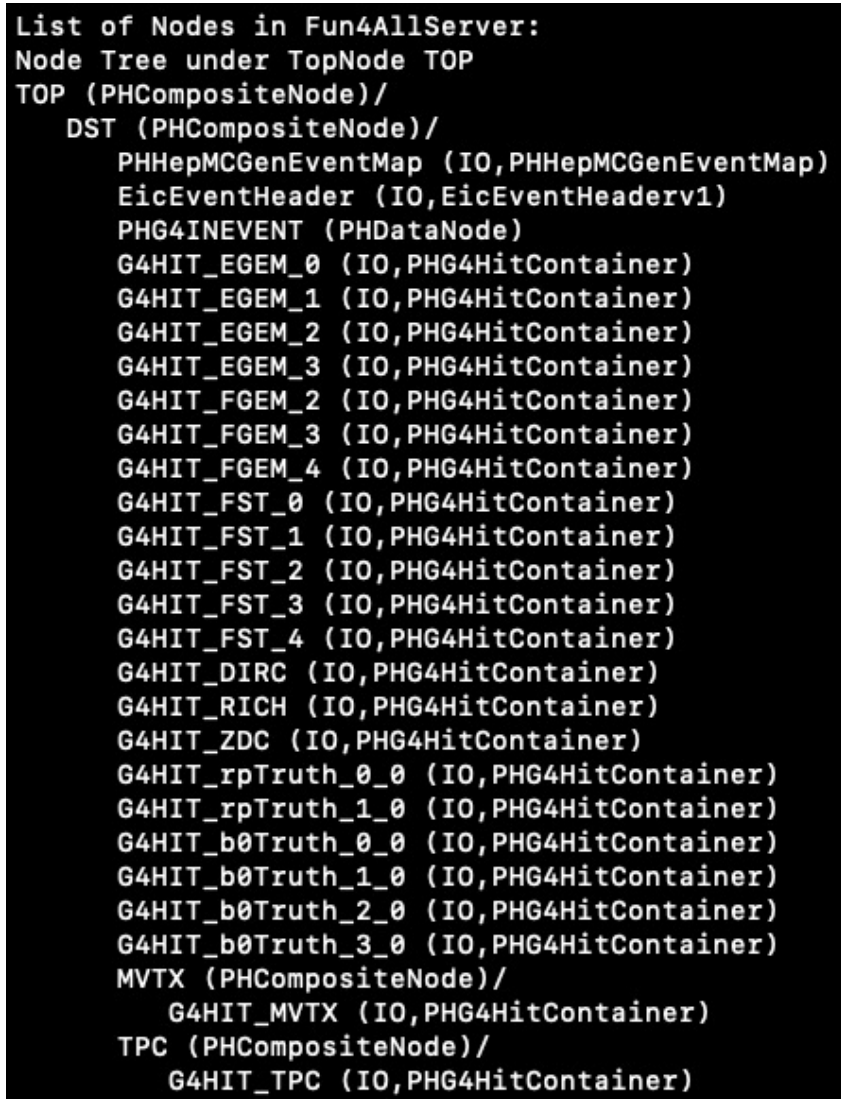
- For each event, the particles response to their relevant detector is quantified
- Reconstruction algorithms such as tracking are run
- All this information (generator-level particles, hits, reconstructed tracks) are written to DSTs to be used by analysts
- Evaluators are also run for various processes and written to nTuples



```
EventEvaluator *eval = new EventEvaluator("EVENTEVALUATOR",  outputroot + "_eventtree.root");
eval->set_reco_tracing_energy_threshold(0.05);
eval->Verbosity(0);

if (Enable::TRACKING_EVAL)
{
  eval->set_do_TRACKS(true);
  eval->set_do_HITS(true);
  eval->set_do_PROJECTIONS(true);
  if (G4TRACKING::DISPLACED_VERTEX)
    eval->set_do_VERTEX(true);
 }
if (Enable::CEMC_EVAL) eval->set_do_CEMC(true);
if (Enable::EEMC_EVAL) eval->set_do_EEMC(true);
if (Enable::FEMC_EVAL) eval->set_do_FEMC(true);
if (Enable::HCALIN_EVAL) eval->set_do_HCALIN(true);
if (Enable::HCALOUT_EVAL) eval->set_do_HCALOUT(true);
if (Enable::FHCAL_EVAL) eval->set_do_FHCAL(true);
if (Enable::FHCAL_EVAL || Enable::FEMC_EVAL || Enable::EEMC_EVAL)
   eval->set_do_CLUSTERS(true);

eval->set_do_MCPARTICLES(true);
se->registerSubsystem(eval);
```

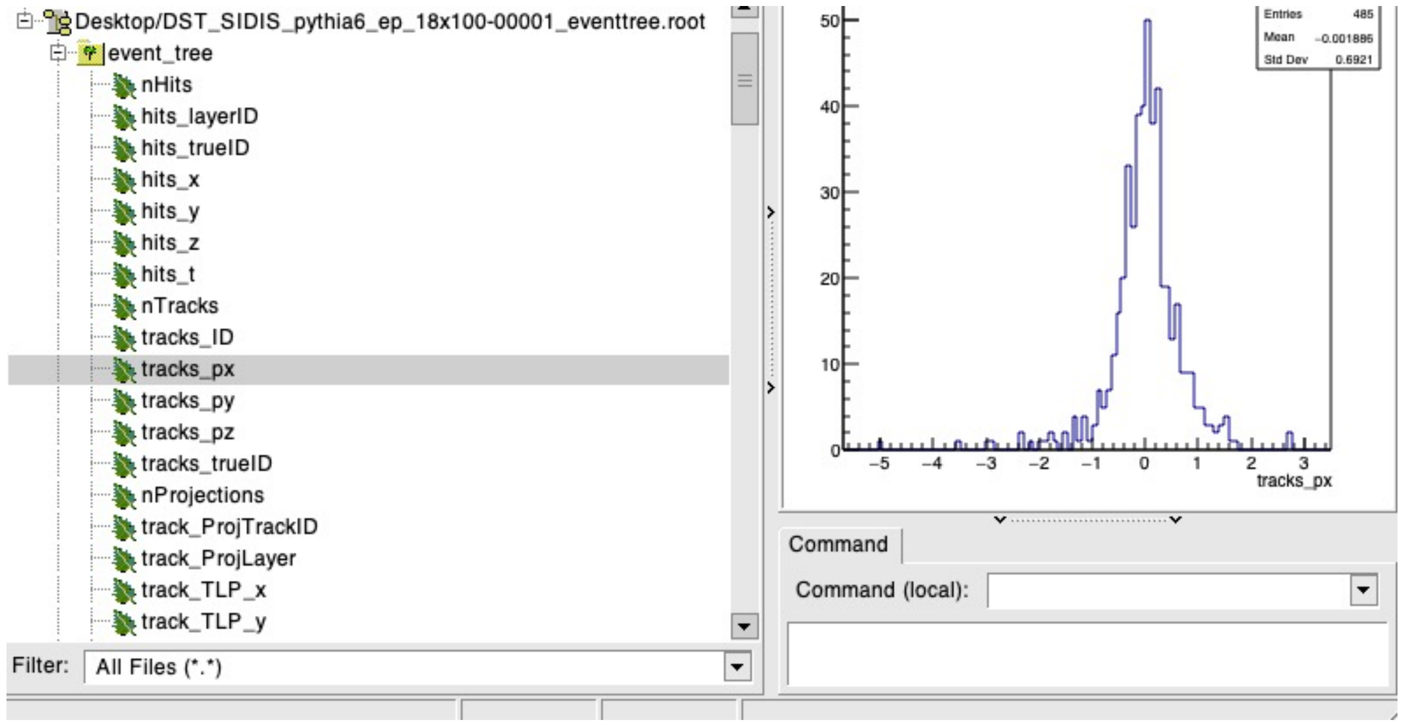
What’s next

- For each event, the particles response to their relevant detector is quantified
- Reconstruction algorithms such as tracking are run
- All this information (generator-level particles, hits, reconstructed tracks) are written to DSTs to be used by analysts
- Evaluators are also run for various processes and written to nTuples



```
List of Nodes in Fun4AllServer:
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
   DST (PHCompositeNode)/
      PHHepMCGenEventMap (IO,PHHepMCGenEventMap)
      EicEventHeader (IO,EicEventHeaderv1)
      PHG4INEVENT (PHDataNode)
      G4HIT_EGEM_0 (IO,PHG4HitContainer)
      G4HIT_EGEM_1 (IO,PHG4HitContainer)
      G4HIT_EGEM_2 (IO,PHG4HitContainer)
      G4HIT_EGEM_3 (IO,PHG4HitContainer)
      G4HIT_FGEM_2 (IO,PHG4HitContainer)
      G4HIT_FGEM_3 (IO,PHG4HitContainer)
      G4HIT_FGEM_4 (IO,PHG4HitContainer)
      G4HIT_FST_0 (IO,PHG4HitContainer)
      G4HIT_FST_1 (IO,PHG4HitContainer)
      G4HIT_FST_2 (IO,PHG4HitContainer)
      G4HIT_FST_3 (IO,PHG4HitContainer)
      G4HIT_FST_4 (IO,PHG4HitContainer)
      G4HIT_DIRC (IO,PHG4HitContainer)
      G4HIT_RICH (IO,PHG4HitContainer)
      G4HIT_ZDC (IO,PHG4HitContainer)
      G4HIT_rpTruth_0_0 (IO,PHG4HitContainer)
      G4HIT_rpTruth_1_0 (IO,PHG4HitContainer)
      G4HIT_b0Truth_0_0 (IO,PHG4HitContainer)
      G4HIT_b0Truth_1_0 (IO,PHG4HitContainer)
      G4HIT_b0Truth_2_0 (IO,PHG4HitContainer)
      G4HIT_b0Truth_3_0 (IO,PHG4HitContainer)
      MVTX (PHCompositeNode)/
         G4HIT_MVTX (IO,PHG4HitContainer)
      TPC (PHCompositeNode)/
         G4HIT_TPC (IO,PHG4HitContainer)
```

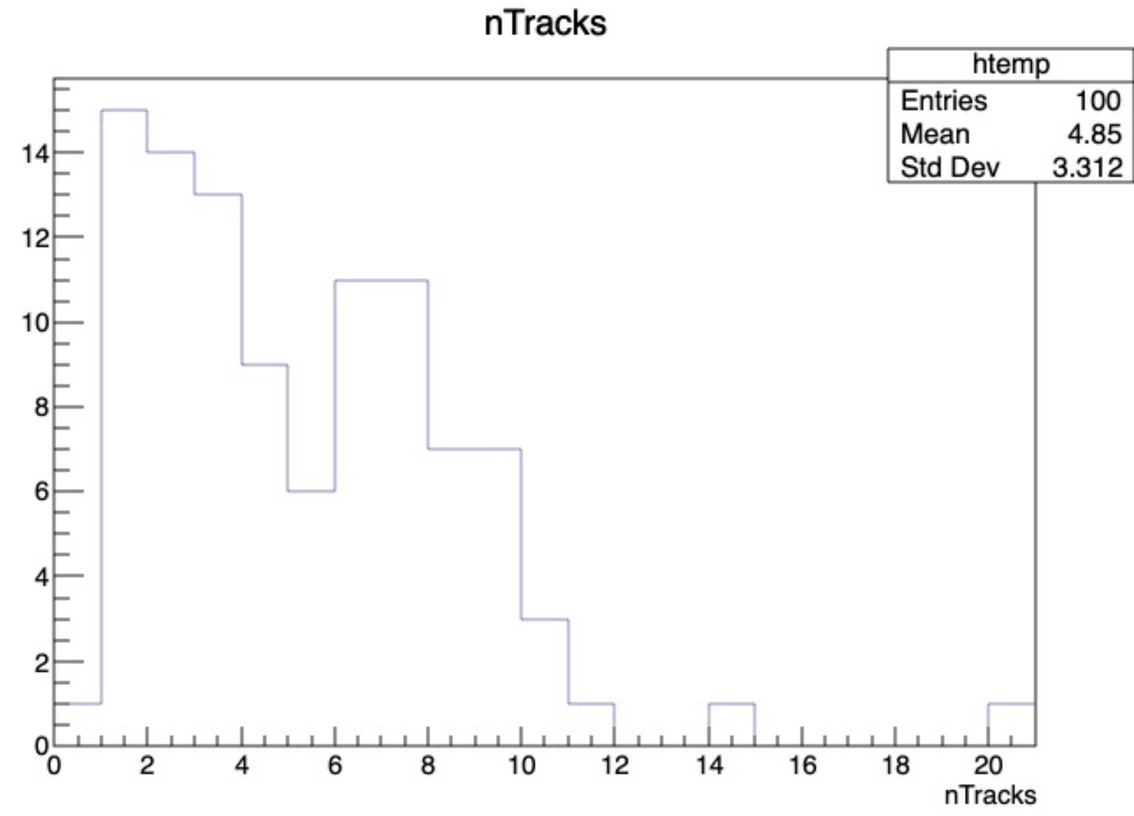
Evaluators preview



Desktop/DST_SIDIS_pythia6_ep_18x100-00001_eventtree.root
- event_tree
  - nHits
  - hits_layerID
  - hits_trueID
  - hits_x
  - hits_y
  - hits_z
  - hits_t
  - nTracks
  - tracks_ID
  - tracks_px
  - tracks_py
  - tracks_pz
  - tracks_trueID
  - nProjections
  - track_ProjTrackID
  - track_ProjLayer
  - track_TLP_x
  - track_TLP_y

Filter: All Files (*.*)

Entries	485
Mean	-0.001886
Std Dev	0.6921



htemp	
Entries	100
Mean	4.85
Std Dev	3.312

- Evaluators give easy access to some information such as track momentum or calorimeter deposits

Part 3

Accessing the Output

Storage locations and tags

- We will require storage in the PB range
  - This is not feasible or practical for everyone or institute to cope
- Output files will be stored at BNL and JLab
  - We will use the S3 protocol to allow off-site access
  - Use this to access the data you need
  - Read-access accounts will be distributed to those that require it
    - Write-access will be limited to a handful of people
- In the end, it will appear that you have the data locally, no need to request SDCC computing access
- We want a record of how we produced the simulations, for each file:
  1. ECCE coresoftware build tag (for reconstruction and evaluators)
  2. Macros version tag (for detector configuration and other parameters)
  3. Input tree and initial event number (To know what data to read and where to start)
  4. Simulation seeds (to exactly reproduce the simulation)

Reading in DSTs

- If you decide to use DSTs over ready-made evaluator nTuples, it is easy to read the information in to Fun4All.
- Inside your C++ macro, you would have:

```
int Fun4All_example()
{
  gSystem->Load("libfun4all.so");
  gSystem->Load("libg4dst.so");
  Fun4AllServer *se = Fun4AllServer::instance();
  Fun4AllInputManager *hitsin = new Fun4AllDstInputManager("DSTin");
  hitsin->AddListFile(“myFileList.txt”);
  #Add your analysis modules here
  se->registerInputManager(hitsin);
  se->run(nEvents);
  se->End();
  return;
}
```

Pass an ASCII list of files (and paths if needed)

Conclusions

- Going from the physics you want to study to getting your data set is a reasonably straight forward process
- However, users must understand what they’re doing
  - A bad steering card with a good simulation wont provide good physics
  - A good steering card with a bad simulation setup wont provide this either
- It all comes down to communication; please don’t stumble in the dark or reinvent the wheel
- We have lots of references to help your development:
  - Doxygen: https://ecce-eic.github.io/doxygen/
  - Github: https://github.com/ECCE-EIC/
  - Software Documentation: https://ecce-eic.github.io/
  - Simulation Office Hours: https://indico.bnl.gov/event/11574/

Backup

Part 0

Setting up the environment

Getting the environment

- We make nightly builds of our setup to cvmfs
- There are 3 ways to access this (4 if you have a BNL SDCC account)
  - These 3 use a singularity container

Method	Pros	Cons
Ubuntu Virtual Box	Everything is prebuilt	Shares resources with host, drivers can cause compatibility issues
Local CVMFS	Works offline	Requires large downloads and user updates
Mounted CVMFS	Is always up-to-date	Requires online connection
RCF	Direct connection to our builds on BNL farm	Needs SDCC account, graphics are difficult to configure if running on a shell

Setting up the environment

- Each method has more specific instructions but in general do:
```
source /cvmfs/eic.opensciencegrid.org/ecce/gcc-8.3/opt/fun4all/core/bin/ecce_setup.sh -n
```
- Note: We also support tcsh
- This gets the latest build AND overwrites environment variables
  - So you won't use any local versions
- Our github repository is located here: https://github.com/ECCE-EIC

Using your own code

- If you want to modify or use your own compiled code, you will need to set the following:

A working folder

Where to install the package

Point to the libraries and headers

Set it all up for local use

```
export ECCE=/sphenix/user/cdean/ECCE
export MYINSTALL=$ECCE/install
export LD_LIBRARY_PATH=$MYINSTALL/lib:$LD_LIBRARY_PATH
export ROOT_INCLUDE_PATH=$MYINSTALL/include:$ROOT_INCLUDE_PATH
source /cvmfs/eic.opensciencegrid.org/ecce/gcc-8.3/opt/fun4all/core/bin/setup_local.sh $MYINSTALL
```

- tcsh equivalent is "`setenv MYINSTALL ${ECCE}/install`"
- Every package should have: configure.ac, Makefile.am and autogen.sh
- Build the package with:

```
mkdir build;
cd build;
../autogen.sh --prefix=$MYINSTALL;
make install;
```

Using custom macros

- If you checkout our macros repo, you can modify the files without compiling them
- You will need to set the following environment variable to see the changes though:

```
export ROOT_INCLUDE_PATH=${ECCE}/macros/common:${ROOT_INCLUDE_PATH}
```

(Assuming you downloaded the repo to $ECCE)