Delphes

Aleesha KT

IISER KOLKATA

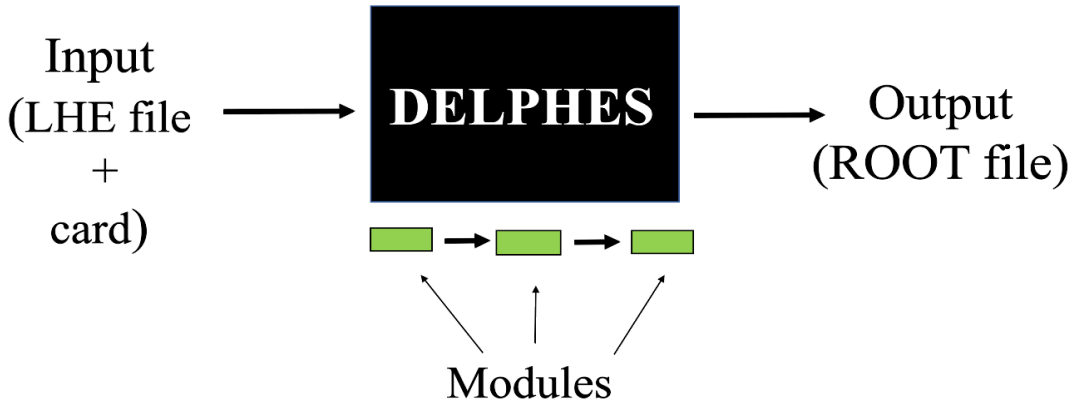
April 3, 2021

Outline

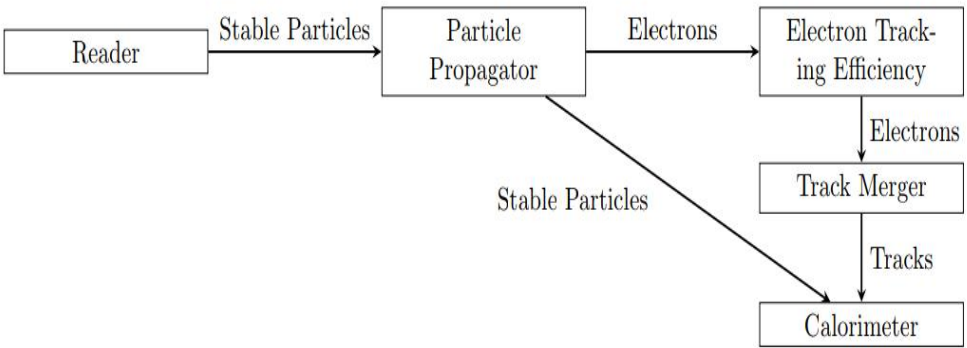
- Introduction to Delphes
- Basics of ROOT
- Macro-based analysis

Delphes

Framework which allows to simulate a multipurpose collider detector. It's based on a modular system. Each module is devoted to a function.

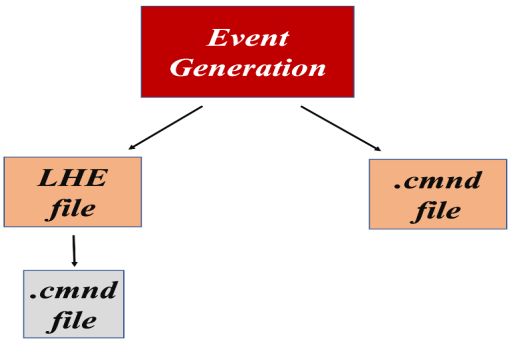

Input
(LHE file
+
card)
DELPHES
Output
(ROOT file)
Modules

Work-flow chart of DELPHES simulation



Delphes with Pythia8

- Running Delphes with Pythia8, requires 2 input files - .cmnd file and detector card.
- .cmnd file contains the commands to read and process the event file.



- You can find some sample cmnd files inside `Delphes/examples/Pythia8`.

Event generation with an LHE file

```
! 1) Settings used in the main program.

Main:numberOfEvents = 1000          ! number of events to generate
Main:timesAllowErrors = 3           ! how many aborts before run stops

! 2) Settings related to output in init(), next() and stat().

Init:showChangedSettings = on       ! list changed settings
Init:showChangedParticleData = off  ! list changed particle data
Next:numberCount = 100              ! print message every n events
Next:numberShowInfo = 1             ! print event information n times
Next:numberShowProcess = 1          ! print process record n times
Next:numberShowEvent = 1            ! print event record n times

! 3) Set the input LHE file

Beams:frameType = 4
Beams:LHEF = provide the path to your LHE file
```

Event generation without an LHE file

```
! 1) Settings used in the main program.
Main:numberOfEvents = 1000         ! number of events to generate
Main:timesAllowErrors = 3          ! how many aborts before run stops

! 2) Settings related to output in init(), next() and stat().
Init:showChangedSettings = on      ! list changed settings
Init:showChangedParticleData = off ! list changed particle data
Next:numberCount = 1000            ! print message every n events
Next:numberShowInfo = 1            ! print event information n times
Next:numberShowProcess = 1         ! print process record n times
Next:numberShowEvent = 0           ! print event record n times

! 3) Beam parameter settings. Values below agree with default ones.
Beams:idA = 2212                   ! first beam, p = 2212, pbar = -2212
Beams:idB = 2212                   ! second beam, p = 2212, pbar = -2212
Beams:eCM = 14000.                 ! CM energy of collision
Beams:frameType = 1
```

Event generation without an LHE file

```
! 4) Settings for the hard-process generation.
Top:gg2ttbar = on                ! g g -> t tbar
Top:qqbar2ttbar = on             ! q qbar -> t tbar


! 5) Switch on/off the key event generation steps.
PartonLevel:MPI = off            !  multiparton interactions
PartonLevel:ISR = on             !  initial-state radiation
PartonLevel:FSR = on             !  final-state radiation
HadronLevel:Hadronize = on       !  hadronization
HadronLevel:Decay = on           !  decays

! 6) Other settings. Can be expanded as desired.
PDF:pSet = 14                    ! NNPDF2.3 QCD+QED LO alpha_s(M_Z) = 0.130
SigmaProcess:renormScale2 = 4          ! Q^2 renormalization scale for 2 → 2 processes: m_{invariant}^{2} of
the system
SigmaProcess:factorScale2 = 4          ! Q^2 renormalization scale for 2 → 2 processes: m_{invariant}^{2} of
the system
```

Detector card

- Simulation is completely described by the card which contains the sequence of modules implemented.
- Can be found inside `cards` directory
- List of modules are arranged in the order of their execution.
- Allows us to write and add our own modules.

Running Delphes

```
cd Delphes
./DelphesPythia8 cards/delphes_card_CMS.tcl file_name.cmnd
output_file_name.root
```

ROOT Basics

ROOT is a data-analysis framework. In Delphes, the output file is produced in ROOT format. The information is stored in one of the data structure provided by ROOT called the `'Tree'` . In ROOT, large quantities of data can be stored as TTree, which has substructres called TBranch class inside which you can find variables called leaf that belongs to Tleaf[1]

- Official homepage of ROOT : https://root.cern/
- Link to CERN ROOT Forum : https://root-forum.cern.ch/

[1] Classes in ROOT have a prefix 'T' after which follows the name of the class

TTrees

- By default, the names of the Trees here are "Delphes".
- You will find various branches like Event, Particle, Track, Jet, Electron etc. Inside these branches say for example in 'Jet' Branch you can find pT(transverse momenta), eta(pseudorapidity), BTag(multiplicity of b-jets) etc.
- We can inspect the TTree by opening the ROOT file in TBrowser[2].

[2] The easiest way to identify the names of the Branches and leaves is by the inspection of ROOT files in TBrowser

TBrowser

```
root
TFile* f= new TFile("file_name.root","READ")
TBrowser f
```

- In the window that appears, click on your file and then click on the TTree named 'Delphes'.
- If you click on any of the leaves it will display the histogram in the window. You may even call these histograms separately in terminal.

ROOT Basics

- ROOT macro is a C++ code which uses ROOT classes and ROOT objects.
- The macro must be saved with '.C' extension .
- The name of the main() function must be same as the name of the macro.
- Include all the header files corresponding to the class you would need

ROOT Class	**Purpose**
TChain	collection of files containg TTree objects
ExRootTreeReader	takes ROOT Tree as parameter
TClonesArray	Obtain pointers to branches used in the analysis
TH1	1D Histograms
TCanvas	Creates a new canvas
TLorentzVector	four-vector class

ROOT Basics

- Do not forget to use '$\rightarrow$' while calling the variables defined as pointers. Here h1 was defined as pointer in Line 33. Say h1 was not defined as a pointer then the syntax would be `h1.Draw()`. i.e. all the arrows are replaced by dot.
- List of Arrays in Delphes
- List of classes used to store output data

Code structure

The following lines of code must be added while writing any macro. It loads the libraries and header files.

```
#ifdef __CLING__
R__LOAD_LIBRARY(libDelphes)
#include "classes/DelphesClasses.h"
#include "external/ExRootAnalysis/ExRootTreeReader.h"
#include "external/ExRootAnalysis/ExRootResult.h"
#else
class ExRootTreeReader;
class ExRootResult;
#endif
```

Code structure

```
void name_of_the_macro(const char *inputFile)
{
 //Load the shared delphes library from gSystem
 gSystem->Load("libDelphes");

 // Create chain of root trees
 TChain chain("Delphes");
 chain.Add(inputFile);

 // Create an object of class ExRootTreeReader
 ExRootTreeReader *treeReader = new ExRootTreeReader(&chain);
 Long64_t numberOfEntries = treeReader->GetEntries();

 // Get pointers to branches used in this analysis
 TClonesArray *branchJet = treeReader->UseBranch("Jet");
 TClonesArray *branchElectron = treeReader->UseBranch("Electron");
 TClonesArray *branchMuon = treeReader->UseBranch("Muon");
```

Code structure

Booking histograms

```
TH1 *h1 = new TH1D("name of the histogram", "histogram_title", No
of bins, lower limit, upper limit);
```

Loop over all events

```
for(Int_t entry = 0; entry < numberOfEntries; ++entry){
treeReader->ReadEntry(entry);}
```

Jet Branch

```
for(int i = 0; i < branchJet->GetEntries(); i++)
    {
// extract the jets in the branch
     Jet *jet = (Jet*) branchJet->At(i);
    //Jet transverse momenta
     h1->Fill(jet->PT);
     h2->Fill(jet->Eta);
     h3->Fill(jet->Phi);                                    }

//Jet Multiplicity
int n_jet = branchJet->GetEntries();
h4->Fill(n_jet );
```

Creating a new canvas

```
 TCanvas *c1 = new TCanvas("Title of Canvas window","Canvas
title");
gStyle->SetOptStat(0);
h1->GetXaxis()->SetTitle("Set X axis title");
h1->GetYaxis()->SetTitle("Set Y axis title");
h1->Draw();
```

Extracting electrons

```
TLorentzVector four_vec_e1;

for(Int_t k1 = 0; k1 < branchElectron->GetEntries(); k1++){

      //take the electrons in the branch and append their four
momentum vector
      Electron *e1 = (Electron *) branchElectron1->At(k1);

      four_vec_e1 = e1->P4();
      px_e->Fill(four_vec_e1.Px()));
}

//Electron Multiplicity
int n_e = branchElectron->GetEntries();
h5->Fill(n_e );
```

Particles in the Event Record

```
TClonesArray *branchParticle = treeReader->UseBranch("Particle");

for(Int_t j=0; j < branchParticle->GetEntries(); j++)
   {
      GenParticle *gen = (GenParticle*) branchParticle->At(j);
      if( (gen->Status ==1) && (abs(gen->PID)!=11) && (abs(gen->PID)!=13) && (abs(gen->PID)!=22) && gen->Eta <= eta_Max){
            pt_h->Fill((gen->PT));
            eta_h->Fill((gen->Eta));
            phi_h->Fill((gen->Phi));
      }
  }
```

Particles in the Event Record

- Status = 1 for Final state particles .
- Status = 2 for Decayed particles.
- Other attributes are PID, Px, Py, Pz, E, Mass, M1, M2, D1, D2.

References

- https://cp3.irmp.ucl.ac.be/projects/delphes/wiki/WorkBook/Pythia8
- http://home.thep.lu.se/~torbjorn/pythia82html/SUSYLesHouchesAccord.html
- https://www.niser.ac.in/sercehep2017/notes/RootTutorial_TTree.pdf
- https://root.cern/manual/
- https://cp3.irmp.ucl.ac.be/projects/delphes
- http://home.thep.lu.se/Pythia/


Thank
you