Delphes Project for 2015 Winter School on Collider Physics

```
====================================
Project 1: Follow the examples below
====================================


Example 1: Quick start with Delphes
===================================

0. Skip 1-4 if you already have Delphes installed.

1. Connect to LINUX:

2. Setup GCC and ROOT:

3. Commands to get the code:

  git clone https://github.com/delphes/delphes.git

4. Commands to compile the code:

  cd delphes

  make -j 4

5. Finally, we can run Delphes:

  ./DelphesSTDHEP

6. and simulate some pp -> tt~ (t->ch->cbb) events:

  wget http://hep.jbnu.ac.kr/downloads/t_ch_bb.hep

  ./DelphesSTDHEP cards/delphes_card_CMS.tcl step_1.root t_ch_bb.hep


Example 2: Simple analysis using TTree::Draw
============================================


Now we can start ROOT and look at the data stored on the tree

1. Start ROOT and load shared library:

  root -l
  gSystem->Load("libDelphes");

2. Open ROOT tree file and do some basic analysis using Draw or TBrowser:

  TFile::Open("step_1.root");
  Delphes->Draw("Muon.PT", "Muon.PT > 20");
  Delphes->Draw("Jet.PT", "Jet.BTag > 0");
  TBrowser browser;

Note 1: Delphes - tree name, it can be learnt e.g. from TBrowser

Note 2: Muon - branch name; PT - variable (leaf) of this branch

Complete description of all branches can be found in

  Delphes/doc/RootTreeDescription.html

This file is also available via web:

  http://cp3.irmp.ucl.ac.be/downloads/RootTreeDescription.html
```

```
Example 3: Macro-based analysis
===============================


Analysis macro consists of histogram booking, event loop (histogram filling),
histogram display

Delphes/examples contains macros Example1.C, Example2.C and Example3.C

1. Here are commands to run a macro reconstructing the Higgs invariant mass:

  git clone https://github.com/monttj/tutorial.git

  root -l -b -q tutorial/InvariantMass.C'("step_1.root", "step_1_plots.root")'

The step_1_plots.root file contains 2 histograms of number of b-jets and
b-bbar invariant mass


Example 4: Modifying configuration file
=======================================

1. Change the b-tagging efficiency in the configuration file:

  edit cards/delphes_card_CMS.tcl

replace b-tagging efficiency with:

  add EfficiencyFormula {0} {0.01}

  # efficiency formula for c-jets (misidentification rate)
  add EfficiencyFormula {4} {                                    (pt <= 15.0) * (0.000) +
                                          (abs(eta) <= 1.2) * (pt > 15.0) * (0.3*tanh(pt*0.03 - 0.4)) +
                             (abs(eta) > 1.2 && abs(eta) <= 2.5) * (pt > 15.0) * (0.2*tanh(pt*0.03 - 0.4)) +
                             (abs(eta) > 2.5)                                  * (0.000)}

  # efficiency formula for b-jets
  add EfficiencyFormula {5} {                                    (pt <= 15.0) * (0.000) +
                                          (abs(eta) <= 1.2) * (pt > 15.0) * (0.6*tanh(pt*0.03 - 0.4)) +
                             (abs(eta) > 1.2 && abs(eta) <= 2.5) * (pt > 15.0) * (0.5*tanh(pt*0.03 - 0.4)) +
                             (abs(eta) > 2.5)                                  * (0.000)}


2. rerun Delphes:

  ./DelphesSTDHEP cards/delphes_card_CMS.tcl step_2.root t_ch_bb.hep

3. redo the invariant mass histograms:

  root -l -b -q InvariantMass.C'("step_2.root", "step_2_plots.root")'

4. creare .rootlogon.C that contains (to load DELPHES library automatically)

{

  gSystem->Load("libDelphes.so");

}

5. compare histograms:

  root -l step_1_plots.root step_2_plots.root

  ((TH1F *)_file0->Get("nbjet"))->SetLineColor(kBlue);
```

```
  ((TH1F *)_file1->Get("nbjet"))->SetLineColor(kRed);

  _file0->Get("nbjet")->Draw();
  _file1->Get("nbjet")->Draw("SAME");

6. You could also try to change the energy resolution of jets.

Example 5: Adding new module
============================

1. Create file modules/JetCleaning.h with the following content (see below)

2. Create file modules/JetCleaning.cc with the following content (see below)

3. Add new module to modules/ModulesLinkDef.h:

#include "modules/JetCleaning.h"
...
#pragma link C++ class JetCleaning+;

4. Add DeltaR variable to Candidate class and Muon class in classes/DelphesClasses.h
and in classes/DelphesClasses.cc

5. Modify TreeWriter::ProcessMuons() in modules/TreeWriter.cc to store DeltaR:

entry->DeltaR = candidate->DeltaR

6. Regenerate Makefile and rebuild Delphes:
./configure
make -j 4

7. Add new module to the configuration file:

set ExecutionPath {
  ...
  GenJetFinder
  FastJetFinder

  ...
  UniqueObjectFinder

  MuonClean
  ...
}

module JetCleaning MuonClean {

  set CandidateInputArray UniqueObjectFinder/muons
  set JetInputArray  UniqueObjectFinder/jets

  set OutputArray muons

  set DeltaRMin 0.0
}


8. Add new ROOT tree branch to the configuration module:
module TreeWriter TreeWriter {
  ...
  add Branch MuonClean/muons MuonClean Muon
  ...
}

9. Now you can rerun Delphes:

  ./DelphesSTDHEP cards/delphes_card_CMS.tcl step_3.root t_ch_bb.hep
```

10. Have a look at the newly stored deltaR variable:

```
root -l step_3.root
Delphes->Draw("MuonClean.DeltaR", "MuonClean.PT > 20 && MuonClean.DeltaR > 0.5")
gPad->SetLogy()
```

****

Here are the new module source files:

```
//------------------------------------------------------------------------
// modules/JetCleaning.h
//------------------------------------------------------------------------

#ifndef JetCleaning_h
#define JetCleaning_h

#include "classes/DelphesModule.h"

class TObjArray;

class ExRootFilter;

class JetCleaning: public DelphesModule
{
public:

  JetCleaning();
  ~JetCleaning();

  void Init();
  void Process();
  void Finish();

private:

  Double_t fDeltaRMin;

  TIterator *fItCandidateInputArray; //!

  TIterator *fItJetInputArray; //!

  const TObjArray *fCandidateInputArray; //!

  const TObjArray *fJetInputArray; //!

  TObjArray *fOutputArray; //!

  ClassDef(JetCleaning, 1)
};

#endif


//------------------------------------------------------------------------
// modules/JetCleaning.cc
//------------------------------------------------------------------------

#include "modules/JetCleaning.h"

#include "classes/DelphesClasses.h"
#include "classes/DelphesFactory.h"
#include "classes/DelphesFormula.h"
```

```
#include "ExRootAnalysis/ExRootResult.h"
#include "ExRootAnalysis/ExRootFilter.h"

#include "TMath.h"
#include "TString.h"
#include "TFormula.h"
#include "TRandom3.h"
#include "TObjArray.h"
#include "TDatabasePDG.h"
#include "TLorentzVector.h"
#include "TVector3.h"

#include <algorithm>
#include <stdexcept>
#include <iostream>
#include <sstream>

using namespace std;


//------------------------------------------------------------------------------

JetCleaning::JetCleaning() :
  fItCandidateInputArray(0), fItJetInputArray(0)
{
}

//------------------------------------------------------------------------------

JetCleaning::~JetCleaning()
{
}

//------------------------------------------------------------------------------

void JetCleaning::Init()
{
  fDeltaRMin = GetDouble("DeltaRMin", 20);

  // import input array(s)

  fCandidateInputArray = ImportArray(GetString("CandidateInputArray", "MuonMomentumSmearing/muons"));
  fItCandidateInputArray = fCandidateInputArray->MakeIterator();

  fJetInputArray = ImportArray(GetString("JetInputArray", "FastJetFinder/jets"));
  fItJetInputArray = fJetInputArray->MakeIterator();

  // create output array

  fOutputArray = ExportArray(GetString("OutputArray", "electrons"));
}


void JetCleaning::Finish()
{
  if(fItJetInputArray) delete fItJetInputArray;
  if(fItCandidateInputArray) delete fItCandidateInputArray;
}

//------------------------------------------------------------------------------

void JetCleaning::Process()
{
  Candidate *jet, *candidate;
  Double_t deltaR = 5.0;
```

```
  // loop over all input leptons

  fItCandidateInputArray->Reset();
  while((candidate = static_cast<Candidate*>(fItCandidateInputArray->Next())))
  {
    const TLorentzVector &candidateMomentum = candidate->Momentum;

    // loop over all input jets
    fItJetInputArray->Reset();

    while((jet = static_cast<Candidate*>(fItJetInputArray->Next())))
    {
      const TLorentzVector &jetMomentum = jet->Momentum;

      Double_t dr = candidateMomentum.DeltaR(jetMomentum);

      if( dr < deltaR ) deltaR = dr;
    }

    candidate->DeltaR = deltaR;

    if(deltaR < fDeltaRMin) continue;

    fOutputArray->Add(candidate);
  }
}

//------------------------------------------------------------------------------
```

Example 6: Run Delphes with pile-up
==========

1. Start from the original configuration file:

```
  cp cards/delphes_card_CMS.tcl cards/delphes_card_CMS_PileUp_nosub.tcl
```

2. add "PileUpMerger" module and replace "ParticlePropagator" in the configuration "delphes_card_CMS_PileUp_nosub.tcl":

```
set ExecutionPath {
  PileUpMerger
  ParticlePropagator
  ...
}

###############
# PileUp Merger
###############

module PileUpMerger PileUpMerger {
  set InputArray Delphes/stableParticles

  set ParticleOutputArray stableParticles
  set VertexOutputArray vertices

  # pre-generated minbias input file
  set PileUpFile MinBias.pileup

  # average expected pile up
  set MeanPileUp 50

  # maximum spread in the beam direction in m
  set ZVertexSpread 0.10

  # maximum spread in time in s
  set TVertexSpread 1.5E-09
```

```
  # vertex smearing formula f(z,t) (z,t need to be respectively given in m,s)

  set VertexDistributionFormula {exp(-(t^2/(2*(0.05/2.99792458E8*exp(-(z^2/(2*(0.05)^2))))^2)))}

  #set VertexDistributionFormula { (abs(t) <= 1.0e-09) * (abs(z) <= 0.15) * (1.00) +
  #                                (abs(t) >  1.0e-09) * (abs(z) <= 0.15) * (0.00) +
  #                               (abs(t) <= 1.0e-09) * (abs(z) > 0.15)  * (0.00) +
  #                               (abs(t) >  1.0e-09) * (abs(z) > 0.15)  * (0.00)}


}

##################################
# Propagate particles in cylinder
##################################

module ParticlePropagator ParticlePropagator {
  set InputArray PileUpMerger/stableParticles

  set OutputArray stableParticles
  set ChargedHadronOutputArray chargedHadrons
  set ElectronOutputArray electrons
  set MuonOutputArray muons

  # radius of the magnetic field coverage, in m
  set Radius 1.29
  # half-length of the magnetic field coverage, in m
  set HalfLength 3.00

  # magnetic field
  set Bz 3.8
}
```

3. Check if you have the Minimum Bias file in your local directory (used for pile-up merging)

```
  ls MinBias.pileup
```

4. Run the configuration:

```
  ./DelphesSTDHEP cards/delphes_card_CMS_PileUp_nosub.tcl step_4.root t_ch_bb.hep
```

5. redo the invariant mass histograms:

```
  root -l -b -q InvariantMass.C'("step_4.root", "step_4_plots.root")'
```

6. compare histograms:

```
  root -l step_1_plots.root step_4_plots.root

  ((TH1F *)_file0->Get("mbb"))->SetLineColor(kBlue);
  ((TH1F *)_file1->Get("mbb"))->SetLineColor(kRed);

  _file0->Get("mbb")->Draw();
  _file1->Get("mbb")->Draw("SAME");
```

7. Run delphes with the configuration "delphes_card_CMS_PileUp.tcl":

```
  ./DelphesSTDHEP cards/delphes_card_CMS_PileUp.tcl step_5.root t_ch_bb.hep
```

8. compare histograms again and see what is changed. Can you explain why we see such a difference?

```
==============================
Project 2: Adding new variable
==============================
```

Currently there is no isolation variable for muon and electron object that

```
can be used at the analysis level in DELPHES.
Instead, the isolation requirement is applied at the production level.
We can add isolation variable and remove the isolation requirement in DELPHES card
so that we can apply the isolation selection later when we do the analysis.

1. Add RelIso variable to Candidate class and Muon and Electron class in classes/DelphesClasses.h
and in classes/DelphesClasses.cc

2. Modify TreeWriter::ProcessMuons() and TreeWriter::ProcessElectrons() in modules/TreeWriter.cc to store RelIso:

entry->RelIso = candidate->RelIso

3. Regenerate Makefile and rebuild Delphes:
./configure
make -j 4

4. Start from the original configuration file:

  cp cards/delphes_card_CMS.tcl cards/delphes_card_CMS_noiso.tcl

5. Remove isolation requirement in the configuration file, cards/delphes_card_CMS_nosio.tcl:

module Isolation MuonIsolation {
  ...
  set PTRatioMax 999
}

module Isolation ElectronIsolation {
  ...
  set PTRatioMax 999
}

6. Now you can rerun Delphes using ttbar events:

  wget http://hep.jbnu.ac.kr/downloads/ttbar.hep

  ./DelphesSTDHEP cards/delphes_card_CMS_noiso.tcl step_6.root ttbar.hep

7. Have a look at the newly stored RelIso variable:

root -l step_6.root
Delphes->Draw("Muon.RelIso", "Muon.PT > 20")
gPad->SetLogy()

8. You can also do the macro-based analysis.

root -l -b -q tutorial/IsoAnalyzer.C'("step_6.root", "step_6_plots.root")'

The step_6_plots.root file contains 2 histograms of number of leptons and
relative isolation variable distribution.

9. Can you repeat the example 6 in Project 1 with pileup for the RelIso variable and compare?

================================================
Project 3: Reconstruct top quark four-momentum
================================================

1. How would you reconstruct the top quark four-momentum in pp > tt~ (t->ch->cbb) events?

2. Can you draw the top quark mass distribution?


=========================================================
Project 4: Searching for flavor changing neutral current
=========================================================
```

1. To prepare background sample, download the t-tbar background sample and produce delphes root file:

   wget http://hep.jbnu.ac.kr/downloads/ttbar.hep (Skip if it is already done.)

   ./DelphesSTDHEP cards/delphes_card_CMS_PileUp.tcl background.root ttbar.hep

   Let's use t_ch_bb.hep file to produce the delphes root file for the signal events.

   ./DelphesSTDHEP cards/delphes_card_CMS_PileUp.tcl signal.root t_ch_bb.hep

2. what is your findings with following questions?

   (a) For two years from 2015 and 2016, assume that LHC will be accumulating the proton-proton collision data with an instantaneous luminosity of 2×10^{33} cm^{-2}s^{-1} at the center of mass energy, $\sqrt{s} = 13$ TeV. The LHC is operational for 80 % of the time during this period. The $t\bar{t}$ background cross section is 680 pb in the standard model. And the predicted branching ratio of t $\rightarrow$ ch in new physics is 0.5 %. Normalize the signal and background events to the integrated luminosity you will get for two years: (H $\rightarrow$ $b\bar{b}$ branching fraction is 0.577 at the Higgs mass of 125.0 GeV/c^2.)

   (b) Can you find the optimized selection (e.g. number of muons, jets with the transverse momentum 20 GeV) so that you can have the largest significance using the number of signal (S) events over background (B) events, $S/\sqrt{\mathrm{B}}$?

   (c) What is the signal acceptance with your selection?