ECAl Tutorial

v.1 (February, 2022) by V. Riabov

1. Geometry.

Current version of ECAL geometry is v.4. Please make sure that this version of ECAL geometry is called in mpdroot/macro/mpd/geometry_stage1.C. The following lines should be verified:

```
FairDetector *Emc = new MpdEmcKI("EMC", kTRUE);
Emc->SetGeometryFileName("emc_v4.root"); // geometry version
fRun->AddModule(Emc);
```

Details of the ECAL geometry can be found in the recent presentation by M. Martemyanov, https://indico.jinr.ru/event/2893/contributions/15531/attachments/11967/19918/Report-Martemianov_2893.pdf

2. Running ECAL simulations.

There are two different versions of the ECAl digitizer/clusterizer available in the MpdRoot:
1) mpdroot/detectors/emc/MpdHitCreation, mpdroot/detectors/emc/MpdClusterCreation – version of the code to work with standalone clusters (one particle per event).
2) mpdroot/detectors/emc/emcKI/ MpdEmcDigitizerKI.cxx, mpdroot/detectors/emc/emcKI/ MpdEmcClusterizerKI.cxx – version of the code that works as with standalone clusters as with high multiplicity collisions.

Code (1) is intended for quick simulation of the detector parameters and comparison to beam tests results. Code (2) is more universal and can deal with overlapping showers in high multiplicity collisions. Results of codes (1) and (2) for single particles may slightly differ due to different algorithms for cluster formation.

For physics studies it is recommended to use code (2). In order to run the code the following lines need to be added in reco.C:

```
//emcKI digitizer
   FairTask *emcHP = new MpdEmcDigitizerKI();
   fRun->AddTask(emcHP);

// emcKI clusterizer
   MpdEmcClusterizerKI *EmcCluster = new MpdEmcClusterizerKI();
   fRun->AddTask(EmcCluster);

// emcKI matching
   MpdEmcMatchingKI * EmcMatcher = new MpdEmcMatchingKI() ;
   fRun->AddTask(EmcMatcher) ;
```

3. Basic operation principles of emcKI code.

Details can be found in the Physics Forum presentation, https://indico.jinr.ru/event/1097/. The operation principles are based on splitting (unfolding) of partly overlapping showers occurring in high multiplicity collisions. The splitting is based on the known shape of electromagnetic showers.

4. Data handling

Produced DST files have an mpdEMCClusters array of MpdEmcClusterKI clusters. Please find below a simple example how to get access to the array and the clusters:

```
#include "MpdEmcClusterKI.h"

...................
  TTree *inTree;
  inTree = (TTree*) inFile->Get("mpdsim");

...................
  TObjArray * mpdEMCClusters  = new TObjArray() ;
  MpdEmcClusterKI* EMCCluster;

  inTree->SetBranchAddress("EmcCluster",&mpdEMCClusters);

...................
    for (long int j=0; j<mpdEMCClusters->GetEntries(); j++){
      EMCCluster = (MpdEmcClusterKI*)mpdEMCClusters->At(j);
      // do whatever you need with a given cluster EMCCluster here …
    }
```

5. Cluster variables

Properties and quality of each cluster are controlled with a number of variables listed below:

//===============
EMCCluster->GetMultiplicity();// number of towers in the cluster.
Usually clusters consist of a few towers. Clusters that consist of one tower only are either very low energy clusters or just fragments of other (higher energy) showers. The cluster energy is counted as a sum of tower energies. The recommendation is to select clusters with number of towers > 1.

//===============
EMCCluster->GetE(); // reconstructed energy of the cluster
Please note that EMCCluster->GetE() returns cluster energy collected in the simulations. However, ~ 70% of the energy deposited by a particle in the ECAL is not collected, it is absorbed by lead or other construction materials. It means that in order to get the true particle energy deposition the simulated energy should be corrected. For v.4 geometry of the ECAL we recommend the following corrections:

```
float     conv = 1.0/0.3065; // only ~ 30% energy is collected
..................
float E_true = EnCorr( EMCCluster->GetE() * conv );
```

Function EnCorr() compensates for non-linearity (3% effect)

```
..................
Float EnCorr(float e)
{
  float x = e;
  if (x > 2.5) x = 2.5; // parameterization is available up to 2.5 GeV only

  float corr[6] = { -1.321760e-002, 8.475623e-002, -9.007282e-002, 4.742396e-002, -1.279561e-002, 1.397394e-003 };
  float ecorr = corr[0] + corr[1]*x + corr[2]*x*x + corr[3]*x*x*x + corr[4]*x*x*x*x + corr[5]*x*x*x*x*x;

  return e/(1.0 + ecorr);
}
```

Recommendation is to select clusters with E_true > 0.05 (50 MeV) in the physical analyses.

//===============

EMCCluster->GetEcore_1p() // truncated reconstructed energy of the cluster

This is pretty much the same variable as EMCCluster->GetE() but in this case the cluster energy is counted only for towers, which contribute > 1% to the total cluster energy. In general, EMCCluster->GetEcore_1p() is smaller than EMCCluster->GetE() and it requires different non-linearity corrections. The EMCCluster->GetEcore_1p() energy is less sensitive to shower overlaps although the energy resolution for this variable is generally worse due to smaller collected energy. Use this variable at your discretion.

//===============

EMCCluster->GetEcore_2p() // truncated reconstructed energy of the cluster

This is pretty much the same variable as EMCCluster->GetE() but in this case the cluster energy is counted only for towers, which contribute > 2% to the total cluster energy. In general, EMCCluster->GetEcore_2p() is smaller than EMCCluster->GetE() and EMCCluster->GetEcore_1p() and it requires different non-linearity corrections. The EMCCluster->GetEcore_2p() energy is even less sensitive to shower overlaps although the energy resolution for this variable is generally worse due to smaller collected energy. Use this variable at your discretion.

//===============

```
EMCCluster->GetX(); // x-coordinate of the cluster
EMCCluster->GetY(); // y-coordinate of the cluster
EMCCluster->GetZ(); // z-coordinate of the cluster
EMCCluster->GetRho(); // radius of the cluster, sqrt(x**2+y**2)
```

These are X,Y,Z coordinates of the cluster in the global coordinate system. The coordinates are for the cluster center of gravity. Hence, cluster radius or zed should not necessary coincide with the detector surface or center of the tower with the largest energy deposition. Recommendation is to use these coordinates to calculate px, py, pz momentum components.

//===============
EMCCluster->GetChi2(); // chi2/NDF of the cluster
This variable says how close the cluster shape to the one expected for electromagnetic shower. The clusters from photons and high-pT electrons are expected to have electromagnetic shape in the ECAL. The shape analysis is possible only is there are more than one tower in the cluster. The larger the number of towers (the higher the energy) the more reliable the results of shower shape analysis. Recommendation is to select clusters with Chi2 < 4 if interested to measure photons.
Please note that clusters from low-momentum electrons are highly asymmetric due to a large incident angle of the electron tracks bent in the magnetic field. As a result, one cannot expect electromagnetic shape for such clusters.

//===============
EMCCluster->GetTime(); // simulated time of the shower
This variable is a simulated time of the shower. The meaning of this variable is close to intrinsic time measured in the ECAL. The real ECAL time resolution is defined by parameters of the electronics, not by intrinsic time resolution of the towers. For realistic timing please smear the simulated time with a resolution of 0.5 ns:

```
  TRandom RND;
  Tcl = RND.Gaus(EMCCluster->GetTime(), 0.5);
```

//===============
EMCCluster->GetDPhi(); // distance from the cluster to the closets TPC track in dPhi
EMCCluster->GetDZ(); // distance from the cluster to the closets TPC track in dZed
These variables help to assert whether the ECAL cluster is initiated by a charged particle or not. The decision is to be made based on n-sigma(pT) deviation of the cluster from the closest track.

//===============
EMCCluster -> GetNumberOfTracks(); // number of MC tracks that contribute to the cluster (only 5 top contributors are saved)
MCCluster -> GetMCTrack(track, MC track index, energy contribution); // returns energy contribution and MC track index for track-th contributor (track < EMCCluster -> GetNumberOfTracks()).
These variables are used to understand what MC tracks contribute to the measured cluster energy. In low multiplicity collisions only one MC track contributes to the cluster and EMCCluster -> GetNumberOfTracks() is equal to one. In high multiplicity events where most of the showers overlap it is impossible to set unambiguous association between the clusters and MC tracks. Instead, up to five MC tracks that contribute the most to the cluster energy are stored. The contributions are sorted by energy so that the first contributor is the most important. See an example below:

```
      long int ind_mc_cont;
      float      e_mc_cont;

      for (int kk = 0; kk < EMCCluster -> GetNumberOfTracks(); kk++){
        EMCCluster -> GetMCTrack(kk,ind_mc_cont,e_mc_cont);
        ind_mc_cont = MC index of the kk-th contributor to the cluster
        e_mc_cont - energy deposited by kk-th continutor
      }
```

//===============

If needed, one can gain full access to the list of towers in the cluster. It might be useful for recalibration purposes. Please see an example below:

```
  MpdEmcGeoUtils* geom = MpdEmcGeoUtils::GetInstance();
………………
   for(long int j=0; j<mpdEMCClusters->GetEntries(); j++){
     EMCCluster = (MpdEmcClusterKI*)mpdEMCClusters->At(j);

     // Towers
     int    detID;
     float  eTow;
     float  xi, yi, zi, phii;
     int    iphi, iz, izmax, iphimax;
     float  eTowMax = 0;


       for (int tower = 0; tower < EMCCluster->GetMultiplicity(); tower++)
         {
          EMCCluster->GetDigitParams(tower, detID, eTow);
          // detID – internal ID of the tower, eTow – energy of the tower

          geom->DetIdToGlobalPosition(detID, xi, yi, zi);
          phii = atan2(yi,xi);
          // xi, yi, zi, phii - global coordinates of the tower

          geom->DetIdToGlobalIphiIz(detID, iphi, iz);
          // iphi and iz – internal indexes of the tower in phi and zed

          if (eTow > eTowMax)
          {
            eTowMax = eTow;
            izmax = iz;
            iphimax = iphi;
          }
         }

      }//j
```

// eTowMax, izmax, iphimax - energy and indexes of the tower with the maximum energy deposition in the cluster

6. Photon identification

Photons can be identified using the following criteria:
1) Shower shape. Recommendation is to select clusters with EMCCluster->GetChi2() < 4. The tighter the cut the higher the probability that the cluster is produced either by a single photon or by high-energy electron. At this the tighter the selection the lower the reconstruction efficiency. The higher the photon energy the higher the chance of correct identification.
2) Time of flight. Photons trajectories do not bend in the magnetic field and they travel with a speed of light. Hence one can expect smaller time of flight for photon signals. Recommendation

is to select clusters with Tcl < 2 ns. The cut is quite efficient at low cluster energies where signal-to-background ratio is the smallest.
3) Charged particle veto. Charged particle tracks reconstructed in the TPC are extrapolated to the cluster radius and the distances in dPhi and dZed from a cluster to the closest track are calculated. These deviations can be parameterized as a function of cluster energy and then n-sigma(E_true) selection can be used to reject clusters reconstructed too close to the charged tracks (presumably initiated by charged particles). For rough estimations one can use a simple cut for photon selection: !(fabs(EMCCluster->GetDPhi()) < 15 && fabs(EMCCluster->GetDZ()) < 15). Please note that the charged particle veto cut is very efficient in low multiplicity events. It remains to be efficient in high multiplicity events, however, due to higher probability to find a charged track next to the true photon signal the reconstruction efficiency decreases.
The photon identification cuts can be used together. However, one should keep in mind that the cuts are correlated to some extent. It means that efficiency of multiple selections won't be equal to the product of efficiencies for each of them.

7. Charged particle identification

ECAL has nothing to offer in addition to hadron identification in the TOF subsystem. However, ECAL can be used for efficient identification of electrons. Electrons are identified in the ECAL by E/p ratio and time of flight. The identification procedure consists of a few steps:
1) Select a high quality track. The definitions depend on a particular physical analysis.
2) Identify track as electron in the TPC (by dE/dx) and TOF (by time of flight).
3) Extrapolate track to the ECAL surface and find the closest ECAL cluster.
4) Verify that the distance from the extrapolated track to the closest ECAL cluster is within 2-3 sigma in dPhi and dZed or within a constant limit optimized for a given physical analysis.
5) Determine energy and time of flight of the ECAL cluster associated with the track in p.4.
6) Calculate E/p ratio and compare it to the value expected for true electron with the same energy or momentum.
7) Calculate mass^2 of the particle and compare it to the value expected for true electron with the same energy or momentum.

Below please find some hints for each of the steps:

//===============
p.1. Use common sense to select high quality tracks. Usually these are tracks with 10-40 hits in the TPC matched to the primary vertex within 2-3 sigma. If you are interested in conversion pairs then on the contrary tracks should not point to the primary vertex.

//===============
p.2. Tracks are usually identified in the TPC and TOF by making 2-3 sigma(p, charge sign) selections on the measured values of dEdx in the TPC and mass^2 (or beta) values measured for tracks matched to the TOF. Discussion of electron identification using TPC and TOF subsystems is beyond the scope of this tutorial.

//===============
p.3. Recommendation is to use the following algorithm for association of tracks and clusters in the ECAL:

```
//Definitions
  MpdTpcKalmanTrack *tr = (MpdTpcKalmanTrack*) mpdKalmanTracks->UncheckedAt(j);

..................
//So we have a selected mpd track. Lets find the closest ECAL cluster

//Matching to ECAL
        dDmin = 1e99;

        if (fabs((*tr->GetParamAtHit())(1,0)) < 310) {
           MpdKalmanHit hEnd;
           hEnd.SetType(MpdKalmanHit::kFixedR);

           //List clusters
           for (Int_t clu = 0; clu < mpdEMCClusters->GetEntries(); clu++) {
              EMCCluster = (MpdEmcClusterKI*) mpdEMCClusters->At(clu);

//Cluster coordinates
              xCl = EMCCluster->GetX();
              yCl = EMCCluster->GetY();
              zCl = EMCCluster->GetZ();
              rCl = sqrt(xCl*xCl + yCl*yCl);
              phiCl = ATan2(yCl, xCl);

              hEnd.SetPos(rCl);

//Copy Kalman track --> tr1
              MpdTpcKalmanTrack tr1(*tr);//new
              tr1.SetParam(*tr1.GetParamAtHit());
              tr1.SetParamNew(*tr1.GetParamAtHit());
              tr1.SetWeight(*tr1.GetWeightAtHit());
              tr1.SetPos(tr1.GetPosAtHit());
              tr1.SetPosNew(tr1.GetPos());
              tr1.SetLength(tr1.GetLengAtHit());

              iok = pKF->PropagateToHit(&tr1,&hEnd,kTRUE); // extrapolation

              if (iok) {
                // track coordinates at the radius of the ECAL cluster
                 phiTr = tr1.GetParamNew(0) / tr1.GetPosNew();
                 xTr = tr1.GetPosNew() * cos(phiTr);
                 yTr = tr1.GetPosNew() * sin(phiTr);
                 zTr = tr1.GetParamNew(1);

                 dD = sqrt( pow(xTr-xCl,2) + pow(yTr-yCl,2) + pow(zTr-zCl,2) );

                 if (dD < dDmin) {
                    dDmin = dD;
                E_track = EMCCluster->GetE();
                dPhi_track = phiTr - phiCl;
                if (dPhi_track < -pi) dPhi_track = dphiTr_mpd[tr_count] + 2*pi;
                if (dPhi_track >  pi) dPhi_track = dphiTr_mpd[tr_count] - 2*pi;
                dZed_track = zTr - zCl;
                ToF_track = EMCCluster->GetTime()
                Length_track = tr1.GetLength();
              }//min
            }//iok
          }//clu
        }
      //Matching done
```

Results:
float E_track – simulated energy of the closest ECAL cluster;
float dPhi_track – distance in phi to the closest ECAL cluster;
float dZed_track – distance in zed to the closest ECAL cluster;
float Length_track – length of the tracks to the ECAL cluster
float ToF_track – simulated time of flight of the tracks to the ECAL cluster

//===============
p.4. Now for each track you have E_track, dPhi_track, dZed_track. Normalize these variables and use 2-3 sigma selections as a function of particle transverse momentum and charge sign. As an example you can use the following function to test 2 sigma matching of the track to the ECAL cluster:

```
int TestMatch( int charge, float pt, float dphi, float dz )
{
  if (pt < 0.1) return 0;
  if (pt > 1.0) pt = 1.0;

  //dPhi (charge>0)
  float meanphiCoeff[6] = { -3.231339e-002, 1.078172e-001, -1.315846e-001, 7.660724e-002, -
2.138274e-002, 2.299298e-003 };
  float widthphiCoeff[6] = { 1.154620e-001, -7.403730e-001, 2.249572e+000, -3.435599e+000,
2.558980e+000, -7.380640e-001 };

  float widthzCoeff[6] = { 8.452819e+000, -2.572417e+001, 4.050086e+001, -3.097804e+001,
1.138158e+001, -1.603740e+000 };

  float meanphi, widthphi, meanz, widthz;

  meanphi = meanphiCoeff[0] +
          meanphiCoeff[1] * pt +
          meanphiCoeff[2] * pt*pt +
          meanphiCoeff[3] * pt*pt*pt +
          meanphiCoeff[4] * pt*pt*pt*pt +
          meanphiCoeff[5] * pt*pt*pt*pt*pt;

  if (charge < 0) meanphi = -meanphi;

  widthphi = widthphiCoeff[0] +
           widthphiCoeff[1] * pt +
           widthphiCoeff[2] * pt*pt +
           widthphiCoeff[3] * pt*pt*pt +
           widthphiCoeff[4] * pt*pt*pt*pt +
           widthphiCoeff[5] * pt*pt*pt*pt*pt;

  meanz = 0.0;

  widthz = widthzCoeff[0] +
         widthzCoeff[1] * pt +
         widthzCoeff[2] * pt*pt +
         widthzCoeff[3] * pt*pt*pt +
         widthzCoeff[4] * pt*pt*pt*pt +
         widthzCoeff[5] * pt*pt*pt*pt*pt;

  if ( sqrt( pow((dphi-meanphi)/widthphi,2) +  pow((dz-meanz)/widthz,2) ) < 2.0 ) return 1;

  return 0;
}
```

Only tracks matched to the ECAL within 2-3 aigma can be considered for the further analysis. Due to track bending in the magnetic field only tracks with p_T larger than ~0.2 GeV/c are matched to the ECAL clusters.

//===============
p.5. Simulated cluster energy and time of flight are not identical to the energy deposited by the particle and the truly measured time of flight, please see Chapter 4 for details. So the track energy (E_track_true) and time of flight (ToF_track_true) are determined as:

```
float    E_track_true = EnCorr( E_track * conv );
float    ToF_track_true = RND.Gaus(ToF_track, 0.5);
```

//===============
p.6. E/p ratio for true electrons is not symmetric and has a rather strong dependence on electron energy/momentum. As a result, there is no sense to use tight cuts on E/p ratio to preserve a reasonable reconstruction efficiency. Recommendation is to use the following function to select electrons by E/p, TestEP(float p_full_momentum_from_TPC, float E_track_true):

```
int TestEP( float p, float E )
{
  float p1 = p;

  if (p1 < 0.15) return 0;
  if (p1 > 1.5) p1 = 1.5;

  float  nsigma = 3.0;
  if ( p1 <= 0.5 ) nsigma = 4.0;
  if ( p1 <= 0.35 ) nsigma = 5.0;

  float meanCoeff[6] = { 3.305878e-001, 3.159067e+000, -6.543087e+000, 6.687606e+000, -
3.316007e+000, 6.341009e-001 };
  float widthCoeff[6] = { 1.484982e-001, -2.321481e-001, 2.680616e-001, -8.780050e-002, -
2.023283e-002, 1.036833e-002 };

  float mean =  meanCoeff[0] +
              meanCoeff[1] * p1 +
              meanCoeff[2] * p1*p1 +
              meanCoeff[3] * p1*p1*p1 +
              meanCoeff[4] * p1*p1*p1*p1 +
              meanCoeff[5] * p1*p1*p1*p1*p1;

  float width =  widthCoeff[0] +
               widthCoeff[1] * p1 +
               widthCoeff[2] * p1*p1 +
               widthCoeff[3] * p1*p1*p1 +
               widthCoeff[4] * p1*p1*p1*p1 +
               widthCoeff[5] * p1*p1*p1*p1*p1;

  if ( E/p >  (mean - nsigma*width)  && E/p < 1.5 ) return 1;

  return 0;
}
```

//===============
p.7. mass^2 is defined by the particle path length to the matched ECAL cluster (Length_track), time of flight (ToF_track_true) and full particle momentum measured in the TPC (p):
mass2 = p*p * (ToF_track_true * ToF_track_true *30*30/ Length_track /Length_track - 1);

Normalize mass2 variable as a function of particle full momentum (p) and use 2-3 sigma selections for electrons. As an example you can use the following function to test 2 sigma consistence of the measured mass2 with the value expected for an electron:

```
int TestPID( float p, float mass2 )
{
 if ( p < 0.1 ) return 0.0;
 if ( p > 2.2 ) return 1.0; //everything is consistent with electron

 float meanCoeff[7] = { -7.291128e-003, -8.512899e-003, 2.687213e-001, -1.176928e+000,
1.708611e+000, -1.038707e+000, 2.242485e-001 };

 float widthCoeff[7] = { 1.798396e-002, -1.943139e-001, 9.359176e-001, -1.590746e+000,
1.652832e+000, -8.260310e-001, 1.547639e-001 };

 float mean =  meanCoeff[0] +
            meanCoeff[1] * p +
            meanCoeff[2] * p*p +
            meanCoeff[3] * p*p*p +
            meanCoeff[4] * p*p*p*p +
            meanCoeff[5] * p*p*p*p*p +
            meanCoeff[6] * p*p*p*p*p*p;

 float width =  widthCoeff[0] +
             widthCoeff[1] * p +
             widthCoeff[2] * p*p +
             widthCoeff[3] * p*p*p +
             widthCoeff[4] * p*p*p*p +
             widthCoeff[5] * p*p*p*p*p +
             widthCoeff[6] * p*p*p*p*p*p;

 if ( mass2 > mean - 2*width && mass2 < mean + 2*width ) return 1;

 return 0;
}
```

8. Code examples
Below you can find two code examples, which use different approaches to analysis of the data.

1) /eos/nica/mpd/users/riabovvg/ECAL_Tutorial_Examples/Pi0_analysis
This is a simplified example of pi0 reconstruction. This code works with the simulated DST files. In order to run the code please type: root -b -q Pi0Analysis.C'("DST.root")' For convenience, you can find an example of the DST file in the same directory, DST_1.root, which has 500 fully reconstructed BiBi@9.2 events simulated with UrQMD v.3.4.

The code will process all events in the input file and produce a number of histograms in the output Pi0Analysis.root file:

hRealAll - invariant mass distribution of gamma-pairs with basic selection cuts
hRealAll_Pi0 - same for true photon pairs from pi0 decays
hMixedAll - corresponding unscaled mixed-event background
hRealPID - invariant mass distribution of gamma-pairs with basic selection cuts + charged veto cut + lambda-PID cut
hRealPID_Pi0 - same for true photon pairs from pi0 decays
hRealCHI - invariant mass distribution of gamma-pairs with basic selection cuts + charged veto cut + Chi2-PID cut
hRealCHI_Pi0 - same for true photon pairs from pi0 decays

Please also see comments in the code.

2) /eos/nica/mpd/users/riabovvg/ECAL_Tutorial_Examples/Photon_analysis
This is a simplified analysis of single photons and tracks matched to the ECAL. The code works with the simulated DST files. In order to run the code please type: root -b –q macro_run.C’(1)’. For convenience, you can find an example of the DST file in the same directory, DST_1.root, which has 500 fully reconstructed BiBi@9.2 events simulated with UrQMD v.3.4.

The code will process all events in the input file and produce a number of histograms and a root Tree in the output TreeHisto_1.root file:

Histograms:
hE - energy spectrum of all clusters
hE_unmatch - energy spectrum of all clusters without rough matching to mpd tracks
hE_match - energy spectrum of all clusters with rough matching to mpd tracks
hE_phot - energy spectrum of all clusters with significant contribution from photons (>50%)
hE_pi - energy spectrum of all clusters with significant contribution from pions (>50%)
hE_k - energy spectrum of all clusters with significant contribution from kaons (>50%)
hE_p - energy spectrum of all clusters with significant contribution from protons (>50%)
hRZ - R vs. Zed for all clusters
hoccup - ECAL occupancy in phi:zed
hdPhi - dPhi cluster matching to mpd tracks vs. energy
hdZed - dZed cluster matching to mpd tracks vs. energy
hdPhiRes - ECAL dPhi resolution for photons vs. E
hdThetaRes - ECAL dTheta resolution for photons vs. E
hdZedRes - ECAL dZed resolution for photons vs. E
hEnRes - ECAL energy resolution for photons vs. E
hdPhi_mpd - dPhi mpdtrack matching to ECAL clusters vs. pT
hdZed_mpd - dZed mpdtrack matching to ECAL clusters vs. pT
hEP_el - E/p ratio for identified electrons vs. pT
hEP_el_true - E/p ratio for true identified electrons vs. pT

Tree variables:
b_mc - impact parameter
x_vertex_mc - true simulated X-vertex
y_vertex_mc - true simulated Y-vertex
z_vertex_mc - true simulated Z-vertex
x_vertex_mpd - reconstructed X-vertex
y_vertex_mpd - reconstructed Y-vertex

z_vertex_mpd - reconstructed Z-vertex
nclusters - number of reconstructed ECAL clusters
HitCluster[nclusters] - umber of towers in a cluster
ECluster[nclusters] - energy of a cluster (calculated as a sum of all tower energies)
ChiCluster[nclusters] - chi2/NDF for shower-shape analysis
TCluster[nclusters] - simulated time-of-flight (smear by 500 ps for realistic time resolution)
XCluster[nclusters] - X of a cluster
YCluster[nclusters] - Y of a cluster
ZCluster[nclusters] - Z of a cluster
MatchdPhiCluster[nclusters] - dphi to the closest mpdtrack (cm)
MatchdZedCluster[nclusters] - dzed to the closest mpdtrack (cm)
MatchIndCluster[nclusters] - index of the closest mpdtrack
MCTrECluster[nclusters][5] - energies of top five MC contributors to a cluster (sorted by energy)
MCTrIndCluster[nclusters][5] - MC track indexes of top fiv MC contributors to a cluster

Please also see comments in the code.
Please also note that particle identification with mpdpid class used in the code might be obsolete.
The tree has all the variables to reproduce analysis of code example (1). In some cases, production of smaller Trees speeds up the analysis.