EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

ALICE

ALICE-INT-2012-xxx
October 4, 2012

# EMCal documentation : code description, simulation and reconstruction strategy ...

EMCal collaboration

Email:alice-emcal-offline@cern.ch

## Abstract

In this document we want to describe the EMCal related code, how works the calorimeter and how we plan to take data and control it.

# Contents

# 1 Introduction

This document is addressed to those who want to work with EMCal software and the different tasks needed to have the data taken ready to be analyzed. It is divided in 2 blocks: a first one with the description of the procedures needed cook the data and a second one with the reconstruction and simulation offline code.

For a fast introduction on the code and how it works you can have a look to the EMCal for beginners guide [1]. Some other interesting references are the AliRoot primer[3] , the offline AliRoot page [2], and the installation page from Dario Berzano [5]

## 1.1 Mechanical description - Federico

## 1.2 How EMCal works - Terry

EMCAL basic units are cells/towers (Pb-scintillator sandwich of about 70 layers). We have 12 SuperModules (4 in 2010, 10 in 2011-2012) composed of 24 (phi direction) x 48 (eta direction) cells (except last 2 SuperModules made of 8 cells in phi direction). Particles traversing the calorimeter, in particular photons and electrons, will deposit energy in different towers. The EMCAL reconstruction measures such energy per tower, forms clusters of cells produced by a given particle, and if possible matches them with particles detected by the tracking detectors in front of EMCAL (charged particles).

## 2 Geometry code - Marco +++

EMCAL Geometry, description and methods This page is intended for a description of the geometry, how it works, how can we access to it and its methods. Very preliminary version, to be worked.

### 2.1 Detailed classes description

The EMCAL geometry is implemented in several classes, here I present a (right now very brief, it should be completed) description:

- AliEMCALGeoUtils: Steering geometry class. No dependencies on STEER or EMCAL non geometry classes. Can be called during the analysis not loading all aliroot classes.
- AliEMCALGeometry: Derives from AliEMCALGeoUtils, contains dependencies on other EMCAL classes (AliEMCALRecPoint)
- AliEMCALEMCGeometry: Does the geometry initialization. Does all the definitions of the geometry (towers composition, size, Super Modules number ...)
- AliEMCALGeoParams: Class container of some of the geometry parameters so that it can be accessed everywhere in the EMCAL code, to avoid "magic numbers". Its use has to be propagated to all the code.
- AliEMCALShishKebabTrd1Module: Here the modules are defined and the position of the modules in the local super module reference system is calculated

### 2.2 How to get the geometry

You can get the geometry pointer in the following ways:

If galice.root is available:

```
AliRunLoader *rl = AliRunLoader::Open("galice.root",AliConfig::GetDefaultEventFolderName(),"read");

rl->LoadgAlice();//Needed to get geometry

AliEMCALLoader *emcalLoader = dynamic_cast<AliEMCALLoader*>(rl->GetDetectorLoader("EMCAL"));

AliRun * alirun = rl->GetAliRun();

AliEMCAL * emcal = (AliEMCAL*)alirun->GetDetector("EMCAL"); AliEMCALGeometry *
geom = emcal->GetGeometry();
```

else, if galice.root is not available:

AliEMCALGeometry * geom = AliEMCALGeometry::GetInstance("EMCAL_COMPLETE") ;

In this case you might need the file geometry.root if you want to access to certain methods that require local to global position transformations. This file can be generated doing a simple simulation, it just contains the transformation matrix to go from global to local.

The way to load this file is:

TGeoManager::Import("geometry.root");

The transformation matrices are also stored in the ESDs so if you do not load this file, you can have to load these matrices from the ESDs.

If you want to see different parameters used in the geometry printed (cells centers, distance to IP, etc), you have just to execute the method PrintGeometry().

### 2.3 Geometry configuration options

Right now we have the following geometry options:

- EMCAL_COMPLETE: 12 Super Modules (2 half Super Modules)
- EMCAL_FIRSTYEAR: 4 Super Modules (year 2010)
- EMCAL_FIRSTYEARV1: 4 Super Modules, corrected geometry (year 2010)
- EMCAL_COMPLETEV1: 10 Super Modules, corrected geometry (year 2011)
- EMCAL_COMPLETE12SMV1: 12 Super Modules (10+2/3), corrected geometry (year 2012)

There are other options but NOT TO BE USED, at some point they have to be removed:

- EMCAL_PDC06: Old geometry, for reading old data (which might not exist).
- EMCAL_WSU: Prototype geometry.

By default geometry is loaded with the EMCAL_COMPLETE12SMV1 configuration.

### 2.4 Mapping

The tower row/column mapping online and offline follows the alice numbering convention, here you will see a few pictures displaying the position of the super modules from different points of view and the position of the tower index in them

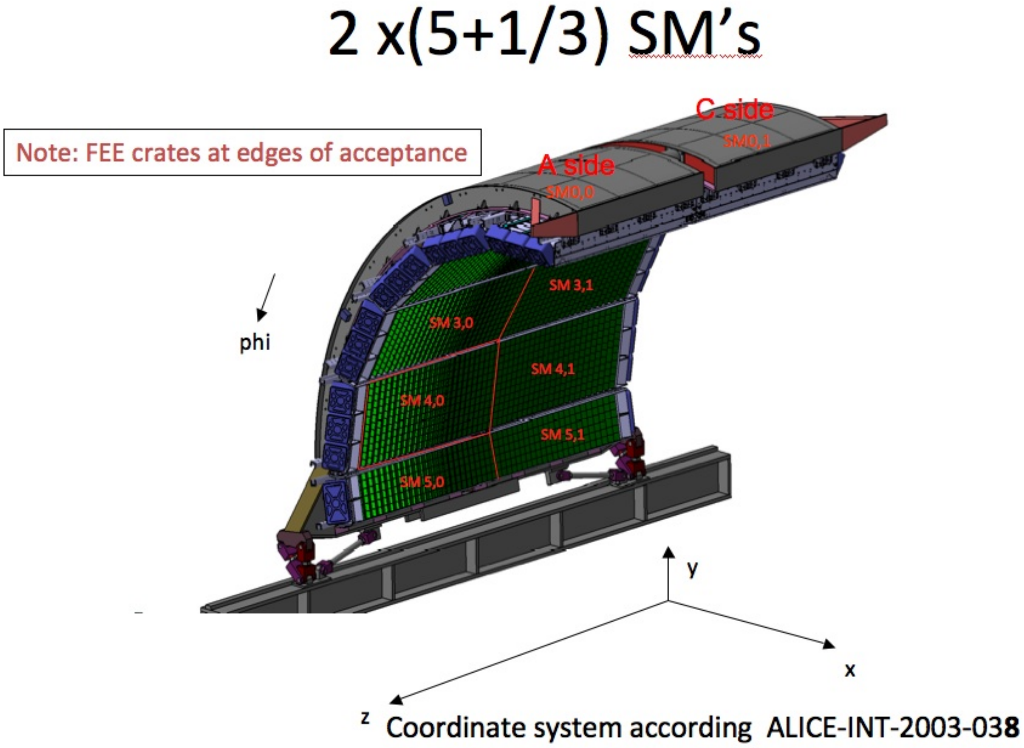

**Fig. 1:**

## 2.5 Tower index transformation methods

### *2.5.1 Absolute tower ID to Row/Column index*

Each EMCAL supermodule is composed of 24x48 towers (phi,eta), grouped in 4x4 modules. Each tower (even each module) has a unique number assigned, called in the code "absolute ID" number (absId). This number can be transformed into a row (phi direction) or column (eta direction) index. Here I list how can we go from the absId to the (row, col) formulation or viceversa:

From absId to col-row: Int_t nSupMod, nModule, nIphi, nIeta, iphi, ieta;

//Check if this absId exists

if(!CheckAbsCellId(absId)) return kFALSE;

// Get from the absId the super module number, the module number and the eta-phi index (0 or 1) in the module

GetCellIndex(absId, nSupMod, nModule, nIphi, nIeta);

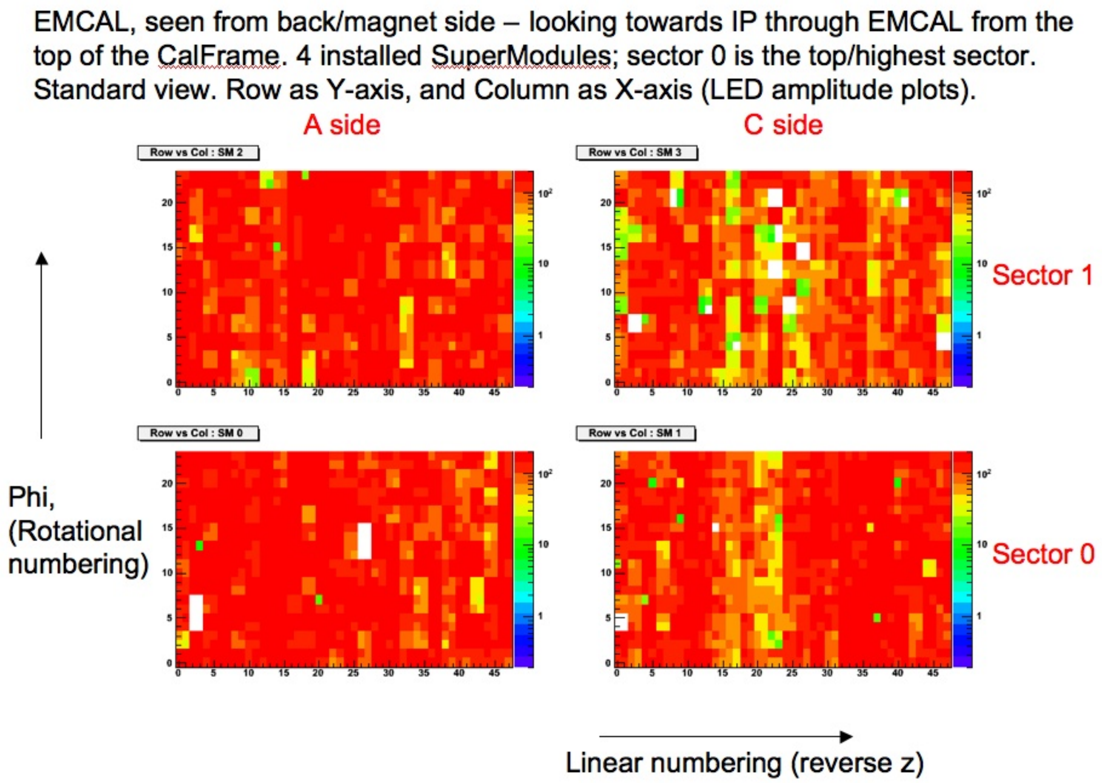
EMCAL, seen from back/magnet side – looking towards IP through EMCAL from the top of the CalFrame. 4 installed SuperModules; sector 0 is the top/highest sector. Standard view. Row as Y-axis, and Column as X-axis (LED amplitude plots).

**Fig. 2:**

// Get from the the super module number, the module number and the eta-phi index (0 or 1) in the module the tower row (iphi) and column (ieta) GetCellPhiEtaIndexInSModule(nSupMod,nModule,nIphi,nIeta, iphi, ieta);

From col-row to absId, following the same notation as above:

absid = GetAbsCellIdFromCellIndexes(nSupMode, iphi, ieta);

or

absid = GetAbsCellId(nSupMod, nModule, nIphi, nIeta);

Other interesting method is

Int_t GetSuperModuleNumber(Int_t absId)

## 2.6 Tower index to local / global reference system position

### *2.6.1 Local coordinates*

To correlate the tower index and its position in local coordinates we have the following methods:

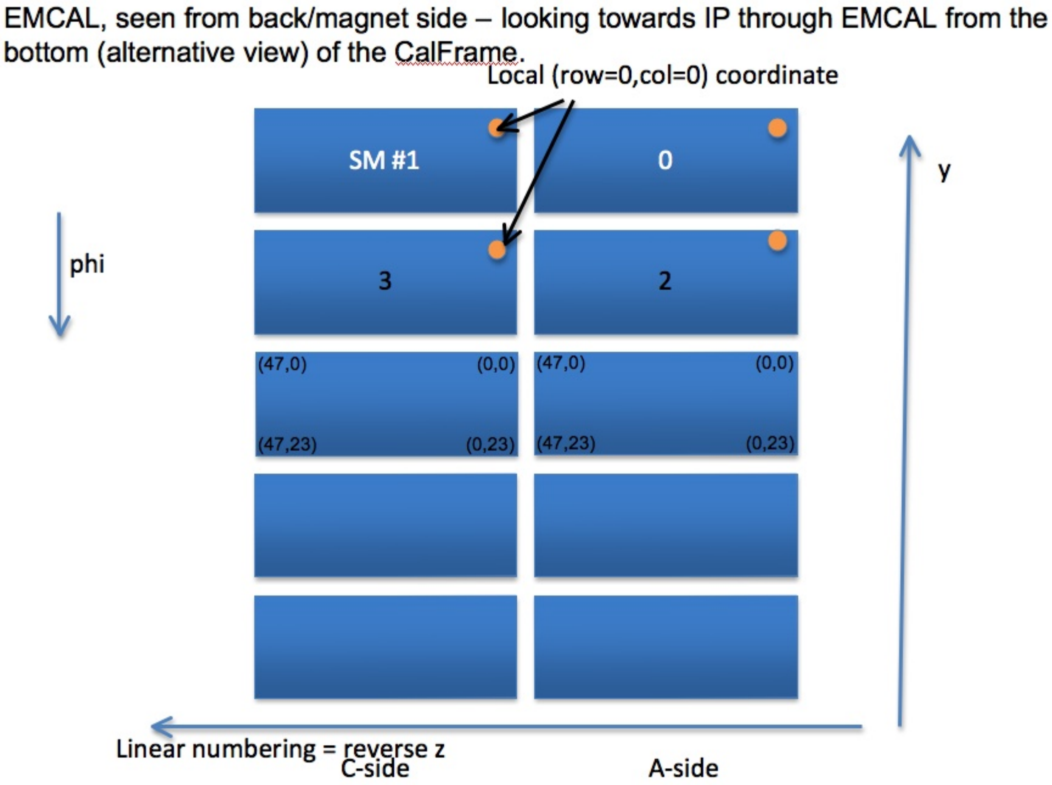

**Fig. 3:**

Bool_t AliEMCALGeoUtils::RelPosCellInSModule(Int_t absId, Double_t &xr, Double_t &yr, Double_t &zr) const;

Bool_t AliEMCALGeoUtils::RelPosCellInSModule(Int_t absId, Double_t loc[3]) const;

Bool_t AliEMCALGeoUtils::RelPosCellInSModule(Int_t absId, TVector3 &vloc) const;

which input is the absId and the output are the coordinates of the center of towers in the local coordinates of the Super Module. What it does inside is to get from the absId the column and row index of the cell, independently of the Super Module (like above), and it gets the center of the cell from 3 arrays (x,y,z) filled with such quantities. How and where are calculated such central positions? The arrays are filled during the initialization of the geometry in method

AliEMCALGeoUtils::CreateListOfTrd1Modules()

«««<Someone else should explain how it works»»>

In case we calculate the cluster position, things are a bit different.

«««< This explanation should go to the clusterization section»»

This is done in

void AliEMCALRecPoint::EvalLocalPosition()

First we calculate the cell position with the method

AliEMCALGeometry::RelPosCellInSModule(Int_t absId, Int_t maxAbsId, Double_t tmax, Double_t &xr, Double_t &yr, Double_t &zr)

The calculation of the cell position done here is different in the "x-z" but the same in "y".

««<Someone else should explain how it works»»>

In this particular case the position calculation per tower depends on the position of the maximum cell, and the sum of the energy of the cells of the cluster. The maximum depth (tmax) is calculated with the method

Double_t AliEMCALRecPoint::TmaxInCm(const Double_t e)

//e: energy sum of cells

static Double_t ca = 4.82;// shower max parameter - first guess; ca=TMath::Log(1000./8.07)

static Double_t x0 = 1.23; // radiation lenght (cm)

static Double_t tmax = 0.; // position of electromagnetic shower max in cm

tmax = TMath::Log(e) + ca+0.5;

tmax *= x0; // convert to cm

After the cells position of the cluster is get, the position of the cluster is calculated averaging the cell positions with a logarithmic weight:

w(cell i) = TMath::Max( 0., logWeight + TMath::Log( energy[cell i] / summed_cluster_cell_energy ));

where the logWeight was chosen to be 4.5 (this value was taken from PHOS, never optimized as far as I know)

So in the end the position, is

f = Sum(f(i) * w(i))/Sum(w(i))

where f=x,y,z.

### *2.6.2 Global coordinates*

To transform from local to global we have the methods

void GetGlobal(const Double_t *loc, Double_t *glob, int ind) const;

void GetGlobal(const TVector3 &vloc, TVector3 &vglob, int ind) const;

void GetGlobal(Int_t absId, Double_t glob[3]) const;

void GetGlobal(Int_t absId, TVector3 &vglob) const;

These methods take the local coordinates and transform them into global coordinates using the transformation matrix of the Super Module.

TGeoHMatrix* m = GetMatrixForSuperModule(nSupMod);

if(m) m->LocalToMaster(loc, glob);

GetGlobal is called in the following useful methods in the geometry class:

Return the eta and phi angular position of the cell from the AbsId void EtaPhiFromIndex(Int_t absId, Double_t &eta, Double_t &phi) const; void EtaPhiFromIndex(Int_t absId, Float_t &eta, Float_t &phi) const; Print information of the cells. For "pri>0" returns more information. "tit" has not much use, this value is printed. void PrintCellIndexes(Int_t absId, int pri, const char *tit)

### 2.7 Geometry Alignment

AliRoot contains a frame for the correction of the misplacement of geometry objects with respect to the ideal positions. You can have a look in STEER to classes:

AliAlignObj AliAlignObjMatrix AliAlignObjParams AliAlignmentTracks

In EMCAL, we have the class AliEMCALSurvey that creates the corrections to the alignable objects. The class AliEMCALSurvey was established to take the survey parameters from OCDB, calculate the shift in position of the center of the end faces of the supermodules from the nominal position, and convert this to a transformation matrix for each supermodule which is applied to correct the global position of the supermodules. All calculations of global positions would then use these corrected supermodule positions to determine their locations within the ALICE global coordinate system.

## 3 EMCal OCDB/OADB - Marcel

OCDB = Offline data base, OADB = Offline Analysis data base. The offline condition data base, OCDB, contains the different parameters used for simulation or reconstruction of the detectors or even the LHC machine parameters that might change for the different run conditions.

The EMCAL OCDB (and other detectors OCDB) is divided in 3 directories that can be found in

$ALICE_ROOT/OCDB/EMCAL

- Calib: Very different type of information, from hardware mapping to calibration parameters.
- Align: Survey misplacements in geometry.
- Config: Detector configuration: Temperatures

Inside these directories you will find other subdirectories with more specific type of parameters. Each of the directories contains a file named in this way:

Run(FirstRun)_(LastRun)_v(version)_s(version).root

being the default and what you will find in the trunk:

Run0_999999999_v0_s0.root

What is actually used for the real data reconstruction can be found in alien here:

/alice/data/20XX/OCDB/EMCAL

There are different repositories for different years (20XX). For the simulation productions, there is another repository on the grid:

/alice/simulation/2008/v4-15-Release/XXX/EMCAL

but there you will find 3 options, XXX=Ideal, Full and Residual. Each one is meant to reproduce the detector with different precision. For EMCAL, right now these 3 repositories contain the same parameters.

In the next lines, what is stored, how to read it and how to fill it, is explained. How to create explained in strategy section.

## 3.1 How to use a different OCDB

In your simulation/reconstruction macro you have to specify a default OCDB, if it is different from $ALICE_ROOT/OCDB. When running on the grid you are forced to do it, so you have to set for example in a reconstruction of simulated data:

```
reco.SetDefaultStorage("alien://Folder=/alice/simulation/2008/v4-15-Release/Residual/");
```

If you have modified one of the OCDB files or several which are not in the default storage OCDB, you have to put in the simulation or reconstruction macro:

```
reco.SetSpecificStorage("EMCAL/Calib/Pedestals","local://your/modified/local/OCDB");
```

In this example, you will have to put in /your/modified/local/OCDB/EMCAL/Calib/Data the modified file with the calibration coefficients.

If you modify more of the OCDB files, you can do it like this:

```
reco.SetSpecificStorage("EMCAL/Calib/","local:/your/modified/local/OCDB");
```

in this example you will have to put in /your/modified/local/OCDB/EMCAL/Calib/ all the directories inside EMCAL/Calib with its corresponding files.

## 3.2 Energy calibration: EMCAL/Calib/Data

Calibration Coefficients tower by towers are stored there. What is stored is an object of the class AliEMCALCalibData which is a container of gains and pedestals per tower. These coefficients are used in:

Simulation: during the digitization, in AliEMCALDigitizer::Digitizer(), when calling AliEMCALDigitizer::DigitizeEnergy(), to transform the deposited energy into ADC counts. Reconstruction: in AliEMCALClusterizerv1::Calibrate() called in AliEMCALClusterizer::MakeClusters(), when forming the cluster, to get the final cluster energy. The macro $ALICE_ROOT/EMCAL/macros/CalibrationDB/Ali is an example on how to set the calibration coefficients per channel, or how to read them from the OCDB file. This macro can set all channels with the same selected value or with random values given a uniform or gaussian smearing of a selected input value. A simple example that shows how to print the parameters is PrintEMCALCalibData.C

All channels in simulation have the same value for the gains and pedestals, gains are 0.0153 GeV/ADC counts and pedestal are set to 0 since the calorimeter works with Zero Suppressed data.

### 3.3 Bad channels: EMCAL/Calib/Pedestals

Storage for the bad channels map found in hardware. What is stored is an object of the class AliCaloCalibPedestal, class used for monitoring the towers calibration and functionality. This class has the data member TObjArray *fDeadMap which consists of an array of 12 TH2I (as many as Super Modules), and each TH2I has the dimension of 24x48 (number of towers in phi x eta direction), each bin corresponds to a tower. The content of each entry in the histogram is an integer which represents the possible status:

enum kDeadMapEntrykAlive = 0, kDead, kHot, kWarning, kResurrected, kRecentlyDeceased, kNumDeadMapStates;

Right now only the status kAlive, kDead, kHot and soon kWarning (soon, not yet) are set but, the code is basically skipping all the channels that are kDead and kHot. The bad channel map is used in the reconstruction code in 3 places:

- AliEMCALRawUtils::Raw2Digits() : Before the raw data time sample is fitted, the status of the tower is checked, and if bad (kHot or kDead), the fit is not done. This avoids trying to fit ill shaped samples. This step is optional though, right now default is to skip the bad channels here. With the RecParam OCDB we can select to use it or not.
- AliEMCALClusterizerv1::Calibrate(): once the cluster is formed, to get the cluster energy from its cells.
- AliEMCALRecPoint::EvalDistanceToBadChannels(): Evaluate the distance of a cluster to the closest bad channel. During the analysis we may want to skip clusters close to a bad channel. This time a bad channel is whatever is not kAlive.

The macro $ALICE_ROOT/EMCAL/macros/PedestalDB/AliEMCALPedestalCDB.C, is an example on how to set the bad channel map and how to read it from a file. When executed it displays a menu that allows to set randomly as bad a given % of the towers, also it allows to set the map from an input txt file, with the format like $ALICE_ROOT/EMCAL/macros/PedestalDB/map.txt, (this map file is the one used in the last mapping in the raw OCDB). It also can read the OCDB file and display the 12 TH2I histograms on screen.

### 3.4 Reconstruction parameters: EMCAL/Calib/RecoParam

Storage for the parameters used in reconstruction. What is stored is an object of the class AliEMCALRecParam which is a container for all the parameters used. There are different kind of parameters, we can distinguish them depending on which step of the reconstruction are used:

#### Raw data fitting and mapping

- Double_t fHighLowGainFactor; // gain factor to convert between high and low gain
- Int_t fOrderParameter; // order parameter for raw signal fit

- Double_t fTau; // decay constant for raw signal fit
- Int_t fNoiseThreshold; // threshold to consider signal or noise
- Int_t fNPedSamples; // number of time samples to use in pedestal calculation
- Bool_t fRemoveBadChannels; // select if bad channels are removed before fitting
- Int_t fFittingAlgorithm; // select the fitting algorithm
- static TObjArray* fgkMaps; // ALTRO mappings for RCU0..RCUX

**Clusterization**

- Float_t fClusteringThreshold ; // Minimum energy to seed a EC digit in a cluster
- Float_t fW0 ; // Logarithmic weight for the cluster center of gravity calculation
- Float_t fMinECut; // Minimum energy for a digit to be a member of a cluster
- Bool_t fUnfold; // Flag to perform cluster unfolding
- Float_t fLocMaxCut; // Minimum energy difference to consider local maxima in a cluster
- Float_t fTimeCut ; // Maximum difference time of digits in EMC cluster
- Float_t fTimeMin ; // Minimum time of digits
- Float_t fTimeMax ; // Maximum time of digits

**Track Matching**

- Double_t fTrkCutX; // X-difference cut for track matching
- Double_t fTrkCutY; // Y-difference cut for track matching
- Double_t fTrkCutZ; // Z-difference cut for track matching
- Double_t fTrkCutR; // cut on allowed track-cluster distance
- Double_t fTrkCutAlphaMin; // cut on 'alpha' parameter for track matching (min)
- Double_t fTrkCutAlphaMax; // cut on 'alpha' parameter for track matching (min)
- Double_t fTrkCutAngle; // cut on relative angle between different track points for track matching
- Double_t fTrkCutNITS; // Number of ITS hits for track matching
- Double_t fTrkCutNTPC; // Number of TPC hits for track matching

**PID**

- Double_t fGamma[6][6]; // Parameter to Compute PID for photons
- Double_t fGamma1to10[6][6]; // Parameter to Compute PID not used
- Double_t fHadron[6][6]; // Parameter to Compute PID for hadrons
- Double_t fHadron1to10[6][6]; // Parameter to Compute PID for hadrons between 1 and 10 GeV
- Double_t fHadronEnergyProb[6]; // Parameter to Compute PID for energy ponderation for hadrons
- Double_t fPiZeroEnergyProb[6]; // Parameter to Compute PID for energy ponderation for Pi0
- Double_t fGammaEnergyProb[6]; // Parameter to Compute PID for energy ponderation for gamma
- Double_t fPiZero[6][6]; // Parameter to Compute PID for pi0

The macro $ALICE_ROOT/EMCAL/macros/RecParamDB/AliEMCALSetRecParamCDB.C, is an example on how to set the parameters. There are different event types that we might record, and each event type might require different reconstruction parameters. The event types that are now defined in STEER/AliRecoParam.h are:

enum EventSpecie_t kDefault = 1, kLowMult = 2, kHighMult = 4, kCosmic = 8, kCalib = 16;

The default event species that we have is kLowMult (low multiplicity). For AliRoot versions smaller than release 4.17 it was set to be kHighMult (high multiplicity). Right now in what EMCAL concerns, kDefault=kLowMult=kCosmic=kCalib. kHighMult differs only from the rest in 2 clusterization parameters, for low multiplicity they are fMinECut=10 MeV and fClusteringThreshold=100 MeV and for high multiplicity they are fMinECut=0.45 GeV and fClusteringThreshold=0.5 GeV.

A simple example that shows how to print the parameters for the different event species is PrintEMCALRecParam.C

### 3.5 Simulation parameters: EMCAL/Calib/SimParam

Storage for the parameters used in simulation. What is stored is an object of the class AliEMCALSimParam which is a container of all the parameters used. There are different kind of parameters, we can distinguish them depending on which step of the simulation they are used:

### SDigitization

- Float_t fA ; // Pedestal parameter
- Float_t fB ; // Slope Digitizition parameters
- Float_t fECPrimThreshold ; // To store primary if Shower Energy loss > threshold

### Digitization

- Int_t fDigitThreshold ; // Threshold for storing digits in EMC = 3 ADC counts
- Int_t fMeanPhotonElectron ; // number of photon electrons per GeV deposited energy = 4400 MeV/photon
- Float_t fPinNoise ; // Electronics noise in EMC = 12 MeV
- Double_t fTimeResolution ; // Time resolution of FEE electronics = 600 ns
- Int_t fNADCEC ; // number of channels in EC section ADC =

The macro $ALICE_ROOT/EMCAL/macros/SimParamDB/AliEMCALSetSimParamCDB.C, is an example on how to set the parameters. A simple example that shows how to print the parameters is PrintEMCALSimParam.C

## 3.6 Alignment

# 4 Simulation code

The class AliSimulation manages this part. Have a look to the macro "$ALICE_ROOT/EMCAL/ macros/TestEMCALSimulation.C". The simulation consists of different steps: geometry and event definition, particle generation, transport of the particle in the material (GEANT) and finally digitization. Note that the final output from the digitization process is not the same as the experimental Raw Data. The process of converting the digitized data to Raw Data is discussed in Sec. 1.4. In Sec. 1.5, I give the recipe to do all the steps.

## 4.1 Event Generation and particle transport: Hits

Once the generator is executed, the generated particles are transported in the detector material with the Monte Carlo code, GEANT3 by default. Other options are GEANT4 or FLUKA (some license problems with FLUKA right now so not in use?). All the generated particles are kept in a file called **Kinematics.root**. After the particle transport is executed, the objects **Hits** are created. They contain the energy deposited in the sensitive material of the detector by the generated particle, their position, impact time (after collision) and the identity of the original particle. Hits are stored in a file called **DETECTOR.Hits.root**, in the calorimeter case: **EMCAL.Hits.root.**

## 4.2 Digitization: SDigits and Digits - Evi

We want to generate events which look like the real data collected by the experiment. In the end, we want to have an amplitude in ADC counts and a time (when particle traverse a cell) per each cell (tower) of the calorimeter. In the code for calorimeters, it is done in the following steps: 1st) **SDigit** objects are created, they consist of the sum of deposited energy by all Hits in a cell (a particle can create Hits in different cells but only one in a single cell), so there is only one SDigit per fired cell; 2nd) **Digit** objects are created, they are like the SDigits but the energy in the cell is transformed into the ADC amplitude units, the electronic noise is added and Digits whose energy does not pass an energy threshold (3 ADC counts) are eliminated. SDigits and Digits are stored in the files **EMCAL.SDigits.root** and **EMCAL.Digits.root**, respectively.

## 4.3 Raw data - David

What we will get directly from the experiment are not Digits but a time samples of ADC counts per each cell. These samples are called **Raw Data**. The samples have a shape, more complicated than a Gaussian distribution, which is fitted offline. With real data, Digits amplitude is just the maximum of the distribution obtained with the fit to the sample. The Digit time (defined by a time the particle hit the active volume of the detector) is the time bin when the signal begins to rise. There is a method to pass from Digits to Raw and vice versa in the class AliEMCALRawUtils: Raw2Digits and Digits2Raw, respectively. For the reconstruction step we need the Digits. The generation of Raw Data is optional during simulations, we can reconstruct data generating directly Digits, but Raw data will be the initial step when reconstructing real data.

## 4.4 How to make a simulation

TestEMCALSimulation.C is a very simple macro where we specify all the simulation parameters and execute the simulation, here I put a similar but a bit more elaborated macro:

```
void TestEMCALSimulation() {
TString detector="EMCAL TPC"; // Define in this variable the detectors
//that you want to be included in the simulation for the digitization.
//They can be less detectors than the detectors defined in the Config.C
//file, imagine that you want all the detectors in front of EMCal present
//to consider the conversion of particles but you are not really
//interested in the output from these detectors.  Option detector="ALL"
//makes all detectors.
AliSimulation sim ; //Create simulation object
// Generation and simulation
sim.SetRunGeneration(kTRUE) ; //Default value is kTRUE, make generation
//For some reason we may want to redo the Digitization, without redoing
//the generation, in this case it must set to kFALSE
// Making SDigits
sim.SetMakeSDigits(detector) ; //We want to make SDigits
// set no detectors if SDigits are already made
// Making Digits
sim.SetMakeDigits(detector) ; //We want to make Digits
// set no detectors if SDigits are already made
//Merging
//sim.MergeWith("bgrd/galice.root") ; //If we want to merge a signal
//and a background, the merging is done at the SDigit level.  The
//background must be located in the repertory defined in the method.
//Write Raw Data, make Raw data from digits
//sim.SetWriteRawData(detector) ;
//sim.SetConfigFile("somewhere/ConfigXXX.C");//Default is Config.C
```

```
//Make the simulation
sim.Run(3) ; // Run the simulation and make 3 events
}
```

# 5 Reconstruction code

The energy deposited by the particles in the towers produces scintillating light that is propagated with optic fibers through the different layers to APD placed at the base of the cells. The APDs amplify the signal and generate an electronic pulse shape that is stored in the raw data format. From this pulse shape, we extract the signal amplitude and the arrival time. The pulse shape is fitted during the reconstruction via a parametrized function and TMinuit, and these 2 values are extracted.

A particle produces signals in different towers (electromagnetic shower expands more than its Molière radius which is a cell size), the next step is the formation of clusters of cells that belong to the same particle, although depending on the energy, granularity, clusterization algorithm or event type, those clusters might have contributions from different particles. The default algorithm in pp collisions is a simple aggregation of neighboring cells until there is no more cells above a certain energy threshold (named clusterizer V1). In case of Pb-Pb collisions environment, where particle showers merge quite often, we apply another algorithm that aggregates cells to the clusters until reaching a cell with more energy than the precedent (named clusterizer V2). Depending on the analysis type you might want to use one or the other clusterization type, that is why the re-clusterization is also possible at the analysis level. A last clusterizer is implemented, which makes 3x3 clusters. It has been used in jet analysis for instance in order to avoid biasing jet reconstruction where one is interested in the energy flow over a large area without explicit reconstruction of photon showers and the driving consideration is that the clusterizer not interfere with the jet finder (whereas for pi0, eta, and direct photon analyses, v2 is most likely preferable).

Once the cluster is defined, we calculate cluster parameters, shower shape parameters, that will help at the analysis level to identify each cluster as one particle type. Also, we compare the cluster position information with the propagation of tracks measured in the central barrel to the EMCAL surface, to identify the clusters generated by charged particles.

The final analysis objects, ESDs and AODs, contain all the cluster and cell basic informations allowing to redo the clusterization if needed at the analysis level.

## 5.1 Offline data base access

How to create explained OCDB/OADB section.

### *5.1.1 Energy calibration*

### *5.1.2 Bad channels - Marie, Alexis*

### *5.1.3 Alignment - Marco*

## 5.2 Raw data fitting: from ADC sample to digits - David

AliEMCALRawUtils, AliCaloRawAnalyzer*, AliCalo*, AliEMCALDigit.

### 5.3 Clusterization: From digits to clusters - Constantin, Adam

AliEMCALClusterizer*, AliEMCALRecPoint

### 5.4 Cluster-Track matching - Rongrong, Shingo, Michael

Propagation of TPC tracks to EMCAL and selection of clusters as belonging to a track or not.

### 5.5 How to execute the reconstruction

The way is very similar as in the simulation case, the macro TestEMCALReconstruction.C (a bit more detailed than the one in $ALICE_ROOT/EMCAL/macros) is as follows:

```
void TestEMCALReconstruction() {

TString detector="EMCAL TPC";//Same function as in Simulation.C

AliReconstruction rec; //Create reconstruction object

//Making Tracking

rec.SetRunTracking(detector) ;

//Particle Reconstruction.  Make Rec Points

rec.SetRunReconstruction(detector);

//read RAW data.  Give directory where raw data is stored

//rec.SetInput("RawDataDirectory/raw.root");

//Make vertex finder

rec.SetRunVertexFinder(kFALSE) ; // false only if the tracking detectors are
not included.

//Fill ESD file with RecPoints information.

rec.SetFillESD(detector) ;

//Run Reconstruction

rec.Run() ;

}
```

# 6 Reconstruction strategy

## 6.1 Calibration

Here we describe how are obtained different correction factors needed : energy calibration (MIP, pi0, run by run), time calibration and bad channel mask.

All these correction factors or masks are stored in the OCDB but also the OADB. Since these calibration parameters do not arrive before full ALICE data reconstructions of the first periods are done, the calibration is stored not only in the OCDB but also in the OADB so that the clusters can be corrected at the analysis level. For the moment we do not store the time calibration and run by run correction factors in OCDB just in OADB.

### *6.1.1 Energy calibration: MIP calibration before installation - Julien*

### *6.1.2 Energy calibration: $\pi^0$ - Catherine*

First, the calibration is done on cosmic measurements done before installing the SuperModules at P2, but the accuracy obtained using MIPs is not good enough. We rely already during the data taking on the measurement of the pi0 mass position per cell. For this we require of the order of 100-200 M events triggered by EMCAL (trigger threshold at 1.5-2 GeV). A few iterations on the data, obtaining in each iteration improved calibration coefficients, are needed to achieve a good accuracy (1-2%). Since the online calibration has a strong effect on the trigger efficiency, the voltage gains of the APDs are varied after each running period, to get a equalized trigger performance. Still there will be some towers that due to the fact that they are behind of a lot of material (TRD support structures), that will be difficult to calibrate, for those MIPs or J/Psi measurement could help, but we have not arrived to the point of being able to use them ALICE has a reconstruction strategy mainly driven by the central barrel detectors. Run by run a calibration pass (CPass) is done with only a restricted amount of the run statistics. This is insufficient for the calorimeters so that is why we do not participate actively on such passes, except for QA purposes. Since we do not enter in this strategy, we need to get the best calibration as soon as possible, for this reason special calibration runs are requested at the beginning of the running period, and as soon as the manpower is available, the calibration parameters are produced. For details on calibration strategy see this presentation on a special calibration session.

### *6.1.3 Energy calibration: Run by run temperature gain variations - Evi, David*

The SuperModules calibration depends on the Temperature dependence of the different towers gains. We observe that from one period to other, where the T changes, the pi0 peak positions also changes. There are 2 ways to correct for this effect : measure the mean T per run, and get the gain curves per tower a calculate the corresponding correction; use the calibration LED events to quantify the variation from one reference run. These 2 procedures have problems, poor or lack of knowledge of the gain curves of some towers or bad performance of the LED system in certain regions.

#### *6.1.4 Time calibration - Marie*

The time of the amplitude measure by a given cell is a good candidate to reject noisy towers, identify pile up events, or even identify heavy hadrons at low energy. The average time is around 650 ns. The aim of the time calibration is to move this mean value to 0, with as small spread as possible (negative values are unavoidable for the moment).

### 6.2 Alignment - Marco

CERN provides survey measurements of the position of different EMCAL Supermodules points at the beginning of the running period (and on request?). As soon this information is available, the ideal EMCAL positions used in the reconstruction by default, are corrected with special position matrices calculated from the measurements. Finally, once the data is reconstructed, the accuracy of the alignment is cross checked with track matching and pi0 mass measurements, since those values change depending on variations on the positions of the SuperModules.

### 6.3 Bad channel finding - Alexis

The analysis is done on the output of QA histograms:

check distribution over the cells of:

- average energy (criteria 1) and
- average number of hit per event (criteria 2) (average computed for E > Emin)
- Shape criteria : $\chi^2/ndf$ (criteria 3), A (criteria 4) and B (criteria 5) which are parameters from the fit of each cell amplitude (the fit function is $A * e^{-B*x}/x^2$ and the fit range is from Emin to Emax). we run each criteria once , at each step we exclude the marked cells (above nsigma from mean value) to compute the next distribution.

(For each criteria we have some parameters Emin (min energy) Emax, (max energy for the Energy distribution fit), and nsigma, nb of sigma we use for excluding the cell;)

The typical nsigma used is 4 or 5; The min energy considered is 0.1 GeV -0.3 GeV. And max energy for fit is depending on the data we are looking at.

We do not distinguish bad/warm automatically, this distinction is made "by a visual" check so it is at some point subjective.

The cells are then marked as bad or warm and passed through OCDB, in the reconstruction pass, the bad ones are excluded.

# 7 Trigger

## 7.1 L0 - Jiri

Documented in [10]. Add Summary or more info here.

## 7.2 L1 - Rachid

## 7.3 L0-L1 simulation - Rachid

## 7.4 HLT - Federico

Documented in [11]. Add Summary or more info here.

# 8 Analysis format and code

All the reconstructed particles of all the detectors will be kept in a file called **AliESDs.root**. The detectors must store there the most relevant information which will be used in the analysis. Together with the AliESDs.root file, another file is created with some reference tags of the simulated events, containing for example the number of events per run. This file is named **Run0.Event0_1.ESD.tag.root** (1 means that only 1 event was simulated).

In order to do the analysis with the data contained in the ESDs, you only need the file **AliESDs.root** in your local directories or a grid collection. It is not necessary that in your working directory you keep other files like galice.root or EMCAL.*.root or any other. Anyway, we may want to access to the primary particles generated during the simulation, in that case we must have also the **galice.root** and **Kinematics.root** file. Also, if you want to access to some information of the detector geometry, you need to keep the **geometry.root** file.

There are other data analysis container file created from the ESD, the AOD (Analysis Object Data) with smaller quantity of data for most of the subsystems but for the calorimeters, where we copy all the information[1].

## 8.1 Calorimeter information in ESDs/AODs

The basic calorimeter information needed for analysis is stored in the ESDs or AODs in the form of CaloClusters and CaloCells (cell = EMCal Tower or PHOS crystal). Also there is some information stored in the AOD/ESD event classes, it will be detailed more in the lines below. Both AOD and ESD classes derive from virtual classes so that with a similar analysis code and access methods, we can read both kind of data formats.

### *8.1.1 AliVEvent (AliESDEvent, AliAODEvent)*

Those are manager classes for the event information retrieval. Regarding the calorimeters they have the following access information (getter) methods (there are the equivalent setters just have a look to the header file of the class):

- AliVCaloCluster *GetCaloCluster(Int_t i) : Returns a CaloCluster listed in position "i" in the array of CaloClusters. It can be either PHOS or EMCal (PHOS list of clusters is before the EMCal list).

- TClonesArray *GetCaloClusters() : Returns the array with CaloClusters PHOS+EMCAL, Only defined for AODs

- Int_t GetEMCALClusters(TRefArray *clusters) ; Int_t GetPHOSClusters(TRefArray *clusters) : Returns an array with only EMCal clusters or only with PHOS clusters.

- Int_t GetNumberOfCaloClusters(): Returns the total number of clusters PHOS+EMCAL.

---

[1] until half 2012 everything but the time of the cells was not stored

- AliVCaloCells *GetEMCALCells(); AliESDCaloCells *GetPHOSCells() : Returns the pointer with the CaloCells object for EMCal or PHOS.
- AliVCaloTrigger *GetCaloTrigger(TString calo) : Access to trigger patch information, for calo="PHOS" or calo="EMCAL"
- const TGeoHMatrix* GetPHOSMatrix(Int_t i); const TGeoHMatrix* GetEMCALMatrix(Int_t i): Get the matrices for the transformation of global to local. The transformation matrices are not stored in the AODs.

### *8.1.2 AliVCaloCluster (AliESDCaloCluster,AliAODCaloCluster)*

They contain the information of the calorimeter clusters. Note that PHOS and EMCAL CaloClusters are kept in the same TClonesArray (see above). The information stored in each CaloCluster is :

- General
  - Int_t GetID(): It returns a unique identifier number for a CaloCluster.
  - Char_t GetClusterType():It returns kPHOSNeutral (kPHOSCharged exists but not used) or kEMCALClusterv1. Another way to get the origin of the cluster:
  - Bool_t IsEMCAL(); Bool_t IsPHOS().
  - void GetPosition(Float_t *pos) : It returns a x,y,z array with the global positions of the clusters in centimeters.
  - Double_t E() : It returns the energy of the cluster in GeV units.
  - void GetMomentum(TLorentzVector& p, Double_t * vertexPosition ): It fills a TLorentzVector pointing to the measured vertex of the collision. It also modifies the cluster global positions to have a vector pointing to the vertex, this has to be corrected. Assumes that cluster is neutral. To be used only for analysis with clusters not matched with tracks.
- Shower Shape
  - Double_t GetDispersion(): Dispersion of the shower.
  - Double_t Chi2(): Not filled.
  - Double_t GetM20() Double_t GetM02() : Ellipse axis.
  - UChar_t GetNExMax() : Number or maxima in cluster. Not filled.
  - Double_t *GetPID(): PID weights array, 10 entries corresponding to the ones defined in AliPID.h
  - enum EParticleType kElectron = 0, kMuon = 1, kPion = 2, kKaon = 3, kProton = 4, kPhoton = 5, kPi0 = 6, kNeutron =7, kKaon0 = 8, kEleCon = 9,kUnknown = 10; : PID tag numbers, corresponding to the PID array

  - Double_t GetDistanceToBadChannel() : Distance of the cluster to closest channel declared as kDead, kWarm or kHot.
  - Double_t GetTOF() : Measured Time of Flight of the cluster.

- Track-Cluster matching
  - TArrayI * GetTracksMatched(): List of indexes to the likely matched tracks. Tracks ordered in matching likeliness. If there is no match at all, by default it contains one entry with value -1. Only in ESDs.
  - Int_t GetTrackMatchedIndex(Int_t i): Index of track in position "i" in the list of indices stored in GetTracksMatched(). Only in ESDs
  - Int_t GetNTracksMatched() : Total number of likely matched tracks. Size of GetTracksMatched() array.
  - Double_t GetEmcCpvDistance() : PHOS method, not used anymore. Use instead those below.
  - Double_t GetTrackDx(void), Double_t GetTrackDz(void): Distance in x and z to closest track.
  - TObject * GetTrackMatched(Int_t i): References to the list of most likely matched tracks are stored in a TRefArray. This method retrives the one in position "i". Tracks are listed in order of likeliness. The TObject is a AliAODTrack. Only for AODs

- MonteCarlo labels:
  - TArrayI * GetLabels(): List of indexes to the MonteCarlo particles that contribute to the cluster. Labels ordered in energy contribution.
  - Int_t GetLabel(): Index of MonteCarlo particle that deposited more energy in the cluster. First entry of GetLabels() array.
  - Int_t GetLabelAt(UInt_t i): Index of MonteCarlo particle in position i of the array of MonteCarlo indices.
  - Int_t GetNLabels() : Total number of MonteCarlo particles that deposited energy. Size of GetLabels() array.

- Cluster cells
  - Int_t GetNCells() : It returns the number of cells that contribute to the cluster.
  - UShort_t *GetCellsAbsId(): It returns the array with absolute id number of the cells contributing to the cluster. Size of the array is given by GetNCells().
  - Double32_t *GetCellsAmplitudeFraction(): For cluster unfolding, it returns an array with the fraction the energy that a cell contributes to the cluster.
  - Int_t GetCellAbsId(Int_t i) : It returns the absolute Id number of a cell in the array between 0 and GetNCells()-1.
  - Double_t GetCellAmplitudeFraction(Int_t i) : It returns the amplitude fraction of a cell in the array between 0 and GetNCells()-1.

### 8.1.3 *AliVCaloCells (AliESDCaloCells, AliAODCaloCells)*

They contain an array with the amplitude or time of all the cells that fired in the calorimeter during the event. Notice that per event there will be a CaloCell object with EMCAL cells and another one with PHOS cells.

- Short_t GetNumberOfCells(): Returns number of cells with some energy.
- Bool_t IsEMCAL(); Bool_t IsPHOS(); Char_t GetType(): Methods to check the origin of the AliESDCaloCell object, kEMCALCell or kPHOSCell.
- Short_t GetCellNumber(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the absolute cell number (from 0 to NModules*NRows*NColumns - 1).
- Double_t GetCellAmplitude(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the measured amplitude of the cell in GeV units.
- Double_t GetCellTime(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the measured time of the cell in second units.
- Double_t GetAmplitude(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the amplitude of the cell in GeV units.
- Double_t GetTime(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the time of the cell in second units.
- Double_t GetCellMCLable(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the index of the most likely MC label.
- Double_t GetCellEFraction(Short_t cellNumber): Given absolute cell number of a cell (from 0 to NModules*NRows*NColumns - 1), it returns the fraction of embedded energy from MC to real data (only for embedding)
- Double_t GetMCLabel(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the index of the most likely MC label.
- Double_t GetEFraction(Short_t pos): Given the position in the array of cells (from 0 to GetNumberOfCells()-1), it returns the fraction of embedded energy from MC to real data (only for embbedding)
- Bool_t GetCell(Short_t pos, Short_t &cellNumber, Double_t &amplitude, Double_t &time, Short_t &mclabel, Double_t &efrac); : For a given position of the list of cells, it fills the amplitude, time, mc lable and fraction of energy.

#### *8.1.4 AliVCaloTrigger (AliESDCaloTrigger, AliAODCaloTrigger) - Rachid)*

### 8.2 Macros

You can find example macros to run on ESDs or AODs in

$ALICE_ROOT/EMCAL/macros/TestESD.C or TestAOD.C

All the ESDs information is filled via the AliEMCALReconstructor/AliPHOSReconstructor class, in the method FillESD(). The AODs are created via the analysis class

$ALICE_ROOT/ANALYSIS/AliAnalysisTaskESDfilter.cxx,.h

and as already mentioned, for the calorimeters it basically just copies all the information from ESD format to AOD format. In the lines below I will try to explain what is the information stored and how to retrieve it. The location of the classes that I am going to describe below is

$ALICE_ROOT/STEER

### 8.3 Example code

The analysis is done using the data stored in the ESD. The macro
**$ALICE_ROOT/EMCAL/macros/TestESD.C**
is an example of how to read the data for the calorimeters PHOS and EMCal (just replace where it says EMCAL by PHOS in the macro to obtain PHOS data). For these detectors we have to use the ESD class AliESDCaloCluster or AliESDCaloCells to retrieve all the calorimeters information. For the tracking detectors, the class is called AliESDtrack, but the way to use it is very similar (see “$ALICE_ROOT/STEER/AliESDtrack.* ” and “$ALICE_ROOT/STEER/AliESDCaloCluster* ” for more details). In AliESDCaloCluster we keep the following cluster information: energy, position, number of Digits that belong to the cluster, list of the cluster Digits indeces, shower dispersion, shower lateral axis and a few more parameters. In AliESDCaloCells we keep the following tower information: amplitude (GeV), time (seconds), absolute cell number.

The structure of the ESD testing macro (TestESD.C) is the following:

- Lines 0-29: This macro is prepared to be compiled so it has "`includes`" to all the Root and AliRoot classes used.
- Lines 30-36: This macro prints some information on screen, the kind of information is set here. We print by default clusters information and optionally, the cells information, the matches information, the cells in the clusters information or the MonteCarlo original particle kinematics.
- Lines 40-64: Here are the methods used to load AliESDs.root , geometry or kinematics files. Also loop on ESD event is here.
- Lines 65-66 Gets the measured vertex of the collision.
- Lines 69-78 Loops on all the CaloCell entries and prints the cell amplitude, absolute number and time.

- Lines 84- end: We access the EMCAL AliESDCaloCluster array and loop on it. We get the different information from the CaloCluster.
- Lines 111-130: Track Matching prints. Access to the matched track stored in AliESDtrack.
- Lines 133-159: Cells in cluster prints
- Lines 161 - end: Access the stack with the MC information and prints the parameters of the particle that generated the cluster.

## 8.4 Advanced utilities : Reconstruction/corrrections of cells, clusters during the analysis

### *8.4.1 AliEMCALRecoUtils*

### *8.4.2 Tender : AliEMCALTenderSupply*

# 9 Run by run QA, how to and code

## 9.1 Online - Francesco, Michael

DQM, etc

## 9.2 Offline - Marie

Analysis code, what we control, how

## 9.3 Event display

## 9.4 Logbook tips

# References

[1] EMCal for beginners, https://twiki.cern.ch/twiki/pub/ALICE/EMCalOffline/EMCalBeginners.pdf

[2] AliRoot: ALICE offline project, http://aliceinfo.cern.ch/Offline/

[3] AliRoot Documentation, http://aliceinfo.cern.ch/Offline/AliRoot/Manual.html

[4] AliRoot Installation, http://aliceinfo.cern.ch/Offline/AliRoot/Installation.html

[5] AliRoot Installation from Dario Berzano, http://newton.ph.unito.it/~berzano/w/doku.php?id=alice:compile-any&redirect=1

[6] AliEn web page, http://alien.cern.ch/twiki/bin/view/AliEn/Home

[7] AliRoot in SVN http://alisoft.cern.ch/viewvc/?root=AliRoot

[8] EMCAL documentation, http://aliceinfo.cern.ch/Offline/Detectors/EMCALOffline.html

[9] EMCal Offline twiki, https://twiki.cern.ch/twiki/pub/ALICE/EMCalOffline

[10] EMCAL L0 http://www.sciencedirect.com/science/article/pii/S0168900212007681#

[11] EMCAL HLT http://arxiv.org/abs/1209.3647