

AliRoot for beginners: a fast guide on how to generate and analyse EMCal data

Gustavo Conesa Balbastre*

03/03/2010

Contents

1	Simulation	3
1.1	Configuration file	3
1.2	Event Generation and particle transport: Hits	4
1.3	Digitization: SDigits and Digits	5
1.4	Raw data	5
1.5	How to make a simulation	6
1.6	Root compilation and execution	6
2	Reconstruction	9
2.1	Particle reconstruction	9
2.2	Cluster-Track matching	9
2.3	Event Summary Data (ESD)	9
2.4	How to make the reconstruction	10
3	Analysis	11

*For any query, criticism or improvement about this document, you can contact me at: Gustavo.Conesa.Balbastre@cern.ch

Introduction

This document is addressed to those who want to begin to work with AliRoot [1], and in particular with EMCal, and have never worked before in this environment. It shows very briefly how can we proceed to execute some basic things like the event generation, reconstruction and data analysis. For more details on how AliRoot works and how the experimental data can be analyzed, I strongly recommend to read the AliRoot primer [2] (the present document is not a substitute of the “AliRoot primer” but an introduction for beginners), and also the AliEn pages [3] or go directly to the code in [4]. For detailed information on the EMCAL code you can go to the EMCAL offline pages (work in progress) [5].

I will assume that you have already AliRoot and its environment properly installed in your machine, if not, look at [2] or to the offline installation page [6]. If you type “cd \$ALICE_ROOT”, you will find all the AliRoot code. In the directories “\$ALICE_ROOT/macros” and “\$ALICE_ROOT/EMCAL/macros”, you will find different examples of macros. All the EMCal code is in “\$ALICE_ROOT/EMCAL”, in case you want to study thoroughly how the simulation, reconstruction and analysis software code of the calorimeter work.

If we want to generate some data (realistic physics data or just single particle to study the performance of the ALICE detectors) or read real experimental data, and analyze it with AliRoot, we have to pass through three steps: **simulation**, **reconstruction** and **analysis**. This steps are explained in the next sections.

1 Simulation

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.

1.1 Configuration file

The detector geometry and type of events that we want to generate are defined in a file called **Config.C**. Examples of such file can be found in “\$ALICE_ROOT/macros/” or a simple one in “\$ALICE_ROOT/EMCAL/macros”. The structure of the file is the following:

- Lines 0-45: “include” of the classes which perform the particle generation, detector geometry definition, and the transport of the particles in the detectors.
- Lines 46-95: Calls to the Monte Carlo generation, including a particular seed value, the steering class AliRunLoader and the Geometrical modeler. It is not necessary to understand all, I just want to mention three things: 1) with “SetSeed(0)”, the seed taken for the event generation depends on the computer clock time; 2) the AliRunLoader opens the file **galice.root** which is necessary in the simulation and reconstruction; and 3) the call “`rl->SetNumberOfEventsPerFile(3);`” defines the number of events per file, change 3 for a larger number if you don’t want to have a large number of output files, with three events each, as a result of the simulation of large number of events. There is no need to touch the rest.
- Lines 95-130: Set different GEANT parameters needed for the particle transport. It is not necessary to touch.
- Lines 130-147: This part and the next one are the most important parts. Here, we define which event generator we want to use, for example PYTHIA, HIJING, HERWIG, etc.

Below, I will give a few more details and a simple example of how to use a event generator.

- Lines 148-End: here, we define which detectors will be included in the simulation and their geometry. First, you need to set the flags for the detectors that you want to include. Next, there are calls to the detectors and geometry definition. In the case of EMCAL the call must be:

```
AliEMCAL * EMCAL = new AliEMCALv2("EMCAL", geometry);,
```

where `geometry = "EMCAL_COMPLETE"` or `"EMCAL_FIRSTYEAR"`, are the working geometries right now.

The simplest generator is `AliGenBox`, and it is called like this:

```
//Generate a number of particles npart uniformly distributed
//in energy and position
AliGenBox *gener = new AliGenBox(npart);
gener->SetMomentumRange(3,4); //Momentum range in GeV
gener->SetPhiRange(0,360); //Azimuthal angle range in degrees
Float_t thmin = EtaToTheta(8); // theta min. <---> eta max
Float_t thmax = EtaToTheta(-8); // theta max. <---> eta min
gener->SetThetaRange(thmin,thmax); //Longitudinal angle range dg
gener->SetOrigin(0,0,0); //vertex position
gener->SetSigma(0,0,0); //Sigma in (X,Y,Z) (cm) on IP
gener->SetPart(22); //PDG particle type generated (photon)
gener->Init(); //Init generation
```

The methods to set the generator parameters used here are general and most generators use them. To see more examples of generators and their most important parameters, please open the file `"$ALICE_ROOT/ macros/Config_gener.C"` (in this file no particle transport is requested: `gener->SetTrackingFlag(0)`, if the flag is 1 the transport is executed).

If you want to simulate only particles in the EMCAL acceptance, set the azimuthal range between 80 and 190 degrees and the pseudorapidity range `[-0.7, 0.7]`.

During the simulation, a file called **geometry.root** will be created and which will contain the transformation matrices (local to global reference system) of the volumes of the geometry.

1.2 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**.

1.3 Digitization: SDigits and Digits

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.

1.4 Raw data

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.

1.5 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
}
```

1.6 Root compilation and execution

If you are not familiarized with Root, here are some tips how to execute macros, they are equivalent:

1. Type:

/terminal prompt > **alroot**, then type

/alroot prompt > **.x TestEMCALSimulation.C**, and when it finishes type to exit

/alroot prompt > **.q**, and exit.

You must be sure that the name of the file and of the “main” are the same.

2. Type:

/terminal prompt > **alroot**, type

/alroot prompt > **.L TestEMCALSimulation.C**, then type

/alroot prompt > **TestEMCALSimulation()**, (the name of the “main” in the file). When it finishes type

/alroot prompt > **.q**, and exit.

3. Type: **alroot -q TestEMCALSimulation.C**, in the terminal command line. This is the fastest way. “**-q**” means that when it finishes it will exit from alroot. Other interesting argument is “**-b**”, it means that now display is necessary, you will not be able to open windows with canvases for example from your Root session. It is useful when sending jobs to batch.

You may pass some arguments to the macro without having to modify it. For example, you may want to change the number of events (for example from 1 to 10) or the detectors involved in the digitization (from “EMCAL” to “EMCAL TPC ...”). In the macro, you define:

```
void TestEMCALSimulation(Int_t nEvents = 1, TString detector = "EMCAL") {...},
```

and pass the new arguments in command line (all three bullets below are equivalent):

1. Type:

/terminal prompt > **alroot**, then type,

/alroot prompt > **.x TestEMCALSimulation.C(10,"EMCAL TPC")**, for example. If it finishes type,

/alroot prompt > **.q**, and exit.

2. Type:

/terminal prompt > **alroot**, then type,

/alroot prompt > **.L TestEMCALSimulation.C**, then type,

/aliroot prompt > **TestEMCALSimulation(10,"TPC EMCAL")**, for example. If it finishes type,

/aliroot prompt > **.q**, and exit.

3. Type: **aliroot -q TestEMCALSimulation.C(10,"TPC EMCAL")**, in the terminal command line. This is the fastest way.

Sometimes it is useful to compile the macro in order to be sure that what you have in the macro is formally correct. I want to suggest you to copy the standard Root macro **rootLogon.C** to your home directory or any place where your Linux environment variable "\$PATH" points to. It loads libraries, "includes", and paths necessary for compiling macros. In order to compile a macro, type in the beginning of the macro, before the method definition `#include "AliWhatever.h"` for each Root or AliRoot class that you have used in your macro. Then do (both options are equivalent):

1. Type:

/terminal prompt> **aliroot**, then type,

/aliroot prompt > **.x Simulation.C++**, and if it compiles, it will execute the macro. Type,

/aliroot prompt > **.q**, and exit.

2. Type:

/terminal prompt > **aliroot**, then type,

/aliroot prompt > **.L Simulation.C++**, and if it compiles well, then type,

/aliroot prompt > **Simulation()**. If it finishes type,

/aliroot prompt > **.q**, and exit.

When we compile a macro, a library will be created, in this case **Simulation_C.so**. Now, you can directly load this file and execute the macro, type:

/terminal prompt > **aliroot**, then type,

/aliroot prompt > **.L Simulation_C.so**, and then,

/aliroot prompt > **Simulation()**. If it finishes type,

/aliroot prompt > **.q**, and exit.

2 Reconstruction

The class AliReconstruction manages this part. This step can begin either from the Digits or from the Raw data. We can distinguish different steps described in the following sections.

2.1 Particle reconstruction

Particle passing through the calorimeter produce clusters of Digits. The reconstruction procedure has to: 1st) find which digits belong to a cluster produced by a particle (if there is any cluster); 2nd) calculate the cluster energy summing the amplitude of all digits in the cluster, amplitude calibrated to GeV values; 3rd) calculate the center of the cluster, to give the position of the particle; and 4th) calculate different cluster parameters (dispersion, ellipse axis, etc.), useful for the identification of the particle. With all these parameters, we create objects called **RecPoints** which are stored in **EMCAL.RecPoints.root**.

2.2 Cluster-Track matching

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

2.3 Event Summary Data (ESD)

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. The way to extract the information from this file is explained in the Analysis section. 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 directories. 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 analysis containers that derivate from the ESD, the AOD (Analysis Object Data) with smaller quantity of data, only those events interesting for a special physics case, for example jet studies. I am not going to discuss about that.

2.4 How to make 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() ;
}
```

3 Analysis

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 indices, 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 AliESD-track.
- 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.

References

- [1] AliRoot: ALICE offline project, <http://aliceinfo.cern.ch/Offline/>
- [2] AliRoot Documentation, <http://aliceinfo.cern.ch/Offline/AliRoot/Manual.html>
- [3] AliEn web page, <http://alien.cern.ch/twiki/bin/view/AliEn/Home>
- [4] AliRoot in SVN <http://alisoft.cern.ch/viewvc/?root=AliRoot>
- [5] EMCAL documentation, <http://aliceinfo.cern.ch/Offline/Detectors/EMCALOffline.html>
- [6] AliRoot Installation, <http://aliceinfo.cern.ch/Offline/AliRoot/Installation.html>