Table of Contents

FamosSimHit production (FamosProducer)

Complete:

Overview

The famosSimHit production is "equivalent" the g4SimHit production from GEANT-based simulation, in the sense that it produces exactly the same objects (a factor of 1000 quicker, though), i.e.,

- A collection of SimTrack, i.e., the simulated particles after the propagation and the interactions of all generated particles through/with the tracker material, all the way to the electromagnetic calorimeter entrance;
- A collection of SimVertex, i.e., the origin vertices of all aforementioned simulated particles, expressed in cm.
- A collection of PSimHit, i.e., the position (and other information) for all simulated hits in the pixel detector and silicon tracker;
- A collection of PCaloHit, i.e., the Energy left in each ECAL crystal or HCAL tower.

The random number initialization for this step is included as follows,

```
    untracked uint32 famosSimHits = 47592
```

together with the following .cff include file.

```
# Famos SimHits
   include "FastSimulation/EventProducer/data/FamosSimHits.cff"
```

This .cfi configuration file is itself included in

`FastSimulation/Configuration/data/FamosSequences.cff`

, together with a number of additional includes, documented elsewhere, needed to get to PDG particle information, the CMS geometry defintions, the magnetic field in the whole detector volume, and reconstructed object sequences (RecHits, RecTracks, and other Physics Object). The only thing you have to do therefore to include the `FamosSequences.cff` file in your cfg file, and possibly modify the relevant simulation (and reconstruction) parameters can be modified with the "replace" command. The meaning of the simulation parameters (those from the `FamosSimHits.cfi` file) is explained in detailed in turn below.

First, the common include files are grouped in a file named
`FastSimulation/Configuration/data/CommonInputs.cff`, which looks as follows:

```
 # The particle data table ESSource
include "SimGeneral/HepPDTESSource/data/pythiapdt.cfi"

#The ideal Geometry ESSource
include "Geometry/CMSCommonData/data/cmsIdealGeometryXML.cfi"

#The Tracker geometry ESSource and ESProducer
include "Geometry/TrackerNumberingBuilder/data/trackerNumberingGeometry.cfi"
include "Geometry/TrackerGeometryBuilder/data/trackerGeometry.cfi"
include "RecoTracker/GeometryESProducer/data/TrackerRecoGeometryESProducer.cfi"

#The Magnetic Field ESProducer
include "MagneticField/Engine/data/volumeBasedMagneticField.cfi"

# The Calo geometry service model
include "Geometry/CaloEventSetup/data/CaloGeometry.cff"
```

```
include "Geometry/CaloEventSetup/data/CaloTopology.cfi"
```

A first thing that can be done is to use the parametrized magnetic field in the tracker, with the following line:

```
    replace VolumeBasedMagneticFieldESProducer.useParametrizedTrackerField = true
```

This trick slightly reduces the CPU time consumption with no loss of accuracy, but not in a substantial way, because the use of the full magnetic map has already been optimized in the fast simulation, taking into account the phi-symmetry and the fact that the field values are needed only on the tracker layers.

Let's now examine the content of the `FamosSimHits.cfi` configuration file, and what is left as a handle for the user, to switch on and off a number of effects, or even better, for the developer, for tuning and comparison with the GEANT-based simulation.

General switches

As of today, the general switches are as follows.

```
module famosSimHits = FamosProducer
   {
   untracked int32 RunNumber = 1001
   bool UseMagneticField = true
   bool SimulatePileUp = false  (OBSOLETE, see below)
   bool SimulateTracking = true
   bool SimulateCalorimetry = true
   bool UseTRandomEngine = true

   ....
   }
```

The `RunNumber` and the `Verbosity` things are just a remnant of the original cfg file that was used for the GEANT-based simulation about a year ago. The latter is not used whatsoever in the Fast Simulation (hence will be safely removed from next versions of the .cfi file). The use of the former is relevant only when events are to be written out, for which Fast Simulation has not yet been used for (nor is it particularly useful for the time being, because of the very fast event processing). You can therefore safely ignore these two parameters for now. This documentation will be updated when they either become useful or are removed.

The other four parameters, in contrast, have a precise meaning, described below.

- `UseMagneticField`, when `true` (default), enables the use of the full magnetic field map indicated above. When false, a uniform, axial, magnetic field of 4 T is used throughout. The use of the latter slightly reduces the CPU time consumption, but not in a substantial way, because the use of the full magnetic map has already been optimized in the fast simulation, taking into account the phi-symmetry and the fact that the field values are needed only on the tracker layers. As a result, setting this parameter to **false will not set the magnetic field to zero**.

- `SimulatePileUp` is an obsolete (i.e., non existing) flag in CMSSW_1_8_0 and later versions, as well as with the HEAD of the branch FAMOS_1_6_0 within CMSSW_1_6_7, as the pile-up is now produced in an independent producer (see here for documentation). In earlier versions, it enables the internal fast simulation of in-time pile-up events superimposed to your signal events, by setting it to true with `replace famosSimHits.SimulatePileUp = true`. (See below for the corresponding parameter setting.) This simulation consists of reading minimum bias event root files with only the MC truth information of the primary particles therein, and append the content of a Poisson-distributed number of such events to each of your signal events. As of today, one million such events have been generated, and come automatically with the standard installation of CMSSW.

- `SimulateTracking`, when `true` (default), enables the production of the tracking PSimHits for charged particles hitting a sensitive module of the tracker. When set to `false`, the particles are propagated through the tracker material and experience all interactions with the tracker material, but no PSimHits are produced (rather time consuming: about 30 ms/event), hence no track reconstruction is possible later on. This option, useful for those doing only calorimetric studies and/or code development, requires the following `replace` statement:

```
replace famosSimHits.SimulateTracking = false
```

- `SimulateCalorimetry`, when `true` (default), enables the simulation of electromagnetic and hadron showers in the ECAL and HCAL, together with the production of PCaloHits. When set to false, the calorimeter simulation (very time consuming: about 400ms/event) is turned off. This option, useful for those doing only tracking or muon studies and/or code development, requires the following `replace` statement:

```
replace famosSimHits.SimulateCalorimetry = false
```

- `UseTRandomEngine`, when `true` (default), enforces the use of the random engine from Root (TRandom3) instead of that of CLHEP, as it is faster and more reliable. (The same is true for all producers of the Fast Simulation.)

Note that turning `SimulateTracking` and `SimulateCalorimetry` to `true` is not enough to get RecHits in theTracker and in the Calorimeters. The only thing you'll get with that is PSimHit's and PCaloHit's, from which RecHit's can later be produced, with different modules.

Internal vertex smearing

If you have not read the documentation very carefully until now, you will be surprised to see yet another vertex smearing facility in addition to that described above. The reason is threefold:

- Historical: FAMOS existed long before CMSSW, and even longer before the VtxSmeared module!;
- CPU-time related: Indeed, the `VtxSmeared` module, in its present form, is very much time consuming, in the sense that it takes the whole generated event, add a smeared vertex to all vertices of the generated events, and copy the "smeared" event to the event. All in all, it takes about 6ms/event! This procedure will be modified in versions >140 of CMSSW, by simply adding a smeared vertex (i.e., three `double` s) in the event, and letting the simulation deal with it. Meanwhile, it is strongly advised *not* to use the VtxSmeared module, but instead use the internal vertex smearing of the FastSimulation, which does exactly what the future VtxSmeared module will do in the future.
- Requested by pile-up internal simulation (see above and below)

The default smearing is done according to a beta function, but the user can choose among three possibilities (Gaussian, Flat or No vertex smearing) by commenting in their local copy of =FamosSimHit_cfi.py = the following line.

```
from FastSimulation.Event.Early10TeVCollisionVertexGenerator_cfi import *
```

and replace it with one of the file listed in this directory. The pile-up is generated in an other module, therefore the local copy of `FastSimulation/PileUpProducer/python/PileUpProducer_cff.py` has to modified by hand accordingly. The Fast Sim is not able to detect any inconsistency between the two files.

The parameters applied in the vertex smearing are taken from this file . As an example, the parameters of the so-called `Early10TeVCollisionVtxSmearingParameters` smearing are as follows:

```
Early10TeVCollisionVtxSmearingParameters = cms.PSet(
    Phi = cms.double(0.0),
```

```
    BetaStar = cms.double(300.0),
    Emittance = cms.double(7.03e-08),
    Alpha = cms.double(0.0),
    SigmaZ = cms.double(3.8),
    TimeOffset = cms.double(0.0),
    Y0 = cms.double(0.0),
    X0 = cms.double(0.0322),
    Z0 = cms.double(0.0)
)
```

while, the `GaussVtxSmearingParameters` smearing looks like

```
GaussVtxSmearingParameters = cms.PSet(
    MeanX = cms.double(0.0),
    MeanY = cms.double(0.0),
    MeanZ = cms.double(0.0),
    SigmaY = cms.double(0.0015),
    SigmaX = cms.double(0.0015),
    SigmaZ = cms.double(5.3),
    TimeOffset = cms.double(0.0)
)
```

If needed, the parameters can be modified in the config file with the following syntax.

```
# Set the early collions 10TeV parameters (as in the standard RelVals)
process.famosSimHits.VertexGenerator.SigmaZ=cms.double(3.8)
process.famosSimHits.VertexGenerator.Emittance = cms.double(7.03e-08)
process.famosSimHits.VertexGenerator.BetaStar = cms.double(300.0)
```

Important remark 1 : the Fast Sim completely ignores the MeanX, MeanY and MeanZ* values that are present in these files, and uses the values returned by the `offlineBeamSpot` instead. Therefore, it is useless trying to modify MeanX/Y/Z with such a replace. If you want to modify the beam spot position, have a look here. **Important remark 2** : if any of the vertex smearing parameters are modified here, they must be modified identically for pile-up events, as explained here.

It is important to note that the Fast Simulation code is smart enough to notice whether or not the vertex was already smeared (e.g., if you read the MC truth from a fully-simulated event sample, or if you have a VtxSmeared module enabled in your configuration file), and will never smear the vertex twice, even if requested.

Particle Filter

Not all the particles are actually processed through the FastSimulation (in the same manner as not all the particles are processed through the full simulation), and not all the particles are saved in the SimTrack container, to save disk space. In the Fast Simulation, neutrinos are by default ignored. The kinematics of the particles to be processed through the Fast Simulation are defined in

```
# Kinematic cuts for the particle filter in the SimEvent
   include "FastSimulation/Event/data/ParticleFilter.cfi"
```

in which the smallest transverse momentum (for charged particles), the smallest energy (for all particles) and the largest pseudo-rapidity absolute value is given, according to

```
PSet ParticleFilter =
{
# Particles with |eta| > etaMax (momentum direction at primary vertex)
# are not simulated
   double etaMax = 5.0
# Charged particles with pT < pTMin (GeV/c) are not simulated
   double pTMin =  0.200
```

```
# Particles with energy smaller than EMin (GeV) are not simulated
   double EMin = 0.100
}
```

Obviously, only the stable or quasi-stable particles are indeed processed. These particles, together with their mothers and the corresponding origin vertices, are saved in the SimTrack and SimVertex container.

Pile-Up simulation (obsolete)

The following documentation is no longer valid in CMSSW_1_8_0 and later versions. A new documentation is available in SWGuideFastSimPileUp. This new facility has also been backported to CMSSW_1_6_7, if the HEAD of the FAMOS_1_6_0 branch is used.

More ▸ Less ▾

In-time pile-up events can be simulated internally in the FamosProducer for `CMSSW` versions higher than `150pre1`. To do so, `SimulatePileUp` has to be set to `true` (default value) in your cfg file, as indicated above. This option is turned off by default in earlier versions. Options of the pile-up simulation are steered from the following `.cfi` file

```
# PileUp Event Generation
   include "FastSimulation/PileUpProducer/data/PileUpSimulator.cfi"
```

which contains the following `ParameterSet`:

```
# Take pileup events from files
PSet PileUpSimulator =
{
# Special files of minimum bias events (generated with
# cmsRun FastSimulation/PileUpProducer/test/producePileUpEvents.cfg)
         untracked vstring fileNames = {
      "MinBiasEvents_01.root",
      "MinBiasEvents_02.root",
      "MinBiasEvents_03.root",
      "MinBiasEvents_04.root",
      "MinBiasEvents_05.root",
      "MinBiasEvents_06.root",
      "MinBiasEvents_07.root",
      "MinBiasEvents_08.root",
      "MinBiasEvents_09.root",
      "MinBiasEvents_10.root"
   }
   double averageNumber = 5
# The file with the last minimum bias events read in the previous run
# to be put in the local running directory (if desired)
   untracked string inputFile = "PileUpInputFile.txt"
}
```

In the above include file, the average number of minimum bias events to be superimposed to the signal event can be replaced with the following statement:

```
replace famosSimHits.PileUpSimulator.averageNumber = 2
```

The files `MinBiasEvents_*.root` just contain the primary particle momenta (px,py,pz,m,id) of for one million minimum bias events, produced, e.g., by `Pythia`, with the `FastSimulation/PileUpProducer/test/producePileUpEvents.cfg` configuration file. These files are automatically downloaded for each release, pre-release or nightly, and in your local `CMSSW` setup when the first `scramv1 b` is issued. More files were generated (up to 10 million events), copied to CASTOR and can be added to the above list if you so wish.

The number N of in-time pile-up events to be superimposed to a bunch crossing is Poisson distributed around `averageNumber`. The N requested events are read randomly from any of the files for each new signal event, their primary vertices are smeared with the aforementioned internal vertex smearer (the parameters of which must therefore be identical to those of the VtxSmeared module) and the corresponding particles are added to the SimTrack (and SimVertex) containers, and then decayed and propagated as any other particle of the signal event.

Note that this part of the code has undergone a careful timing & disk space optimization (600Bytes/events on disk, and 220ms for simulating low-lumi pile-up). Even after this optimization, it is possible that the pile-up mixing with the official mixing module is more time-effective. Indeed, the internal pile-up simulation must produce PSimHit and PCaloHit for all the pile-up particles (hence track these particles through the entire fast simulation), while the pile-up mixing module mixes PSimHits, PCaloHits and even TrackerRecHits and TrackCandidates already produced in a previous run of the FamosProducer with minimum bias events. On the other hand, there is a framework overhead in the MixingModule of ~80ms/event for low-lumi pile-up and of ~500ms/event for high-lumi pile-up.

Stay tuned, then!

The last line of the above parameter list deserves some more comments. Indeed, the minimum bias events are read sequentially from each of the above files (file choice is ramdom), starting by default from the first event in each file. This technique, as smart as it is, breaks however the random reproducibility of the simulation. For example, if the Fast Simulation experiences a crash at event #43687, that the user would like to reproduce without having to simulate again the 43686 first events, it would not work because the pile-up sequence would not be reproduced. For this reason, a file named `PileUpOutputFile.txt` is saved in the local directory after each simulated event, with the current pointers in each of the files. Renaming this file to `PileUpInputFile.txt` would cause the next Fast Simulation job to start from these pointers, hence reproduce the crash of event #43687 directly (provided that the proper sequence of random numbers for all modules is also (i) saved and (ii) restored, according to the documentation of the RandomNumberGeneratorService).

Propagation and material effects in the tracker

After the stable or quasi-stable generated particles from the signal and pile-up events, in the detector acceptance, are selected, they are propagated in turn through the magnetic field all the way to the calorimeters (Preshower, ECAL, HCAL). The propagation is done in steps, each step corresponding to a layer (sensitive or not) of the tracker. These layers are assumed to be disposed as a series of infinitely thin nested cylinders, where the material - assumed to be silicon - is concentrated, either on the barrel or on the end-cap part (depending on whether the layer is a barrel or an end-cap layer). The thickness of each layer in terms of radiation lengths is tuned to reproduce the amount of material experienced by particles when they traverse it, in particular the amount of Bremsstrahlung for electrons. The configuration of the Tracker Interaction Geometry is explained in details hereThe (analytical) propagation from one layer to the next one is performed under the assumption of an axial, locally uniform, magnetic field, the value of which is determined at the particle intersection with the first of the two layers.

The conditions under which these material effects are simulated are compiled in the following .cfi file,

```
# Material effects to be simulated in the tracker material and associated cuts
   include "FastSimulation/MaterialEffects/data/MaterialEffects.cfi"
```

which contains only one parameter set

```
# Material effects to be simulated in the tracker material and associated cuts
PSet MaterialEffects =
{
...
}
```

In general, it is advised not to change any of the parameters below (except to switch on and off the various interactions). The current settings were determine both the realism of the simulation and to keep the CPU time in reasonable limits. For this latter reason, the material effects are not simulated for pseudo-rapidity values in excess of 3.0, and the particle propagation is performed in one step to the HCAL in that case. This value cannot be changed via the MaterialEffects parameter set: decreasing this value would make the simulation unrealistic, and increasing it would have no effect, as there is no material beyond a pseudo-rapidity of 3.0 in the Fast Simulation geometry.

Photons may or not convert, according to the following steering switches (the comments explain it all),

```
# Enable photon pair conversion
   bool PairProduction = true
# Smallest photon energy allowed for conversion (in GeV)
      double photonEnergy = 0.100
```

and electrons may experience Bremsstrahlung, if requested like:

```
# Enable electron Bremsstrahlung
   bool Bremsstrahlung = true
# Smallest bremstrahlung photon energy (in GeV)
      double bremEnergy = 0.100
# Smallest bremsstrahlung energy fraction (wrt to the electron energy)
      double bremEnergyFraction = 0.005
```

In both cases, the particles created in the process (e+e- for photon conversions, and photons for Bremsstrahlung) are added the the SimTrack container, for later Fast Simulation. Vertices corresponding to the position of interaction are also added to the SimVertex container. Note, however, that (unlike in the GEANT-based simulation), a Bremsstrahlung vertex has only one daughter (the photon). In the GENAT-based simulation, this vertex would have two daughters: the photon and the modified electron.

Charged particles may experience energy loss by ionization (dE/dx) and multiple Coulomb scattering, steered with the following switches:

```
# Enable dE/dx
   bool EnergyLoss = true
# Enable Multiple Scattering
   bool MultipleScattering = true
# Smallest pT (in GeV/c) for multiple scattering treatment.
      double pTmin = 0.500
```

In these two cases, no SimTrack/!SimVertex are added to the corresponding containers. The energy and direction changes are just internally followed until the particle reaches the calorimeters.

Photon conversion, electron Bremsstrahlung, dE/dx and multiple Coulomb scattering are simulated analytically (i.e., with analytical probability density functions). The references chosen for these analytical formulae are as follows.

- Photon conversions: An old version of the Review of Particle Physics (I could not get hold of it yet) for the e- energy distribution, and a customized GEANT3 analytical parametrization for the e+ and e- angular distributions.
- Electron Bremsstrahlung: Review of Particle Physics, Journal of Physic G (2006), p. 263/264, section 27.4, for the number and the energies of the photons emitted, and a customizedGEANT3 analytical parametrization for the angular distribution;
- dE/dx: C. Grupen, "Physics of particle detection", physics/9906063;
- Multiple scattering: Review of Particle Physics, Journal of Physic G (2006), p. 262, section 27.3.

These interactions can be turned of simultaneously or individually with any combination of the following

`replace` statements:

```
replace famosSimHits.MaterialEffects.PairProduction = false
replace famosSimHits.MaterialEffects.Bremsstrahlung = false
replace famosSimHits.MaterialEffects.EnergyLoss = false
replace famosSimHits.MaterialEffects.MultipleScattering = false
```

Unlike the previously described interactions, nuclear interactions of hadrons in the tracker material are not simulated analytically - because there is no satisfactory analytical ersatzt. First, all nuclear interactions are modelled like pion-proton elastic or inelastic interactions. The thickness of each tracker layer in terms of interaction lengths is pragmatically assumed to be 25% of the thickness in terms of radiation lengths, for pions of 5 GeV. The variation of the inelastic cross section with the pion energy, and the ratio elastic/inelastic is taken from experimental measurements (Review of Particle Physics, Journal of Physic G (2006), p. xxx, Figure xx.x).

Librairies of inelatic pion-proton inelastic interactions are built from single pions fully simulated at various energies, namely 1, 2, 3, 5, 9, 15, 20, 30, 50, 150, 225 and 300 GeV, for the time being. (It is hoped to complete it in the future with a point at 100 GeV, and a point at 1000 GeV). The nuclear interactions are written therein as vectors of "Particles" (a Particle = {px*, py*, pz*, mass, type}, where p* is determined in the pion-proton centre-of-mass frame, and is scaled to the pion-proton centre-of-mass energy) in a customized and fast-access Root Tree. In the Fast Simulation, for each hadron of energy E selected to produce an inelastic interaction, a vector of "Particles" is read from the file with one of the two closest energies, the Particle momenta are multiplied by the local pion-proton centre-of-mass energy, and boosted back to the laboratory frame. A end SimVertex is added to the hadron that experienced an inelastic nuclear interaction, and a number of SimTrack's is added to the event, for later Fast Simulation.

When a hadron is selected to experience an elastic interaction, its direction is changed analytically according to F.W. Jones, Triumf design note TRI-DN-92-K152 (1990), i.e., proportionally to m/p. No SimVertex/!SimTrack is added to the event in that case (as is done in the GEANT-based simulation).

The steering of Nuclear Interactions is summarized below, for CMSSW versions up to CMSSW_1_6_7 on the one hand, and from CMSSW_1_6_8 onwards, on the other . The librairies are saved in /afs/cern.ch/cms/data/CMSSW/FastSimulation/MaterialEffects/, and are automatically downloaded with `scram` in each CMSSW release.

- Up to CMSSW_1_6_7:

```
# Enable Nuclear Interactions
   bool NuclearInteraction = true
# The libraries of nuclear interactions (ordered by increasing energies)
      untracked vstring fileNames = {
         "CMS.NuclearInteractions_1GeV.root",
         "CMS.NuclearInteractions_2GeV.root",
         "CMS.NuclearInteractions_3GeV.root",
         "CMS.NuclearInteractions_5GeV.root",
         "CMS.NuclearInteractions_9GeV.root",
         "CMS.NuclearInteractions_15GeV.root",
         "CMS.NuclearInteractions_20GeV.root",
         "CMS.NuclearInteractions_30GeV.root",
         "CMS.NuclearInteractions_50GeV.root",
         "CMS.NuclearInteractions_150GeV.root",
         "CMS.NuclearInteractions_225GeV.root",
         "CMS.NuclearInteractions_300GeV.root"
      }
# The energies of the pions used in the above files (same order)
            untracked vdouble pionEnergies = {
   1., 2., 3., 5., 9., 15., 20., 30., 50., 150., 225., 300.}
# The scaling of the inelastic cross section with energy
      untracked vdouble ratioRatio =  {
   0.0892455, 0.613859, 0.897045, 1.00000, 0.971305, 0.956828,
```

```
   0.968624,0.975242, 0.944918, 0.930526, 0.929239, 0.915461 }
# The smallest pion energy for which nuclear interactions are simulated
      double pionEnergy = 0.500
# The ratio between interaction lengths and radiation lengths in the tracker at 5 GeV
      double lengthRatio = 0.2508
# The file with the last nuclear interaction read in the previous run
# to be put in the local running directory (if desired)
      untracked string inputFile = "NuclearInteractionInputFile.txt"
}
```

- From CMSSW_1_6_8 onwards

```
    # Enable Nuclear Interactions
    bool NuclearInteraction = true
    # The energies of the pions used in the above files (same order)
    untracked vdouble pionEnergies = {
        1., 2., 3., 4., 5., 7., 9., 12., 15., 20., 30.,
        50., 100., 200., 300., 500., 700., 1000.
    }
    # The particle types simulated
    untracked vint32 pionTypes = {
        211, -211, 130, 321, -321, 2212, -2212, 2112, -2112
    }
    # The corresponding particle names
    untracked vstring pionNames = {
        "piplus", "piminus", "K0L", "Kplus", "Kminus", "p", "pbar", "n", "nbar"
    }
    # The corresponding particle masses
    untracked vdouble pionMasses = {
        0.13957, 0.13957, 0.497648, 0.493677, 0.493677,
        0.93827, 0.93827, 0.939565, 0.939565
    }
    # The corresponding smallest momenta for which an inleatic interaction may occur
    untracked vdouble pionMinP = {
        0.7, 0.0, 1.0, 1.0, 0.0, 1.1, 0.0, 1.1, 0.0
    }
    # The scaling of the inelastic cross section with energy
    untracked vdouble ratios = {
        // pi+ (211)
        0.031390573,0.531842852,0.819614219,0.951251711,0.986382750,1.000000000,0.985087033,0.982
        0.990832192,0.992237923,0.994841580,0.973816742,0.967264815,0.971714258,0.969122824,0.978
        0.977312732,0.984255819,
        // pi- (-211)
        0.035326512,0.577356403,0.857118809,0.965683504,0.989659360,1.000000000,0.989599240,0.980
        0.988384816,0.981038152,0.975002104,0.959996152,0.953310808,0.954705592,0.957615400,0.961
        0.965022184,0.960573304,
        // K0L (130)
        0.000000000,0.370261189,0.649793096,0.734342408,0.749079499,0.753360057,0.755790543,0.755
        0.751337674,0.746685288,0.747519634,0.739357554,0.735004444,0.803039922,0.832749896,0.890
        0.936734805,1.000000000,
        // K+ (321)
        0.000000000,0.175571717,0.391683394,0.528946472,0.572818635,0.614210280,0.644125538,0.670
        0.685144573,0.702870161,0.714708513,0.730805263,0.777711536,0.831090576,0.869267129,0.915
        0.953370523,1.000000000,
        // K- (-321)
        0.000000000,0.365353210,0.611663677,0.715315908,0.733498956,0.738361302,0.745253654,0.751
        0.750628335,0.746442657,0.750850669,0.744895986,0.735093960,0.791663444,0.828609543,0.889
        0.940897842,1.000000000,
        // proton (2212)
        0.000000000,0.042849136,0.459103223,0.666165343,0.787930873,0.890397011,0.920999533,0.937
        0.950920131,0.966595049,0.979542270,0.988061653,0.983260159,0.988958431,0.991723494,0.995
        1.000000000,0.999962634,
        // anti-proton (-2212)
        1.000000000,0.849956907,0.775625988,0.802018230,0.816207485,0.785899785,0.754998487,0.728
        0.710010673,0.670890339,0.665627872,0.652682888,0.613334247,0.647534574,0.667910938,0.689
        0.709200185,0.724199928,
```

```
        // neutron (2112)
        0.000000000,0.059216484,0.437844536,0.610370629,0.702090648,0.780076890,0.802143073,0.819
        0.825829666,0.840079750,0.838435509,0.837529986,0.835687165,0.885205014,0.912450156,0.951
        0.973215562,1.000000000,
        // anti-neutron
        1.000000000,0.849573257,0.756479495,0.787147094,0.804572414,0.791806302,0.760234588,0.741
        0.724118186,0.692829761,0.688465897,0.671806061,0.636461171,0.675314029,0.699134460,0.724
        0.742556115,0.758504713
    }

    #
    # The smallest pion energy for which nuclear interactions are simulated
    double pionEnergy = 0.500
    # The ratio between radiation lengths and interation lengths in the tracker at 15 GeV
    vdouble lengthRatio = {
#        pi+      pi-    K0L      K+      K-      p      pbar     n      nbar
#       0.2508, 0.2549, 0.3380, 0.2879, 0.3171, 0.3282, 0.5371, 0.3859, 0.5086 # before 170 tunin
        0.2257, 0.2294, 0.3042, 0.2591, 0.2854, 0.3101, 0.5216, 0.3668, 0.4898 # after 170 tuning
    }
    # and a global fudge factor for TEC Layers to make it fit
    double fudgeFactor = 1.20 # after 170 tuning

    # The correspondence between long-lived hadrons/ions and the simulated hadron list
    untracked vint32 protons = { 2212, 3222, -101, -102, -103, -104 }
    untracked vint32 antiprotons = { -2212, -3222 }
    untracked vint32 neutrons = { 2112, 3122, 3112, 3312, 3322, 3334 }
    untracked vint32 antineutrons = { -2112, -3122, -3112, -3312, -3322, -3334 }
    untracked vint32 K0Ls = { 130, 310 }
    untracked vint32 Kplusses = { 321 }
    untracked vint32 Kminusses = { -321 }
    untracked vint32 Piplusses = { 211 }
    untracked vint32 Piminusses = { -211 }

    # The file with the last nuclear interaction read in the previous run
    # to be put in the local running directory (if desired)
    untracked string inputFile = "NuclearInteractionInputFile.txt"
```

While essentially everything in the above parameter list is self explanatory, the last line deserves some more comments. Indeed, the inelastic nuclear interactions are read sequentially from each of the above libraries, starting by default from the first interaction in each library. This technique, as smart as it is, breaks however the random reproducibility of the simulation. For example, if the Fast Simulation experiences a crash at event #43687, that the user would like to reproduce without having to simulate again the 43686 first events, it would not work because the nuclear interaction sequence would not be reproduced. For this reason, a file named `NuclearInteractionOutputFile.txt` is saved in the local directory after each simulated event, with the current pointers in each of the libraries. Renaming this file to `NuclearInteractionInputFile.txt` would cause the next Fast Simulation job to start from these pointers, hence reproduce the crash of event #43687 directly (provided that the proper sequence of random numbers for all modules is also (i) saved and (ii) restored, according to the documentation of the RandomNumberGeneratorService).

Finally, the nuclear interactions can be turned off with the following `replace` statement:

```
replace famosSimHits.MaterialEffects.NuclearInteraction = false
```

Particle decays

Along their propagation through the tracker material and in the magnetic field, quasi-stable particles (i.e., not already decayed at the generator level, like K0's, Lambda's, K+, pi+, ...) may experience a decay in flight, according to the relevant lifetime. These decays are simulated with the PYDECY fortran subroutine, and are activated by default, through the following Parameter Set,

```
# (De)activate decays of unstable particles (K0S, etc...)
```

```
   include "FastSimulation/TrajectoryManager/data/ActivateDecays.cfi"
```

which contains only one line:

```
# Set to true to activate decays of unstable particles (K0S, etc...)
   bool ActivateDecays = true
```

When a particle decay occurs, the SimTrack and SimVertex containers are updated with the decay products, for later Fast Simulation.

Particle decays can be turned off with

```
replace famosSimHits.ActivateDecays.ActivateDecays = false
```

Tracker PSimHit production

The PSimHit's for charged particles are determined as the intersections of the charged particle trajectories and the actual sensitive modules of the tracker, the position of which is taken from the tracker reconstruction geometry. These hits are saved in the event only if `famosSimHits.SimulateTracking` is set to `true` (default). They are then taken as input to generate the smeared Tracker RecHit's. To save CPU time, not all charged particles give PSimHit's. Additional conditions are given with the following configuration file:

```
# Conditions to save Tracker SimHits
   include "FastSimulation/TrajectoryManager/data/TrackerSimHits.cfi"
```

the content of which is as follows:

```
PSet TrackerSimHits =
{
# Smallest charged particle pT for which SimHit's are saved (GeV/c)
   untracked double pTmin = 0.8
# Save SimHit's only for the first loop
   untracked bool firstLoop = true
}
```

and which can be modified in the usual way, e.g.,

```
replace famosSimHits.TrackerSimHits.pTmin = 0.5
replace famosSimHits.TrackerSimHits.firstLoop = false
```

The first parameter (`pTmin`) indicates that charged particles with a momentum transverse to the beam smaller than 800 MeV/c won't release PSimHit's. This cut is motivated by the fact that only tracks with a momentum in excess of 900 MeV/c are currently reconstructed in the complete reconstruction. Should this cut be loosened in the future, the default value would follow this move. The second parameter (`firstLoop`) indicates that only the first half loop is bound to give SimHit's. It saves a great deal of CPU time for low momentum loopers.

Note:It is important to note that the FastSimulation includes a fast track reconstruction, in which **only** the hits created by a given charged particle are fitted to form a reconstructed tracks, hence *de facto* turning off the fake rate. (This approximation is justified by the fact that the fake rate is found to amount to a few per mil at low luminosity.) The hits are therefore saved track by track, ordered according to their distance to the primary vertex, for a faster later access by the track fitting algorithms. If the complete track reconstruction (with full pattern recognition) is used instead - which would be much more CPU-time demanding, but which is needed either for comparison studies and tuning, or for SLHC initial studies - the above two cuts would have to be somewhat loosened to reproduce the proper amount of fake tracks. To be complete in this documentation, the reader must know that the "full pattern recognition option" is not yet available in the FastSimulation, but that its implementation is foreseen to allow for the aforementioned detailed studies.

Geometry: While the interactions with the tracker material are simulated in a specific, simplified, nested-cylinder-based, tracker geometry, the PSimHit are placed instead in the real, official, tracker geometry, called "Tracker Reconstruction Geometry", in which all the sensitive modules are present, and obtained from the following ESProducer:

```
include "RecoTracker/GeometryESProducer/data/TrackerRecoGeometryESProducer.cfi",
```

included in `FastSimulation/Configuration/data/CommonInputs.cff`.

A particle is first propagated to one of the nested cylinders the "interaction geometry" (defined and hard-coded in `FastSimulation/TrackerSetup/src/TrackerInteractionGeometry.cc`), in which it interacts, as described earlier in this documentation. The coordinates (radii and lengths) of the sensitive interaction-geometry nested cylinders are determined from the Tracker Reconstruction Geometry, but their thicknesses are hard-coded. The coordinates of dead-material cylinders are hard coded and must be compatible with (i.e., nested within) the sensitive cylinders. The particle, if charged and if it satisfies the above cuts, is then propagated to the closest sensitive module(s) of the corresponding layer of the "reconstruction geometry", in which PSimHit's are recorded.

SLHC studies require the reconstuction geometry (hence the interaction geometry) to be modified, so as to test a number of possible layouts. This can only be achieved by making both geometries configurable, with a ParameterSet of radii, lengths and thicknesses for the dead material in the interaction geometry, an XML file for the reconstruction geometry, and an algorithm to determine the radii, lengths and thicknesses for the sensitive material in the interaction geometry. This work is currently in progress in the SLHC community. Stay tuned!

Calorimeter PCaloHit simulation

In the ECAL and in the HCAL, the simulation is done in two steps. A shower, made of spots, i.e a point in space associated to an energy deposit, is generated. Then, the shower is transported into the relevant detector. The shower generation is done `FastSimulation/ShowerDevelopment`, the properties of the calorimeters, such as their density in terms of radiation and interaction length, are used in the process, they are stored in `FastSimulation/CalorimeterProperties`. The second step is done in `FastSimulation/CaloHitMakers`. The full process is controlled by the `CalorimetryManager` present in `FastSimulation/Calorimetry` and needs the following parameter configuration file.

```
# FastCalorimetry
   include "FastSimulation/Calorimetry/data/Calorimetry.cff"
```

Electromagnetic showers

The Grindhammer parameterization of electron showers is used. The longitudinal profile of the shower is modeled by a gamma function, and the lateral profile by the sum of two functions, one for the core of the shower and the other for the tail. The shower-to-shower fluctuation are simulated. In its default version, a 40 GeV shower, made of about 10000 spots is generated in less 10ms. The number of spots can however be reduced, without loss of accuracy, thus allowing the shower to be faster moved into the calorimeter. The energy of each spot is multiplied accordingly.

```
#SpotFraction < 0 <=> deactivated. In the case, CoreIntervals and
#TailIntervals are used
        double SpotFraction = -1.
```

SpotFraction is the fraction of spots wrt. the Grindhammer parametrization used for the shower simulation. SpotFraction=1 corresponds to the default parametrization. A rough reduction of the number of spots tends to degrade the shower simulation, especially the lateral profile. Therefore, it is advised to put SpotFraction to a negative value. In this case, the following default following parameters are used:

```
# For the core 10% of the spots
        vdouble CoreIntervals = {
          100.,
          0.1}
# For the tail 10% of r<1RM. 100% otherwise
        vdouble TailIntervals = {
          1.,
          0.1,
          100,
          1.}
```

In the core of the shower, only 10% of the spots are kept without loss of accuracy. In the tail, close to the core of the shower, i.e, for r<1 Moliere radius, the number of spots is divided by 10, while for r>1 Moliere radius, 100% of the spots are kept.

After having been simulated in an homogeneous medium, the shower has to be transported in the calorimeter. Several essential effects are taken into account in the process:

- The front leakage The energy deposited in front of the crystals most of the time is actually collected by the crystals. `FrontLeakageProbability = 1`. This parameter has not been tuned, and currently, all the energy deposited in front of the crystals is supposed to be collected.

- The inter-module gaps A fraction of the energy deposited between the ECAL modules is lost `GapLossProbability = 0.9` This parameter has not been tuned, and currently it is supposed that 10% of the energy is lost. (Humm.. the name of this parameter should probably be changed)

- The rear leakage, i.e., the energy deposited between the ECAL and the HCAL is lost. There is no control parameter for that. The high energy shower can deposit some energy in the HCAL.

  ```
  ## Watch out ! The following two values are defined wrt the electron shower simulatio
  ## There are not directly related to the detector properties
  double HCAL_PiOverE = 0.2
  double HCAL_Sampling = 0.0035
  ```

  These parameters have been tuned with OSCAR/ORCA. The so-called H/E distribution for high energy electron is not expected to be well simulated currently. Indeed, the current distribution of this variable in the GEANT-based simulation and reconstruction is not realistic, since there is no zero-suppression applied in the HCAL. Therefore it has not been possible to tune it.
- The enlargment of the shower due to the magnetic field. This is effect is simulated, but only in the barrel. It is supposed to be negligible in the endcaps.

Several parameters present of electromagnetic shower part of Calorimetry.cfi have not been described yet:

- To obtain a good agreement between the GEANT-based and the fast simulation, the electromagnetic showers as obtained from the Grindhammer parameterization have to be enlarged by this factor. It is important to mention that several studies have shown that the showers are narrower in the data (test-beam) than in the GEANT-based simulation. To obtain a good agreement between the fast simulation and the test beam data, this parameter should be set to 1. `RadiusFactor = 1.096` As of CMSSW_1_8_0. It is possible to tune the radius of the core and of the tail of the shower independentely with the two `RCFactor` and `RTFactor` factors respectively. They are set by default to 1.

- By default, the showers are simulated in a 7x7 crystal grid (`GridSize=7`) An energy dependant grid size is foreseen in the future. A 7x7 crytal grid has been proved to be sufficient for "standard" analysis. A larger grid can be used for analysis at the TeV scale.

- In the preshower, the energy deposit, deduced from the thickness(in cm) of the layer is turned into a number of mips. This is controlled by the following parameters.

```
double PreshowerLayer1_mipsPerGeV = 35.7
double PreshowerLayer1_thickness = 1.6
double PreshowerLayer2_mipsPerGeV = 59.5
double PreshowerLayer2_thickness = 0.38
```

Hadron showers

Muons

Muon parametrization

The muon parameterization (see documentation here) only needs as an input the SimTrack's from muon that reach the calorimeters, and the tracks actually reconstructed in the tracker (see documentation here) . While these muons are already contained in the collection of SimTrack's produced by the FamosProducer module, the CPU time needed by a loop on these SimTrack's for each reconstructed tracks is a little large (10ms/event). To overcome this effect, a secondary collection of SimTrack's is produced and written to the event, later recoverable with

```
Handle<std::vector<SimTrack> > simMuons;
iEvent.getByLabel("famosSimHits","MuonSimTracks",simMuons);
```

which contains only the muons that reach the calorimeters, with a pT in excess of 1 GeV/c and a pseudo-rapidity between -3.0 and +3.0. It allows the timing of the muon parametrization module to be reduced by a factor ~100 (down to ~0.1 ms/event).

Muon PSimHit simulation

The same simulated muons SimTrack's as above are taken as an input, or more precisely, their position and momentum at the calorimeter entrance. The SimHit production, however is done in a separate producer, and is described elsewhere

Review status

Reviewer/Editor and Date (copy from screen)	Comments
FlorianBeaudette - 13 Dec 2007	Added zero magnetic field documentation
PatrickJanot - 31 October 2007	Update pile-up documentation (now obsolete in this producer)
FlorianBeaudette - 8 October 2007	Updated the pile-up simulation
PatrickJanot - 18 June 2007	Add documentation for muon parameterization input
PatrickJanot - 18 June 2007	Improve Tracker PSimHit documentation
FlorianBeaudette - 10 June 2007	Add calorimeter PCaloHit documentation
PatrickJanot - 28 May 2007	Add Tracker PSimHit documentation
PatrickJanot - 25 May 2007	Documentation updates for 150 release
PatrickJanot - 26 Apr 2007	Describe in-time pile-up internal simulation
PatrickJanot - 4 Mar 2007	Nuclear interaction and particle decay simulation
PatrickJanot - 3 Mar 2007	Bremsstrahlung, photon conversion, dE/dx and multiple scattering simulation
PatrickJanot - 2 Mar 2007	

	External data, general switches, internal vertex smearing, internal pile-up simulation
JennyWilliams - 23 Feb 2007	created template page

Responsible: AndreaGiammanco
Last reviewed by: Reviewer

This topic: CMSPublic > SWGuideFastSimSimHit
Topic revision: r27 - 2011-12-14 - AndreaGiammanco