Developing in Key4HEP: good practice and advice

Alvaro Tolosa-Delgado (CERN)

FCC physics week 2024, Annecy

Jan. 30th, 2024

Developing in Key4hep

- What is Key4hep
- Gaudi, framework of FCC software
- Full simulation
- Appendix
  - How to use HTCondor & EOS for heavy duty jobs
  - Some recommendations when implementing the geometry of a detector in DD4hep/Geant4

Developing in Key4hep

- What is Key4hep
  - Software stack
  - How to work with local installation of particular packages
- Gaudi, framework of FCC software
- Full simulation
- Appendix

What is Key4hep

- Key4hep is a software stack, distributed via CVMFS, which provides the required packages and repositories for future collider studies
- The software stack includes many packages:
  - HEP Software tools: ROOT, Geant4, DD4hep, Gaudi…
  - Detector implementation: k4geo
  - Event data model: podio, EDM4hep
  - Reconstruction and Particle flow: ACTS, Pandora, CLUE
  - FCCSW
  - ILCsoft: Marlin, LCIO, LCFiPlus
  - CEPCSW
  - ...

What is Key4hep

- There are two stack flavors according to the release frequency:
  - **Nightlies**, all packages are built on a daily basis, using the last version of each package

```
source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
```

  - **Stable**, packages are built on a monthly basis,

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

- The main difference is that stable is thoroughly tested, while nightlies may include some broken dependencies. Older releases of the nightly stack may be used, and they are available here:

```
ls -lah /cvmfs/sw-nightlies.hsf.org/key4hep/releases
```

- Active discussion in the github repository and the bi-weekly key4hep meetings

Key4hep stack. Work with local installation

- It is possible to work with a local installation of a package instead of the corresponding one available in the key4hep stack (from CVMFS)
- When working with a local installation of a package:
  - **Check that is possible**, ask in case of doubt
  - **Use the same key4hep stack to compile** the package to ensure compatibility with the other packages
  - **Remove paths** pointing to the key4hep (CVMFS) package in the corresponding environmental variables: `LD_LIBRARY_PATH,PATH, PYTHONPATH, ROOT_INCLUDE_PATH, CMAKE_PREFIX_PATH`
  - **Add** the path of the local version of the package to the corresponding environmental variables
  - **Update** environmental variable that holds the path to installed components of the package. The variables that may be affected are shown with the command: `env | grep 'K4'`

Key4hep stack. Example of local installation of k4geo

Remove central k4geo from LD_LIBRARY_PATH variable:

```
export LD_LIBRARY_PATH=`echo $LD_LIBRARY_PATH | tr ':' '\n' | grep -Ev "/k4geo/" | tr '\n' ':' `
```

Now we can build the local version of k4geo

```
git clone -b CLD_with_ARC
https://github.com/atolosadelgado/k4geo.git
cd k4geo/
cmake -B build -S . -D CMAKE_INSTALL_PREFIX=install
cmake --build build -j 6 -- install
export LD_LIBRARY_PATH=$PWD/install/lib:$LD_LIBRARY_PATH
```

Other repositories may need to update other environmental variables as well: `PATH, PYTHONPATH, ROOT_INCLUDE_PATH, CMAKE_PREFIX_PATH`

Developing in Key4hep

- What is Key4hep
- Gaudi, framework of FCC software
  - EDM4hep as FCC event data model
  - Gaudi functionals
- Full simulation
- Appendix

Gaudi as framework for FCC software

- Gaudi is a framework software package that is used to build data processing applications [link to docs, link],
- The input data and newly generated data during the execution is stored in the so-called **Event Transient Store (ETS)**
- The input data is event-wise (see next slide about data format)
- Each event is processed by a chain of functions called **algorithms**
    - Stateless functions (that is, they do not store state between events) grant scalability and thread safety. However, not all algorithms are stateless.
- Some functionality is needed during the whole process, such as the random number generation or the geometry description. This functionality is encapsulated in the so-called **services**.
- Part of the functionality can be encapsulated into an external object called **tool**, and assigned to the algorithm/service at initialization time in the steering file
- To list all the algorithms and services available use the command: **`k4run -l`**

Gaudi steering file. Material scan example

- Example of a Gaudi steering file that performs a material scan of a detector

```
# Name of file: materialScan.py
# To execute: k4run materialScan.py

# import Gaudi application manager (Gaudi Service)
from Configurables import ApplicationMgr

## import DD4hep geometry Gaudi Service and parse compact file
from Configurables import GeoSvc
geoservice = GeoSvc("GeoSvc")
geoservice.detectors = [ './compact/ARC_standalone_o1_v01.xml' ]
ApplicationMgr().ExtSvc += [geoservice]

## import material scan Service and add it to the list of Svc of the ApplicationMgr
from Configurables import MaterialScan_2D_genericAngle
materialservice = MaterialScan_2D_genericAngle("GeoDump")
materialservice.filename = "out_material_scan.root"
materialservice.angleBinning = 0.001
ApplicationMgr().ExtSvc += [materialservice]
```

EDM4hep, an event data model for FCC

- EDM4hep is a generic event data model for future HEP collider experiments [link]
- PODIO library [link] is an abstraction layer that
    - provide IO capabilities, at the moment based on ROOT TTrees
    - generates the Event Data Model (EDM4hep), from a `yaml` input file
- The command `podio-dump` may be used to inspect the data types contained in a file or to read one entry (similar output to Ttree::Show(), see slides later)
- EDM4hep describes more than 100 classes, but new classes may be added by:
    - Declaring the class in a dedicated yaml file [example]
    - Creating the Data Model using Podio CMake macros [example]
- The possibility of adding new EDM4hep classes is addressed in our weekly EDM4hep/key4hep meeting, you are welcome to attend
- See T. Madlener talk for further details about EDM4hep

EDM4hep, an event data model for FCC

- Example showing how to retrieve metadata from simulation output file (C++):

```
auto reader = podio::ROOTFrameReader();

reader.openFile("ALLEGRO_sim_edm4hep.root");

const auto metadata = podio::Frame(reader.readEntry("metadata", 0));

# cellid_encoding = "system:4,cryo:1,type:3,layer:8,module:11"

const auto& cellid_encoding =
metadata.getParameter<std::string>("ArcCollection__CellIDEncoding"));
```

```
dd4hep::DDSegmentation::BitFieldCoder decoder(cellid_encoding);
```

```
for (size_t i = 0; i < reader.getEntries("events"); ++i) {

    auto event = podio::Frame(reader.readNextEntry("events"));

    auto& simCalorimeterHits =
event.get<edm4hep::SimCalorimeterHitCollection>("ECalBarrelEta");

    for (auto simCalorimeterHit : simCalorimeterHits){

      auto cellID = simCalorimeterHit->getCellID();

      int layer = decoder.get(cellID, "layer");

    }

  }
```

EDM4hep, an event data model for FCC

- Example showing how to retrieve metadata from simulation output file (python):

```
import podio

reader = podio.root_io.Reader("test.root")

metadata = reader.get("metadata")[0]

# cellid_encoding = "system:4,cryo:1,type:3,layer:8,module:11"

cellid_encoding =
metadata.get_parameter("ArcCollection__CellIDEncoding")
```

```
import dd4hep as dd4hepModule

from ROOT import dd4hep

decoder = dd4hep.BitFieldCoder( cellid_encoding )

input_cellID = 1234

my_bar_value = decoder.get( input_cellID, "type")
```

Other examples of EDM4hep API can be found [here]

Gaudi functionals

- Gaudi::Functional provides a general building block that is well defined and multithreading friendly. This standardizes the common pattern of getting data out of the TES, working on it, and putting it back in (in a different location).
- The following code shows how to implement an algorithm of type Transform to sum up two input quantities

```
class MySum
 : public TransformAlgorithm<OutputData(const Input1&, const Input2&)> {

     MySum(const std::string& name, ISvcLocator* pSvc)
      : TransformAlgorithm(
           name,
           pSvc, {
              KeyValue("Input1Loc", "Data1"),
              KeyValue("Input2Loc", "Data2") },
           KeyValue("OutputLoc", "Output/Data")) { }

    OutputData operator()(const Input1& in1, const Input2& in2) const override {
        return in1 + in2;
    }
```

- A complete list of `functionals` may be found here
- This tutorial [link] reviews the main steps for the the implementation of a custom Gaudi functional algorithm

Gaudi functionals. A more realistic example

- Reconstruction algorithm for PID based on Cerenkov angle in a RICH detector

```
class RICH_PID
 : public TransformAlgorithm<ParticleID(const Track&, const TrackerHit&)> {

     RICH_PID(const std::string& name, ISvcLocator* pSvc)
      : TransformAlgorithm(
           Name, Psvc, {KeyValue("i1Loc", "tr"), KeyValue("i2Loc", "RICH_hits")},
           KeyValue("oLoc", "Output/Data")) { }

    ParticleID operator()(const Track& tr, , const TrackerHit& RICH_hit) const {
        auto e_point   = estimate_emission_pos_from_track(tr, geoSvc);
        auto h_point   = extract_hit_position(RICH_hit);
        auto mirror_pr = get_mirror_properties(tr, geoSvc);
        auto Cerenkov_angles = solve_raytracing_equation(e_point,h_point,mirror_pr);
        auto ParticleID = local_pattern_recognition_alg(Cerenkov_angles);
        return ParticleID;
    }
```

Gaudi functionals. A word of warning

- Ancillary functions/classes/enums must **always be hidden inside the class**!
- Floating functions/classes/enums pollute the (global) namespace, leading to bugs

```
class RICH_PID,
 : public TransformAlgorithm<ParticleID(const Track&, const TrackerHit&)> {
   private:
     enum Particle_Hypothesis { e, mu, pi, proton, kaon, background };
     struct Cerenkov_angles { ... };
     ParticleID local_pattern_recognition_algorithm(Cerenkov_angles & );

   Public:
     RICH_PID(const std::string& name, ISvcLocator* pSvc)
      : TransformAlgorithm(
           Name, Psvc, {KeyValue("i1Loc", "tr"), KeyValue("i2Loc", "RICH_hits")},
           KeyValue("oLoc", "Output/Data")) { }

    OutputData operator()(const Track& tr, , const TrackerHit& RICH_hit) const;
    }
```

Developing in Key4hep

- What is Key4hep
- Gaudi, framework of FCC software
- Full simulation
  - Detector description.
    - Implementation of a new geometry
  - Physics simulation
    - Custom Sensitive Detector (Action)
    - Tuning secondary production threshold
    - Working with non default physics
- Appendix

What is full simulation

- Full detector simulation is a way of estimating the detector response to some particular physical event (physics simulation) and the later processing of the scored quantities to reconstruct the physical event (reconstruction and analysis)
- It consist of these successive steps:
  1. Generation of primary particles
  2. Simulation of particle transportation and physics (Geant4), and scoring of hits
  3. Digitization, reconstruction
  4. Analysis of reconstructed data, and comparison with MC truth
- Key4hep stack provides the necessary packages to run a full simulation
  - DDG4 (ddsim, part of DD4hep) for physics simulation
  - Gaudi as framework for anything else
  - New packages may be added if needed

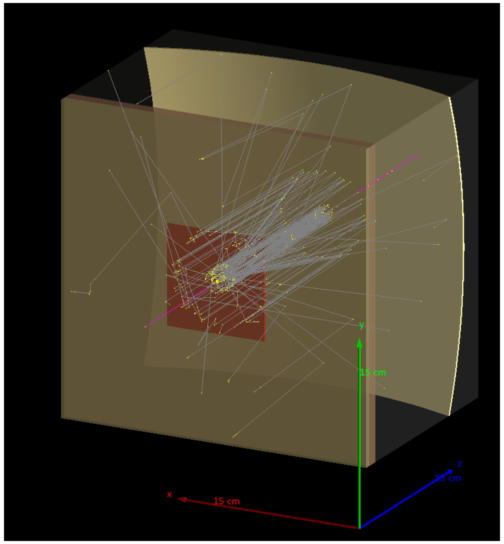
Detector description and physics simulation

The tools for a physics simulation are

- MC generators / particle gun
- DD4hep for detector description
- GEANT4 for physics & transport simulation
  - Hadronic & EM physics
  - Optical physics

Resources:

- Manuals of DD4hep, Geant4
- FCCee Detector Full Sim docs
- Key4hep tutorials
- **Monthly full simulation meetings**



Detector description in DD4hep

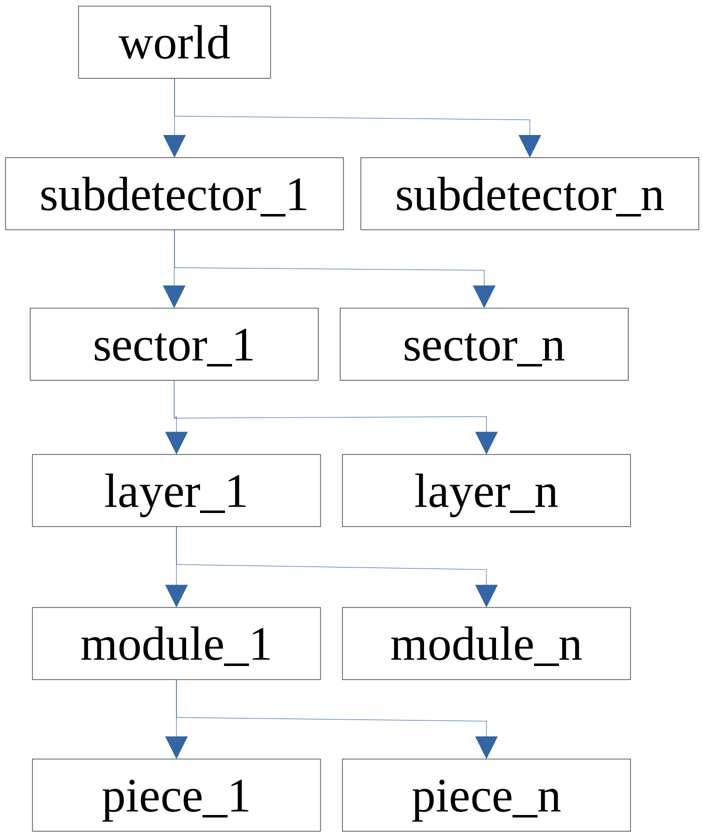
- Detector description consist of a hierarchical **tree of volumes**, and a set of specifications (materials, visualization, readout, fields, etc)

- The detector description is built out of **XML compact files** by DD4hep

- The complexity of a subdetector may be handled by a dedicated C++ function, called **Detector Constructor**, which is called from the XML file by the detector “type” tag. One compact file may call several Detector Constructors

- DD4hep offers a genuine tree called Detector Element tree, which links each **placed volume** to an Detector Element

- **See appendix for a list of good practices**



Detector description in DD4hep

- **Now lets see an example of compact file (XML) [link]**

```xml
<includes>
  <gdmlFile ref="elements.xml"/>
  <gdmlFile ref="materials.xml"/>
</includes>

<display>
  <vis name="vessel_vis"  r="236/256" g="237/256" b="232/256" alp
  <vis name="sensor_vis"  r="255/256" g="0/256"   b="0/256"   alp
  <vis name="no_vis" showDaughters="true" visible="false" />
</display>

<define>
  <constant name="world_side"            value="10*m"        />
  <constant name="world_x"               value="world_side"  />
  <constant name="world_y"               value="world_side"  />
  <constant name="world_z"               value="world_side"  />
  <constant name="tracker_region_zmax"   value="world_side"  />
  <constant name="tracker_region_rmax"   value="world_side"  />
</define>
```

It is allowed to include other XML files, for example containing the list of materials or one single subdetector

Visual attributes may be defined here

Global constants associated to the full detector.
world_x,y,z must be defined

Detector description in DD4hep

- **Now lets see an example of compact file (XML) [link]**
  - Readout structures correspond to the output data structure containing the hits. Only 1 readout per subdetector, but several collections per subdetector are allowed
  - Region defines a secondary production threshold (as in Geant4)

```
    <readouts>
      <readout name="MY_HITS">
        <segmentation type="CartesianGridXY" grid_size_x="1*mm" grid_size_y="1*mm" />
          <id>system:8,x:12:-6,y:24:-6</id>
      </readout>
    </readouts>

      <regions>
        <region name="myregion" eunit="MeV" lunit="mm" cut="0.001" threshold="0.001">
        </region>
        </regions>
```

Detector description in DD4hep

- **Now lets see an example of compact file (XML) [link]**

```xml
  <detectors>
    <detector
        id="1"
        name="MY_FIRST_CUBE"
        type="MYCUBE_T"
        material="CalorimeterMaterial"
        vis="sensor_vis"
        readout="MY_HITS"
        region="myregion"
      >
      <dimensions x="10.*mm" y="10.*mm" z="10.*mm" />
      <!-- /detectors/detector/myproperties -->
      <myproperties
          zposition="-5.*mm"
        >
      </myproperties>
    </detector>


  </detectors>
```

This tag triggers the building of a **subdetector**

Mandatory. Type corresponds to the alias of the C++ Detector Constructor

Optional but very convenient tags

There are reserved names for tags

Custom tags may be defined

Remember to close each tag!

Detector description in DD4hep

- **Lets see an example of Detector Constructor (C++) [link]**

```
static Ref_t createDetector(Detector &desc, xml::Handle_t handle, SensitiveDetector sens)
{
  xml::DetElement detElem = handle;

  // Get detector name and ID from compact file
  std::string detName = detElem.nameStr();
  int detID = detElem.id();
```

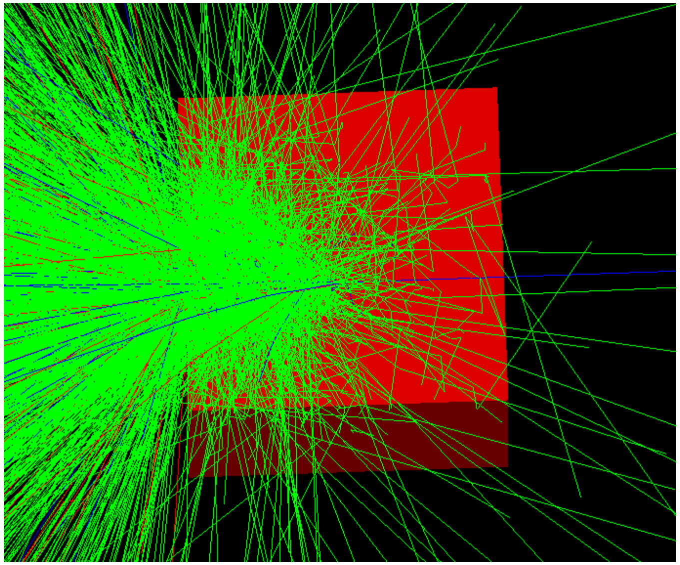
The reserved names for scopes can be accessed by DetElement class methods

```

   // Assign the system ID to our mother volume
   sensorPV.addPhysVolID("system", detID);

   // Associate the silicon Placed Volume to the detector element.
   det.setPlacement(sensorPV);

   return det;
 }

 DECLARE_DETELEMENT(MYCUBE_T, createDetector)
```

Associate the createDetector function to the detector type “MYCUBE_T”

Physics simulation

- DD4hep provides an interface to Geant4 for running physics simulation called **ddsim**
- The main ingredients given to ddsim are:
  - Detector description, in DD4hep format
  - Physics. Default is FTFp-Bertini + EM0 provided by Geant4
  - Actions:
    - Primary particle: from ddsim, G4, external
    - Scoring, by means of a Sensitive Detector Action (recommended)
- A **steering file** may specify these components



100 GeV proton in lead
1cm thick

Physics simulation

- **Lets inspect a simple ddsim steering file [link]**

```python
#steering.py
# ddsim --compactFile ./compact/simple_detector.xml --runType batch --steeringFile steering.py
from DDSim.DD4hepSimulation import DD4hepSimulation
SIM = DD4hepSimulation()

## if there is a user provided SDAction which needs additional parameters these can be passed as a dictionary
SIM.action.mapActions["MY_FIRST_CUBE"] = "mySDaction"

SIM.filter.tracker = ""
SIM.filter.calo    = ""
SIM.filter.filters = {}

# Particle gun settings
SIM.numberOfEvents = 10
SIM.enableGun = True
SIM.gun.energy = "100*GeV"
SIM.gun.particle = "proton"
SIM.gun.direction = "0 0 -1"
SIM.gun.multiplicity = 1
SIM.gun.position = "0 0 1*cm"
```

Custom Sensitive Detector Action to control what is saved as output

Native ddsim particle gun for the generation of primaries. Geant4 gun or GPS are available, but their configuration must be placed in a Geant4 macro file

Custom sensitive detector actions

- Each subdetector have some volumes that are marked to be sensitive, that is, they will register the interaction of particles with them (deposited energy, time, position, particle type, etc) to be later saved into an output file
- DD4hep (ddsim) provides the machinery to run Geant4 simulations and extract information (scoring) by means of Tracker or Calorimeter types of sensitive detectors (SD)
- The native SD types determine the type of output data (tracker hit type, calo hit type), and the action associated with them. There are some default actions for each type [link], but an external custom action may be used instead [link]

SD Type	Output data class	Default SD Action	Other native SD actions
Tracker	edm4hep::SimTrackerHit	Geant4TrackerWeightedAction	Geant4TrackerAction, Geant4OpticalTrackerAction
Calorimeter	edm4hep::SimCalorimeterHit, edm4hep::CaloHitContribution	Geant4ScintillatorCalorimeterAction	Geant4CalorimeterAction, Geant4OpticalCalorimeterAction

Custom sensitive detector actions

- We can run a simulation using ddsim using the previous steering file

```
ddsim --compactFile ./compact/simple_detector.xml --runType batch --
steeringFile steering.py --outputFile mySimulation.root
```

- Using podio-dump command to check what is inside the ddsim output file

```
datamodel model definitions stored in this file: edm4hep

Frame categories in this file:
Name                    Entries
-------------------------------
runs                    1
metadata                1
events                  10

################################### events: 0
####################################
Collections:
Name         ValueType                Size   ID
-----------  ----------------------  ------  --------
EventHeader  edm4hep::EventHeader         1  d793ab91
MCParticles  edm4hep::MCParticle        288  a1cba250
MY_HITS      edm4hep::SimTrackerHit    1238  512bf904
```

metadata → May contain the segmentation encoding

MCParticles → Primary particles

MY_HITS → Hits in the sensitive volume

Custom sensitive detector actions

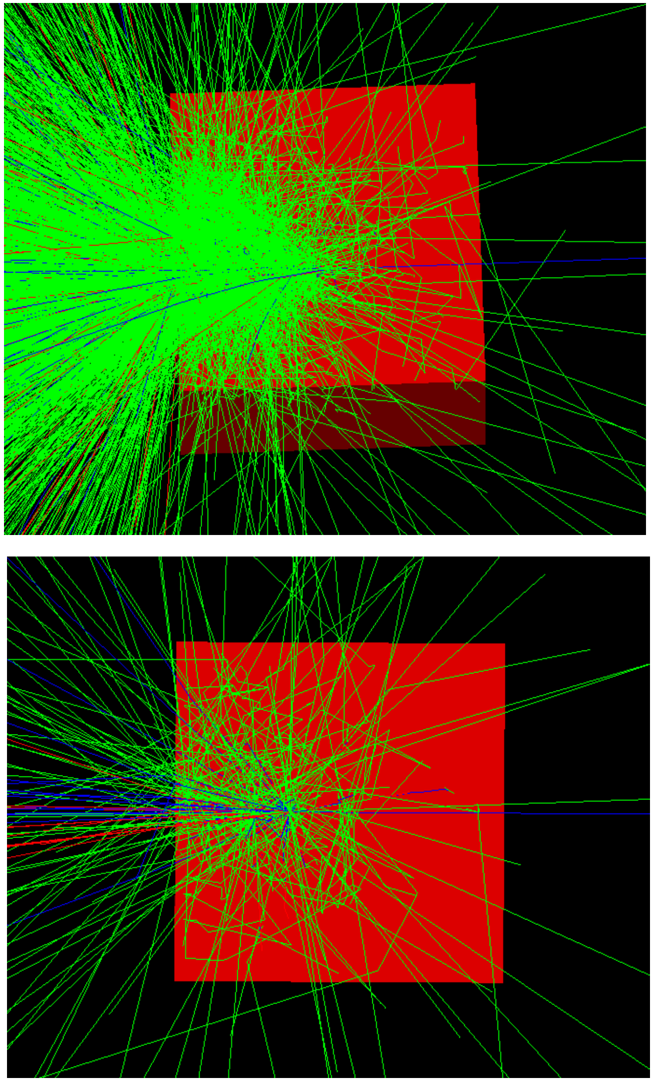
- We can change the SD type, from tracker to calorimeter, and run again the simulation using the same command
- Using podio-dump command to check what is inside the ddsim output file:

```
datamodel model definitions stored in this file: edm4hep

Frame categories in this file:
Name                     Entries
-------------------------------
runs                     1
metadata                 1
events                   10

################################### events: 0
####################################
Collections:
Name                      ValueType                         Size  ID
--------------------  ----------------------------  ------  --------
EventHeader               edm4hep::EventHeader                 1  d793ab91
MCParticles               edm4hep::MCParticle                288  a1cba250
MY_HITS                   edm4hep::SimCalorimeterHit          52  512bf904
MY_HITSContributions      edm4hep::CaloHitContribution      2049  7f8794a2
```

One readout, two output collections

Understanding the production threshold

- A fraction of the showers are EM showers, made up by gamma/e-/+
- Some EM processes have an infrared divergence at secondaries production (the lower the energy, the more secondaries are created)
- A production threshold is used to prevent the production of EM secondaries below that limit. This threshold can be expressed in distance (range) or energy
- **Minimal threshold/range should be at least half the smallest size of the sensitive volume**



Different thresholds for one proton at 100 GeV in 1cm of lead

How to use non default physics

- Geant4 provides several builtin physics lists (PL) [link]
- Builtin physics list may be defined in the ddsim steering file
- Similarly as Geant4, new physics may be added on top of the builtin PL

```
## The name of the Geant4 Physics list.
SIM.physics.list = "FTFP_BERT"

def setupCerenkov(kernel):
        from DDG4 import PhysicsList

        seq = kernel.physicsList()
        cerenkov = PhysicsList(kernel, "Geant4CerenkovPhysics/CerenkovPhys")
        cerenkov.MaxNumPhotonsPerStep = 10
        cerenkov.MaxBetaChangePerStep = 10.0
        cerenkov.TrackSecondariesFirst = False
        cerenkov.VerboseLevel = 0
        cerenkov.enableUI()
        seq.adopt(cerenkov)
        ph = PhysicsList(kernel, "Geant4OpticalPhotonPhysics/OpticalGammaPhys")
        ph.addParticleConstructor("G4OpticalPhoton")
        ph.VerboseLevel = 0
        ph.BoundaryInvokeSD = True
        ph.enableUI()
        seq.adopt(ph)
        return None

SIM.physics.setupUserPhysics(setupCerenkov)
```

Builtin Physics List

Encapsulate addition of custom physics on top of the kernel PL

New Physics process (Cerenkov process)

New Particle (optical photon)

Register custom physics

Summary

- It is possible to use key4hep stack and work with local installations while developing new features
- EDM4hep is an extensive set of classes used for full simulation studies, but it is possible to add new classes if needed
- Podio API may be used to inspect and read files
- Follow typical C++ guidelines when developing Gaudi components
- When developing components of DD4hep,
  - Build the geometry as a hierarchical tree of volumes
  - Reuse ddsim Sensitive Detector types, and reimplent the process function as a Sensitive Detector Action plugin
  - Understand the subdetector when defining its region and limits
- Check the appendix for some notes about HTCondor and EOS

Thank you for your time

Developing in Key4hep

- What is Key4hep
- Gaudi, framework of FCC software
- Full simulation
- Appendix
    - How to use HTCondor & EOS for heavy duty jobs
    - Some recommendations when implementing the geometry of a detector in DD4hep/Geant4

alvaro.tolosa.delgado@cern.ch

General recommendations when using batch. HTCondor & EOS

Alvaro Tolosa-Delgado

Personal notes

Nov. 21th, 2023

Using batch. HTCondor and EOS

- The two main ingredients for batching are
  - Executable files (bash, python) which encapsulate each task
  - Condor job descriptor file, which launches the execution of the different tasks
- At CERN, job submission is done from lxplus.cern.es user interface
- The job copy the user token, so they will have the same privileges
- It is recommended to leave the transfer of output files as the last step
- **Avoid using AFS for large/numerous files**
- **Use EOS remote access for heavy duties (see next slide)**

Using batch. Example of executable

```
#!/bin/bash

source /cvmfs/sw.hsf.org/key4hep/setup.sh

ddsim  --compactFile /cvmfs/sw.hsf.org/key4hep/releases/2023-11-23/x86_64-almalinux9-gcc11.3.1-opt/k4geo/0.19-qfldo5/share/k4geo/FCCee/CLD/compact/CLD_o2_v05/CLD_o2_v05.xml    --outputFile  SIM_CLD_o2_v05_mu-_20_deg_50_GeV_1000_evts.root    --steeringFile /afs/cern.ch/user/a/aaaaaaa/Public/ddsim_steering_CLD.py     --random.seed  1 --numberOfEvents 1000    --enableGun     --gun.particle mu-      --gun.energy 50*GeV   --gun.distribution uniform      --gun.thetaMin 20*deg   --gun.thetaMax 20*deg   --random.enableEventSeed

# Setup EOS entry point

export EOS_MGM_URL=root://eosuser.cern.ch

# lets try non verbose copy of the file...

xrdcp -v  --debug 2 --retry 5  --nopbar SIM_CLD_o2_v05_mu-_20_deg_50_GeV_1000_evts.root root://eosuser.cern.ch//eos/user/a/aaaaaaa/condor/comparison_cld_o2_cld_o3//SIM_CLD_o2_v05_mu-_20_deg_50_GeV_1000_evts.root
```

Using batch. Example of condor job descriptor

```
executable     = $(filename)
output = output.$(ClusterId).$(ProcId).out
error = error.$(ClusterId).$(ProcId).err
log = log.$(ClusterId).log
should_transfer_files = YES
transfer_input_files = /afs/cern.ch/user/a/aaaaaa/Public/ddsim_steering_CLD.py
transfer_output_files = ""
+JobFlavour = "microcentury"
+AccountingGroup = "group_u_FCC.local_gen"
queue filename matching files *.sh
```

Tips:

- Use the proper **AccountingGroup** (default is none)
- Pick the proper **JobFlavour** (and CPU & memory)
- Use the syntax **queue xxxxx** to execute all jobs, jobs will be **executed faster**

Using batch. EOS

EOS

EOS can be mounted in the local machine or accessed via `eos` command

- When doing the following, remote EOS is accessed:

```
eos root://eosuser.cern.ch ls /eos/user/a/antonio
```

- When doing the following, local synchronized copy of EOS is accessed. The synchronization may be spoiled if used heavily. Previous option should be preferred

```
ls /eos/user/a/antonio
```

General recommendations for optimal geometry implementation

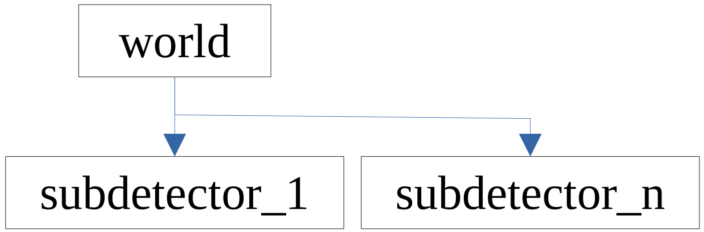
Alvaro Tolosa-Delgado

Personal notes

Nov. 21th, 2023

General ideas

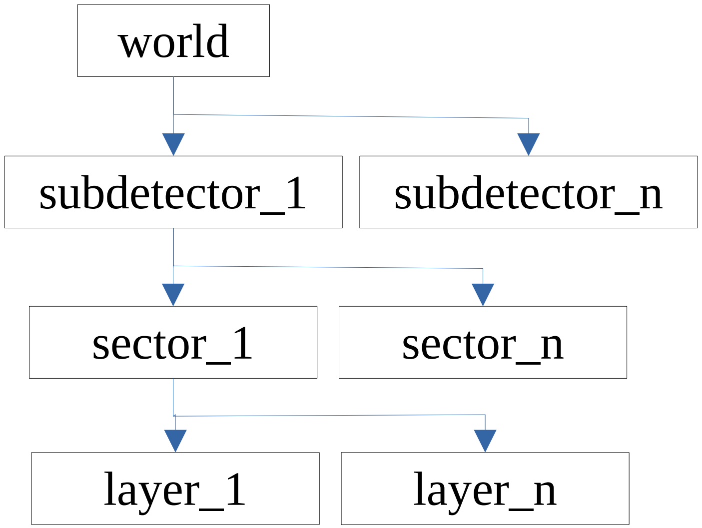
Full geometry tree



- Each box corresponds to a volume object, and each arrow corresponds to a geometrical transformation (position+rotation in the local coordinate system of the mother volume)
- Each subdetector is expected to be contained in an envelope volume. This volume may be made of air as the world.
- Endcaps and barrel of a subdetector may be considered as different subdetectors (and may have different C++ detector constructors). They can still share the same readout.

General ideas

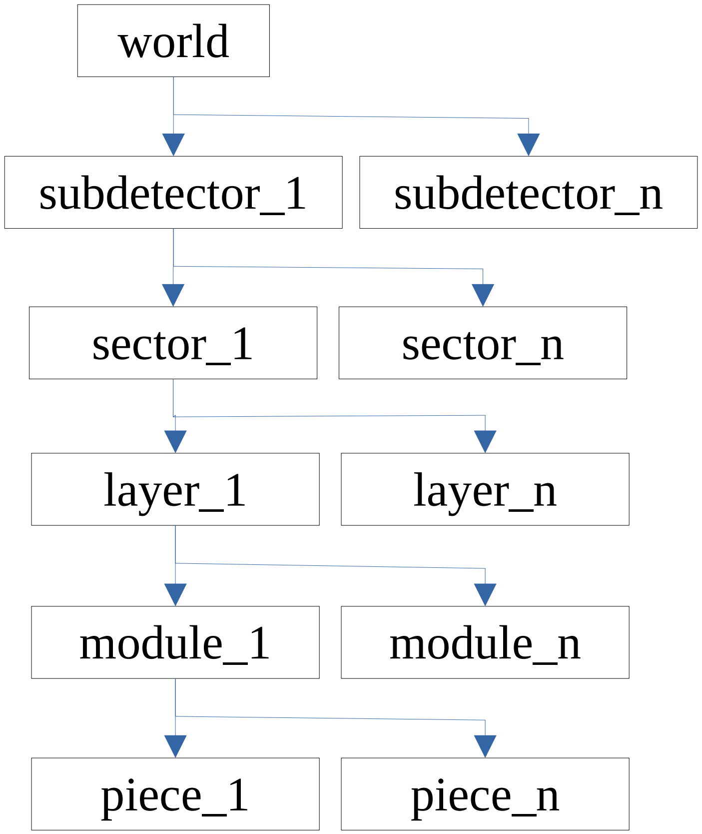
Full geometry tree



- Each little piece of the detector can be grouped into bundles according to the symmetry.
- For example, if there is a symmetry around Z-axis (phi), intermediate envelopes (e.g., named sectors) which will contain all the daughter volumes, may be defined and placed many times around Z-axis.
- If there is radial symmetry (that is, the same layer placed several times), an intermediate envelope volume which contains a layer (and all the sub-volumes) can be created and placed as many times as needed

General ideas

Full geometry tree



- Deeper grouping may be done if symmetry allows it
- Physical volume ID is assigned to all daughter volumes. If volume “sector_1” has “Phi” bitfield “1”, all daughters will have that bitfield set to “1”.
- Tip: create volumes and shapes outside the loops that will iterate over phi/theta/R.
- Liking Placed Volumes to DD4hep detector elements may be done with care

General considerations

- Please, keep the number of daughter volumes below o(1000). Use intermediate envelope volumes to reduce this number. This will speed up geometry construction, navigation and reduce memory consumption.

- Please, do not abuse the assembly-type volume. During translation of geometry, all daughters are placed directly into the mother volume, assembly does not exist during the simulation.

- Please think twice before using shapes such as Boolean operations or tessellated solids, they will impact performance