DD4HEP for EIC simulations

Barak Schmookler

Ryan Milton

CALIFORNIA EIC CONSORTIUM

UC RIVERSIDE

Outline

- Setting up the container
- Running default detector setup
- Modifying the default setup
- Simulating single (or a few) detectors in DD4HEP
  - Simple geometry – no compilation
  - Writing a new detector class
- Examples of analyzing the output
- Juggler calorimeter algorithms

Use cases – from a user's perspective

1. Run the default full detector simulation – both with and without including reconstruction algorithms – and perform analysis on the output ROOT file. Can use either single particle or full event generator as input.
2. Make a specific modification to the default detector setup or reconstruction library, and then run the full simulation as above.
3. Define a single detector, throw particles at it, apply some reconstruction algorithms, and then analyze the results.
4. Define part of the full detector system (e.g. the forward endcap calorimeters) and run simulation as above.

In all cases, we need to start enter the software container

- Prerequisites: Git, Singularity, CVMFS (?). See useful instructions from ECCE on installing the last two: https://github.com/ECCE-EIC/Singularity
- Then do the following:

```
mkdir eic
cd eic
curl https://eicweb.phy.anl.gov/containers/eic_container/-/raw/master/install.sh | bash
./eic-shell
```

Running the container

- After running the commands on the last slide, you will be in the EIC container. You now have access to certain libraries and some environmental variables are defined.

```
[baraks@eic0101 eic]$ ./eic-shell
jug_xl> baraks@eic0101:/gpfs02/eic/baraks/epic/eic$ echo $EIC_SHELL_PREFIX
/gpfs02/eic/baraks/epic/eic/local
jug_xl> baraks@eic0101:/gpfs02/eic/baraks/epic/eic$ echo $LD_LIBRARY_PATH
/gpfs02/eic/baraks/epic/eic/local/lib:/lib/x86_64-linux-gnu:/usr/local/lib:/usr/local/lib64
jug_xl> baraks@eic0101:/gpfs02/eic/baraks/epic/eic$ ls /opt/detector
athena-nightly  calibrations  ecce-nightly  epic-nightly  fieldmaps  lib  setup.sh  share
jug_xl> baraks@eic0101:/gpfs02/eic/baraks/epic/eic$
```

Running the default setup

- To run a simulation with the default setup, we need to do the following:

source /opt/detector/setup.sh

```
jug_xl> baraks@eic0101:/gpfs02/eic/baraks/epic/eic$ source /opt/detector/setup.sh
nightly> baraks@eic0101:/gpfs02/eic/baraks/epic/eic$
```

- This will set a bunch of variables for us, as we'll see on the next slide.

Viewing default setup script

```
#!/bin/sh
export DETECTOR=athena
export DETECTOR_PATH=/opt/detector/athena-nightly/share/athena
export DETECTOR_CONFIG=athena
export DETECTOR_VERSION=master
export BEAMLINE_CONFIG=ip6
export BEAMLINE_CONFIG_VERSION=master
## note: we will phase out the JUGGLER_* flavor of variables in the future
export JUGGLER_DETECTOR=$DETECTOR
export JUGGLER_DETECTOR_CONFIG=$DETECTOR_CONFIG
export JUGGLER_DETECTOR_VERSION=$DETECTOR_VERSION
export JUGGLER_DETECTOR_PATH=$DETECTOR_PATH
export JUGGLER_BEAMLINE_CONFIG=$BEAMLINE_CONFIG
export JUGGLER_BEAMLINE_CONFIG_VERSION=$BEAMLINE_CONFIG_VERSION
export JUGGLER_INSTALL_PREFIX=/usr/local

## Export detector libraries
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/detector/athena-nightly/lib

## modify PS1 for this detector version
export PS1="${PS1:-}"
export PS1="nightly${PS1_SIGIL}>${PS1#*>}"
unset branch
```

We can now run the simulation

- Running 100 single muons:

npsim --compactFile $DETECTOR_PATH/athena.xml --enableGun --gun.distribution uniform --numberOfEvents 100 --outputFile output.edm4hep.root

- Running 100 single pions:

npsim --compactFile $DETECTOR_PATH/athena.xml --enableGun --gun.distribution uniform --gun.particle pi+ --numberOfEvents 100 --outputFile output.edm4hep.root

- Using a HepMC3 file as input:

npsim --compactFile $DETECTOR_PATH/athena.xml --numberOfEvents 25 --inputFiles input.hepmc --outputFile output.edm4hep.root

The default detector is currently ATHENA…but we can also run EPIC

source /opt/detector/epic-nightly/setup.sh

npsim --compactFile $DETECTOR_PATH/epic.xml --enableGun --gun.distribution uniform --numberOfEvents 100 --outputFile output.edm4hep.root

The default EPIC setup script has the same structure as the ATHENA one

```
#!/bin/sh
export DETECTOR=epic
export DETECTOR_PATH=/opt/detector/epic-nightly/share/epic
export DETECTOR_CONFIG=epic
export DETECTOR_VERSION=main
export BEAMLINE_CONFIG=ip6
export BEAMLINE_CONFIG_VERSION=master
## note: we will phase out the JUGGLER_* flavor of variables in the future
export JUGGLER_DETECTOR=$DETECTOR
export JUGGLER_DETECTOR_CONFIG=$DETECTOR_CONFIG
export JUGGLER_DETECTOR_VERSION=$DETECTOR_VERSION
export JUGGLER_DETECTOR_PATH=$DETECTOR_PATH
export JUGGLER_BEAMLINE_CONFIG=$BEAMLINE_CONFIG
export JUGGLER_BEAMLINE_CONFIG_VERSION=$BEAMLINE_CONFIG_VERSION
export JUGGLER_INSTALL_PREFIX=/usr/local

## Export detector libraries
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/detector/epic-nightly/lib

## modify PS1 for this detector version
export PS1="${PS1:-}"
export PS1="nightly${PS1_SIGIL}>${PS1#*>}"
unset branch
```

Aside – three ways to run single particle simulation

1. Directly with *npsim* as above.
2. Using the general particle source (gps). See here for an example: https://github.com/rymilton/eic_endcap_insert/blob/main/run_sim_gps
3. Writing a HepMC3 file 'by hand'. Here is an example: https://github.com/rymilton/eic_endcap_insert/blob/main/hepmc_generation/gen_particles.cxx

Looking at the DD4Hep output

```
[baraks@eic0101 eic]$ root -l output.edm4hep.root
root [0]
```

```
root [1] events->GetEntries()
(long long) 100
root [2] events->Show(0)
======> EVENT:0
 MCParticles      = (vector<edm4hep::MCParticleData>*)0x296cc40
 MCParticles.PDG = 13, 11, 11
 MCParticles.generatorStatus = 1, 0, 0
 MCParticles.simulatorStatus = 33554432, 1493172224, 1493172224
 MCParticles.charge = -1.000000, -1.000000, -1.000000
 MCParticles.time = 0.000000, 4.619186, 6.382085
 MCParticles.mass = 0.105658, 0.000510999, 0.000510999
 MCParticles.vertex.x = 0, -305.583, -129.848
 MCParticles.vertex.y = 0, 931.397, 965.269
 MCParticles.vertex.z = 0, 977.391, 1278.02
 MCParticles.endpoint.x = -16891.8, -306.51, -128.855
 MCParticles.endpoint.y = 30000, 931.703, 965.276
 MCParticles.endpoint.z = 40100.9, 978.793, 1278.55
 MCParticles.momentum.x = -1.780748, -0.001310, 0.001416
 MCParticles.momentum.y = 6.861804, 0.000338, -0.000465
 MCParticles.momentum.z = 7.052984, 0.000849, 0.000131
 MCParticles.momentumAtEndpoint.x = -2.427296, -0.000000, -0.000000
 MCParticles.momentumAtEndpoint.y = 4.237793, 0.000000, -0.000000
 MCParticles.momentumAtEndpoint.z = 5.728499, -0.000000, -0.000000
 MCParticles.spin.x = 0.000000, 0.000000, 0.000000
 MCParticles.spin.y = 0.000000, 0.000000, 0.000000
 MCParticles.spin.z = 0.000000, 0.000000, 0.000000
```

continues ↓

Looking at the DD4Hep output

```
EcalEndcapPHits = (vector<edm4hep::SimCalorimeterHitData>*)0x298a760
EcalEndcapPHits.cellID = 1897078630, 1897144166, 1081869158
EcalEndcapPHits.energy = 0.005235, 0.001490, 0.000727
EcalEndcapPHits.position.x = 1605.000000, 1605.000000, -20.000000
EcalEndcapPHits.position.y = 1680.000000, 1705.000000, 355.000000
EcalEndcapPHits.position.z = 3585.000000, 3585.000000, 3585.000000
EcalEndcapPHits.contributions_begin = 0, 19, 32
EcalEndcapPHits.contributions_end = 19, 32, 39
EcalEndcapPHits#0 = (vector<podio::ObjectID>*)0x29799b0
EcalEndcapPHits#0.index = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
EcalEndcapPHits#0.collectionID = 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
EcalEndcapPHitsContributions = (vector<edm4hep::CaloHitContributionData>*)0x2987040
EcalEndcapPHitsContributions.PDG = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
EcalEndcapPHitsContributions.energy = 0.000395, 0.000012, 0.000171, 0.000212, 0.000116, 0.000480, 0.00055
51, 0.000795, 0.000369, 0.000141, 0.000173, 0.000153, 0.000130, 0.000136, 0.000171, 0.000002, 0.000070
```

continues

Running the DD4Hep output through Juggler

- To run the output through the default physics reconstruction:

export JUGGLER_SIM_FILE=output.edm4hep.root JUGGLER_REC_FILE=rec_output.edm4hep.root JUGGLER_N_EVENTS=10

gaudirun.py /opt/benchmarks/physics_benchmarks/options/reconstruction.py

- If we just want to focus on the calorimeters:

export JUGGLER_SIM_FILE=output.edm4hep.root JUGGLER_REC_FILE=rec_cal_output.edm4hep.root JUGGLER_N_EVENTS=10

gaudirun.py /opt/benchmarks/reconstruction_benchmarks/benchmarks/clustering/options/full_cal_reco.py

Analyzing the Juggler output

- The reconstructed output has a similar format to the DD4HEP (Geant4) output. Since the branches are just arrays, there are a few ways we can analyze the output.
  1. Using TTree::Draw() methods.
  2. ROOT-based in a loop, using TLeaf methods. See here for an example: https://github.com/bschmookler/athena_ana/blob/main/Analysis_examples/athena_particles.C
  3. ROOT-based in a loop, using TTreeReaderArray. See here for an example: https://github.com/bschmookler/athena_ana/blob/main/Analysis_examples/KinematicReco.C
  4. Using *uproot* based array analysis. See here for an example: https://github.com/bschmookler/athena_ana/blob/main/Analysis_examples/mcparticles_check.ipynb
  5. Data-frame type analysis?

Using local builds of detector, ip6, and Juggler

- You may want to repeat the above simulations, after making a small change to the default detector setup or reconstruction.
- You can do this as follows.

```
git clone https://eicweb.phy.anl.gov/EIC/detectors/athena.git
git clone https://eicweb.phy.anl.gov/EIC/detectors/ip6.git
ln -s ../ip6/ip6 athena/ip6
git clone https://eicweb.phy.anl.gov/EIC/juggler.git
```

Building the local versions

To build the ATHENA detector:

```
cd athena
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$ATHENA_PREFIX
make
make install
```

$EIC_SHELL_PREFIX

For the reconstruction software:

```
cd juggler
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$ATHENA_PREFIX
make
make install
```

$EIC_SHELL_PREFIX

To build the beamline:

```
cd ip6
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$ATHENA_PREFIX
make
make install
```

$EIC_SHELL_PREFIX

If you also need to build the data model (to add new variables):

```
git clone https://eicweb.phy.anl.gov/EIC/eicd.git
cd eicd
mkdir build
cd build
cmake ../. –DCMAKE_INSTALL_PREFIX=$EIC_SHELL_PREFIX
–DBUILD_DATA_MODEL=ON
–DEICD_DIR:PATH=$EIC_SHELL_PREFIX/lib/EICD
make
make install
```

How to use your local libraries

- Previously we ran **source /opt/detector/setup.sh** to use the default libraries. (Although note that your local libraries are in the $LD_LIBRARY_PATH. So, if you source the default setup when you have local copies compiled, weird errors can occur.)
- Now we want to use our own libraries. We can source this file: https://github.com/bschmookler/athena_ana/blob/main/mysetup.sh

How does local setup script look

```
#!/bin/sh
export S3_ACCESS_KEY=[redacted]
export S3_SECRET_KEY=

## For using 'official' setup, uncomment line below and comment all following lines
#source /opt/detector/setup.sh

export DETECTOR=athena
## If you use the 'share' version, you will need to make a softlink to ip6
#export DETECTOR_PATH=${ATHENA_PREFIX}/share/athena
## May want to use source directory instead
export DETECTOR_PATH=${ATHENA_PREFIX}/../athena
```

$EIC_SHELL_PREFIX

```
export DETECTOR_VERSION=master
export BEAMLINE_CONFIG=ip6
export BEAMLINE_CONFIG_VERSION=master
## note: we will phase out the JUGGLER_* flavor of variables in the future
export JUGGLER_DETECTOR=$DETECTOR
export JUGGLER_DETECTOR_VERSION=$DETECTOR_VERSION
export JUGGLER_DETECTOR_PATH=$DETECTOR_PATH
export JUGGLER_BEAMLINE_CONFIG=$BEAMLINE_CONFIG
export JUGGLER_BEAMLINE_CONFIG_VERSION=$BEAMLINE_CONFIG_VERSION
export JUGGLER_INSTALL_PREFIX=$ATHENA_PREFIX

## Export detector libraries
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${ATHENA_PREFIX}/lib

## modify PS1 for this detector version
export PS1="${PS1:-}"
export PS1="local${PS1_SIGIL}>${PS1#*>}"
unset branch
```

Again, there are some new instructions for doing this for the EPIC detector which seem to be very similar to the ATHENA case

```
git clone https://github.com/eic/epic.git
git clone https://github.com/eic/ip6.git
ln -s ../ip6/ip6 epic/ip6
```

```
cmake -B build -S . -DCMAKE_INSTALL_PREFIX=install -DEPIC_ECCE_LEGACY_COMPAT=OFF
cmake --build build
cmake --install build
```

```
source install/setup.sh
```

Standalone detector example – detector geometry defined in .xml file

```xml
<lccdd>
  <comment>
   //////////////////////////////////////////////////////////
   // HCAL
   // Fe + Scintillator (Fe/Sci) sandwich sampling calorimeter
   //////////////////////////////////////////////////////////
  </comment>
 <define>
   <constant name="Pi" value="3.14159265359"/>
   <constant name="world_side" value="30*m"/>
   <constant name="world_x" value="world_side"/>
   <constant name="world_y" value="world_side"/>
   <constant name="world_z" value="100*m"/>
     <constant name="tracker_region_zmax" value="10*m"/>
   <constant name="tracker_region_rmax" value="1*m"/>

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

  <detectors>

   <detector id="1" name="HCAL" type="ffi_ZDC_Sampling" readout="HCALHits">
     <position x="0*m" y="0*m" z="350*cm"/>
     <rotation x="0" y="0" z="0"/>
     <dimensions x="72*cm" y="72*cm" z="132*cm"/>
     <layer repeat="60">
       <slice name="Absorber_slice"  material="Steel235"        thickness="1.7*cm"  vis="AnlGray"/>
       <slice name="Scint_slice"     material="Polystyrene" thickness="0.5*cm" vis="AnlOrange" sensitive="true"/>
     </layer>
   </detector>
```

This is the detector class. It already exists in the software container. So, we can use it directly.

```xml
  </detectors>

  <readouts>
    <readout name="HCALHits">
      <segmentation type="CartesianGridXY" grid_size_x="3.0 * cm" grid_size_y="3.0 * cm"/>
      <id>system:8,layer:12,slice:12,x:32:-16,y:-16</id>
    </readout>


  </readouts>
</lccdd>
```

https://github.com/bschmookler/athena_ana/tree/main/Detector_examples/samplinghcal

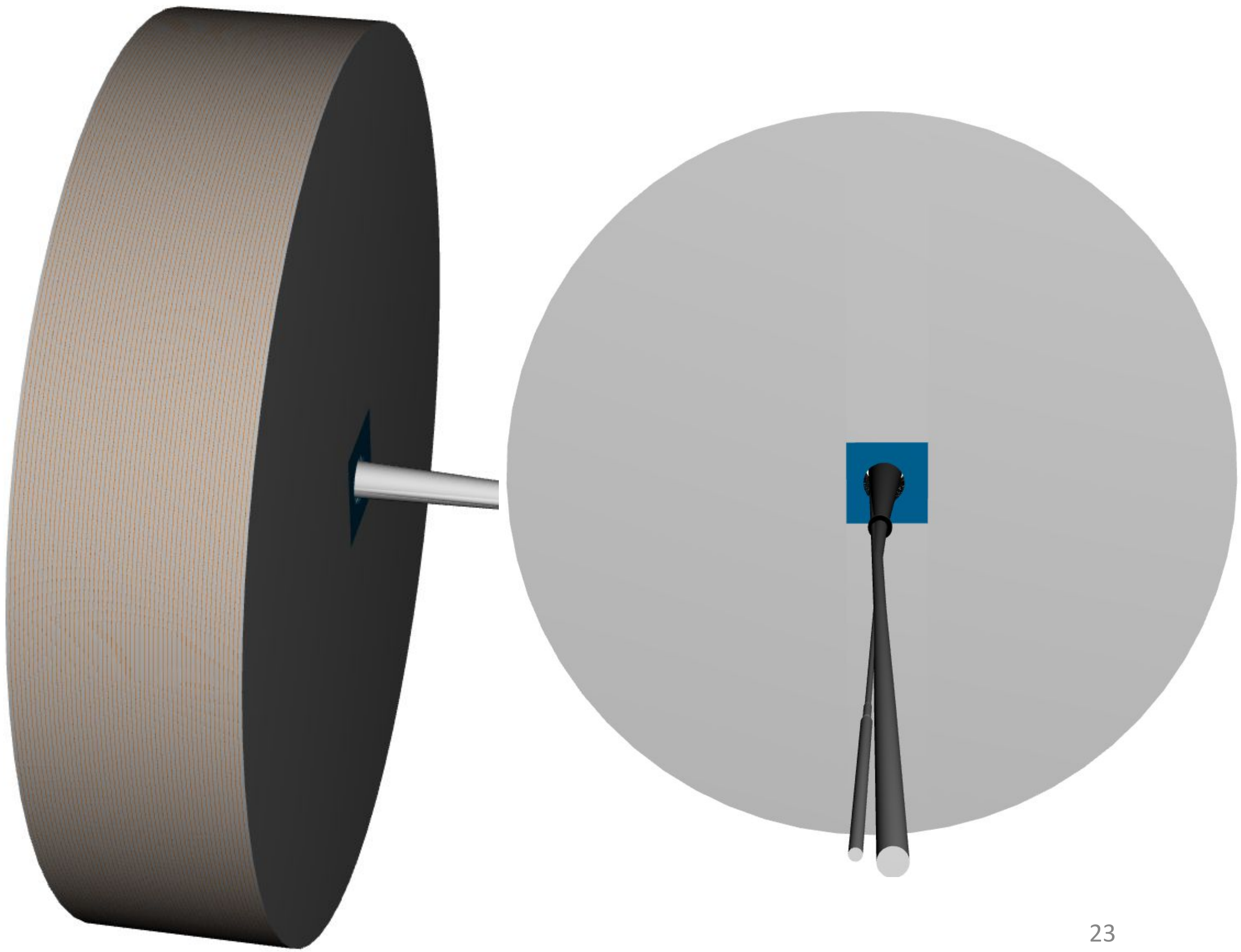
Running the simple standalone simulation

- To run a single-particle simulation with the above detector, we need to run **source /opt/detector/setup.sh** to use the default libraries. (Note that if you have to compile a new detector class, it may still be okay to source the default setup script, as your local library directory is in the $LD_LIBRARY_PATH. But it can be problematic if you are trying to replace a default library with a local one.)
- To the run some events (just DD4HEP/Geant4 part):

```
npsim --runType run --enableG4GPS --macroFile gps.mac --compactFile samplinghcal.xml --outputFile sim_out.root
```

Making a new detector class: ATHENA pEndcap HCal

- Defined a new class for the ATHENA HCal that includes room for insert
- Needs:
  - 50 Layers of steel/Sc
  - 1 back layer of steel
  - Roughly cylindrical with 10x10 cm readouts
  - 60x60 cm hole for insert



Implementation snippet: XML

https://github.com/rymilton/eic_endcap_insert/blob/main/compact/hcal_ATHENA.xml

```xml
<detectors>
  <comment>
    ------------------------------------------
    Forward (Positive Z) Endcap Hadronic Calorimeter
    ------------------------------------------
  </comment>
  <detector
    id="HCalEndcapP_ID"
    name="HcalEndcapP"
    type="pEndcap"
    readout="HcalEndcapPHits">
    <position x="0" y="0" z="pHCal_zmin"/>
    <comment> HCal has 50 layers + 1 layer of just absorber </comment>
    <dimensions
      z="pHCal_LayerThickness*(pHCal_ATHENA_numSteelScLayers) +
        pHCal_AbsorberThickness"
      rmin="pHCal_rmin"
      rmax="pHCal_rmax"
    />
```

```xml
<readouts>
  <readout name="HcalEndcapPHits">
    <segmentation type="CartesianGridXY" grid_size_x="100 * mm" grid_size_y="100 * mm"/>
    <id>system:8,barrel:3,module:4,layer:8,slice:5,x:32:-16,y:-16</id>
  </readout>
</readouts>
```

Needs in XML:

- Unique ID number
- Unique name
- Type of detector
    - Links to cpp source file
- Readout name
    - How the hit info will be recorded
    - Readouts linked to materials set as sensitive (next slide)

XML Detector description

```
  <layer repeat="pHCal_ATHENA_numSteelScLayers" thickness = "pHCal_LayerThickness">
    <slice material="Steel235" thickness="pHCal_AbsorberThickness" vis="AnlLight_Gray"/>
    <slice material="PlasticScint126" sensitive="yes" thickness="pHCal_ScintillatorThickness" vis="AnlOrange"/>
    <slice material="Air" thickness="pHCal_AirThickness" vis="Invisible"/>
  </layer>
  <comment> Final layer of absorber </comment>
  <layer repeat="1" thickness = "pHCal_AbsorberThickness">
    <slice material="Steel235" thickness="pHCal_AbsorberThickness" vis="AnlLight_Gray"/>
  </layer>
  <insert>
    <position x="pHCalInsert_xpos" y="0" z = "0" />
    <dimensions x="pHCalInsert_xtotal" y="pHCalInsert_ytotal"/>
  </insert>
</detector>
```

- Define “XML handles” that will be grabbed in cpp code
- Can put any info here about dimensions, positions, materials, etc.
  - Essentially what your detector is composed of
- Mark anything to be read out as “sensitive”
  - See scintillator slice in third line in above image

C++ detector implementation

https://github.com/rymilton/eic_endcap_insert/blob/main/src/pEndcap_geo.cpp

```
static Ref_t createDetector(Detector& desc, xml_h e, SensitiveDetector sens)
{
  xml_det_t  x_det      = e;
  string     detName    = x_det.nameStr();
  int detID             = x_det.id();

  xml_dim_t  dim        = x_det.dimensions();
  double     rmin       = dim.rmin();
  double     rmax       = dim.rmax();
  double     length     = dim.z();

  xml_dim_t  pos        = x_det.position();
  const double     x    = pos.x();
  double     z          = pos.z();
```

```

  return det;
}
DECLARE_DETELEMENT(pEndcap, createDetector)
```

- createDetector takes in the XML info
- Can then pull the info through the XML handles
- To link a cpp file with your XML file add DECLARE_DETELEMENT at bottom of file
  - Takes in the type defined in XML file

Similarities with Geant4

```
// Getting insert dimensions
const xml::Component &insert_xml = x_det.child(_Unicode(insert));
xml_dim_t  insert_dim        = insert_xml.dimensions();
xml_dim_t  insert_local_pos  = insert_xml.position();

// Defining envelope (mother volume)
Tube envelope(rmin, rmax, length / 2.0);
Box insert(insert_dim.x() / 2., insert_dim.y() / 2., length / 2.);
SubtractionSolid envelope_with_inserthole(
  envelope,
  insert,
  Position(insert_local_pos.x(), insert_local_pos.y(), 0.)
);
Material   Vacuum     = desc.material("Vacuum");
Volume envelopeVol(detName+"_envelope", envelope_with_inserthole, Vacuum);
```

```
for(xml_coll_t c(x_det,_U(layer)); c; ++c)
{
  xml_comp_t x_layer = c;
  int repeat = x_layer.repeat();
  double layer_thickness = x_layer.thickness();

  // Loop over repeat
  for(int i = 0; i < repeat; i++) {
```

Looping over layers

- Like Geant4, all volumes need:
  - Envelope (mother volume)
  - Shape/geometry (G4VSolid)
  - Volume (G4LogicalVolume)
  - PlacedVolume (G4PVPlacement)
- Most Geant4 classes are in DD4hep, e.g.
  - Box = G4Box
  - SubtractionSolid = G4SubtractionSolid
  - But no G4PVParameterised or G4VReplica
    - Loop over layers instead

Physical placements and active areas

```
PlacedVolume pv;
```

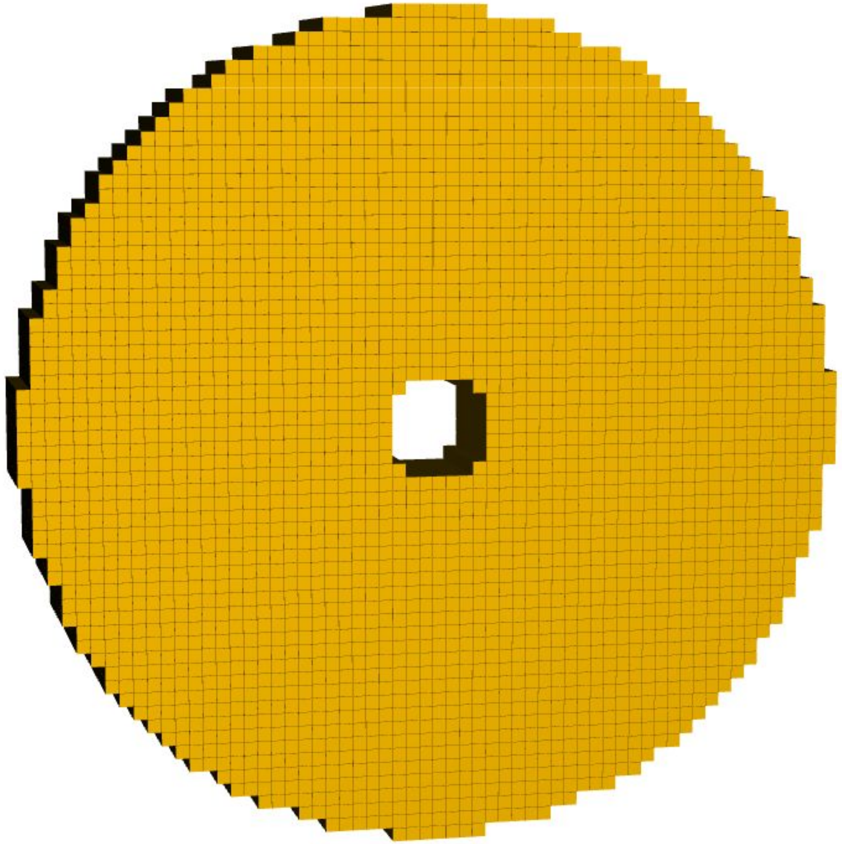
```
pv = envelopeVol.placeVolume(
  layer_vol,
  Transform3D(
    RotationZYX(0, 0, 0),
    Position(0., 0. , zlayer - pos.z() - length/2. + layer_thickness/2.)
  )
);
pv.addPhysVolID("layer", layer_num);
```

```
if(x_slice.isSensitive()) {
  sens.setType("calorimeter");
  slice_vol.setSensitiveDetector(sens);
}
```

- Add volumes to envelope using position and a volume ID
- Can grab the sensitive XML string and make volumes have readout
  - Can change the sensitive detector type

Wrapping up class implementation

```
<include ref="compact/hcal_ATHENA.xml"/>
```



Negative ECal visualization

- Include class XML file in main XML file
- Include both XML and C++ files in CMakeLists
  - XML in compact directory, C++ in src
- Install the changes to fully implement
  - Visualization: `$ dd_web_display --export endcapP_insert.xml`
- Can also do more complex things
  - See negative ECal
    - Defines each individual tower/module and places them

Juggler running for calorimeters

- To run our standalone hadronic calorimeter endcap simulation through Juggler, we do the following:

gaudirun.py endcapP_insert_reco.py

- The reconstruction algorithms to be used are all defined in the **endcapP_insert_reco.py** file

Main parts of this file

Juggler algorithms that may be used:

```
# juggler components
from Configurables import Jug__Digi__CalorimeterHitDigi as CalHitDigi
from Configurables import Jug__Reco__CalorimeterHitReco as CalHitReco
from Configurables import Jug__Reco__CalorimeterHitsMerger as CalHitsMerger
# from Configurables import Jug__Reco__CalorimeterIslandCluster as IslandCluster
```

Input branches DD4HEP/Geant4 output ROOT file:

```
# branches needed from simulation root file
sim_coll = [
    "MCParticles",
    "HcalEndcapPHits",
    "HcalEndcapPHitsContributions",
    "HcalEndcapPInsertHits",
    "HcalEndcapPInsertHitsContributions",
    "EcalEndcapPHits",
    "EcalEndcapPHitsContributions",
    "EcalEndcapPInsertHits",
    "EcalEndcapPInsertHitsContributions"
]
```

Input, output, and parameters for each algorithm being used (same algorithm can be called by multiple detectors):

```
# Hcal Hadron Endcap

ci_hcal_daq = dict(
         dynamicRangeADC=200.*MeV,
         capacityADC=32768,
         pedestalMean=400,
         pedestalSigma=10)
ci_hcal_digi = CalHitDigi("ci_hcal_digi",
         inputHitCollection="HcalEndcapPHits",
         outputHitCollection="HcalEndcapPHitsDigi",
         **ci_hcal_daq)

ci_hcal_reco = CalHitReco("ci_hcal_reco",
        inputHitCollection=ci_hcal_digi.outputHitCollection,
        outputHitCollection="HcalEndcapPHitsReco",
        thresholdFactor=0.0,
        samplingFraction=ci_hcal_sf,
        **ci_hcal_daq)

ci_hcal_merger = CalHitsMerger("ci_hcal_merger",
        inputHitCollection=ci_hcal_reco.outputHitCollection,
        outputHitCollection="HcalEndcapPHitsRecoXY",
        readoutClass="HcalEndcapPHits",
        fields=["layer", "slice"],
        fieldRefNumbers=[1, 0])
```

Example – calorimeter digitization

```python
ci_hcal_daq = dict(
         dynamicRangeADC=200.*MeV,
         capacityADC=32768,
         pedestalMean=400,
         pedestalSigma=10)
ci_hcal_digi = CalHitDigi("ci_hcal_digi",
         inputHitCollection="HcalEndcapPHits",
         outputHitCollection="HcalEndcapHitsDigi",
         **ci_hcal_daq)
```

```cpp
// apply additional calorimeter noise to corrected energy deposit
const double eResRel = (eDep > 1e-6)
                         ? m_normDist() * std::sqrt(std::pow(eResRel[0] / std::sqrt(eDep), 2) +
                                                    std::pow(eRes[1], 2) + std::pow(eRes[2] / (eDep), 2))
                         : 0;

const double ped    = m_pedMeanADC + m_normDist() * m_pedSigmaADC;
const long long adc = std::llround(ped +  m_corrMeanScale * eDep * (1. + eResRel) / dyRangeADC * m_capADC);

double time = std::numeric_limits<double>::max();
for (const auto& c : ahit.getContributions()) {
  if (c.getTime() <= time) {
    time = c.getTime();
  }
}
const long long tdc = std::llround((time + m_normDist() * tRes) * stepTDC);
```

https://github.com/eic/juggler/blob/adbe6c7de7154b72588ab2d7d8503eba9698ac02/JugDigi/src/components/CalorimeterHitDigi.cpp

Additional information

- Information in these slides mostly comes from these repositories:

  https://github.com/bschmookler/athena_ana

  https://github.com/rymilton/eic_endcap_insert

- Official `ATHENA` software documentation:

  https://eic.phy.anl.gov/tutorials/eic_tutorial/