TB analysis algorithm

What you have done so far:

- You have been running RecExTB
- You know how to play with jobOptions properties
- You know how to produce ESD's

Now you are invited to write and run your own analysis algorithm from ESD:

- A skeleton is provided
- « Administrative » things like including the correct headers have already been done for you
- Exercise of increasing difficulty are proposed
- At the beginning, cut and paste from the transparencies is possible
- Then you'll be on your own (almost)
- At the end, you will have remade clusters after masking (pseudo) noisy cells

ESD content

ESD contain (limited to calo):

- « CaloCell »s: energy/time/quality factor, pointer to « CaloDetDescrElement » with Identifier/eta/phi and many other « fixed » quantities
- « LArCluster » (emtb-like, sliding window), « CaloCluster » topo
- beam instruments soon
- For MC, all true particles
- In the future, when combined reco is switched on in RecExTB: egamma (track+cluster including detailed cluster variables), jet, missing ET,….

ESD do NOT contain (by default):

- Raw data (LArDigits, LArRawChannel)

ESD use cases

What you can do with ESD:

- Revisit clustering
- Revisit cluster correction
- Compute sophisticated shower shape variable (e.g. second maximum in strips)
- In the future: rerun egamma

What you CANNOT do with standard ESD:

- Revisit OFC’s, Autocorrelation coeff etc… (need raw data)

Setting up TBAnalysis

TBAnalysis is a skeleton package meant to start an analysis

From main directory:

- cd ~/Athena
- (note that cmt configuration should already have been done source ~/cmthome/setup.(c)sh)
- cmt co -r TBAnalysis-00-00-08 TestBeam/TBAnalysis

  -r TBAnalysis-00-00-08 specifies a tag for a given version of the package. Dropping it means taking the last version
- cmt co –r PyAnalysisCore-00-00-05 PhysicsAnalysis/PyAnalysis/PyAnalysisCore
- cd TestBeam/TBAnalysis/*/cmt
- update the pointer to PyAnalysisCore in requirements
- source setup.sh (only once in a given window)
- cmt broadcast gmake # compiles the two packages
- (suggest you keep this window for compilation, xterm a new one)

Setting up TBAnalysis (2)

In new window (xterm &)

Setup cmt again:

- source ~/cmthome/setup.(c)sh
- cd Reconstruction/RecExample/RecExTB/*/cmt   #change into cmt dir
- source setup .(c)sh   #for run time environment
- cd ../run/   #change into run directory

get_files –jo TBAnalysis/TBAnalysis_topOptions.py

- (you can use <TAB>)
- This copies file TBAnalysis/share/TBAnalysis_topOptions.py into you run directory

ln –fs TBAnalysis_topOptions.py jobOptions.py

- creates a soft link so that TBAnalysis_topOptions.py is now taken by default when running athena

add the patch for the database at the end of jobOptions.py

athena >! tbanalysis.log #runs!

- Check the log file has some lines TBAnalysis INFO In execute
- Try option –s to have the detail of include file
- athena TBAnalysis/TBAnalysis_topOptions.py also runs

This jobOption runs TBAnalysis from the ESD

- Note that it is also possible to run TBAnalysis on the fly after reconstruction but it is of course slower

TBAnalysis log file

There is a number of DEBUG lines:

```
TBAna   DEBUG From cell ID: 747168768 :
  [4.1.1.1.0.71.2] bec 1 sampling 1 region 0 ieta 71 iphi 2
TBAna  DEBUG From cell : e=2170.33 eta=0.223438
  phi=0.0454369
TBAna  DEBUG From caloDDE :  hashID=526 sampling=1
  r raw=1500.02 z raw=337.957 Eta local=0.223438 Phi
  local=0.245437 deltaEta=0.003125 deltaPhi=0.0981748
```

Check in TBAnalysisSkeleton.cxx how easily they are obtained

« offline » Identifier

« Expanded Identifier » (Soft-2001-04): List of integer numbers to locate any readout element in atlas (cell for the calorimeters), e.g for a cell in LarEm barrel [4.1.1.1.0.69.2] :

- LArCalorimeter: 4/Larem:1/Barrel-endcap [-1,1]/Sampling [0-3]/Region [0:1]/Eta/Phi

« Compact Identifier »: a 32 bits integer compactifying the above list (by attributing the right number of bit to each field)

- 747152384

« Hash Identifier »: a numbering of cells from 0 to ncells without any holes

- 518
- « calo hash »: ncells of Larem HEC FCAL Tile
- « subcalo hash » : ncells of each calo

Never ever try to pack/expand/hash « by hand » the identifier but use the provided tools (see example in DEBUG printout)

- Offline ⇔ test beam differences handled by using the right « dictionary » (no code change)

Conversion from/to online identifier (see Walter’s slide)

Browse root ntuple

Browse root file:

```
root
TBrowser b;
```

In the left hand window, select ntuple.root

Then click on ROOT folder (still in left hand window)

Navigate to tree

Double click on variables to plot

(Cells have been deliberately removed see later)

TBAnalysis_topOptions.py 1

```
# input file names
EventSelector.InputCollections = ["athena.root"]
# could have multiple files ..=["athena1.root"  , "athena2.root"]

# Set output level threshold DEBUG, INFO, WARNING, ERROR,
    FATAL
MessageSvc.OutputLevel = INFO

# Number of Events to process
theApp.EvtMax = 10

#histogram file name
HistogramPersistencySvc.OutputFile = "histosESD.root";
```

Using a different input file

In TBAnalysis_topOptions.py change to:

```
EventSelector.InputCollections = [
"/afs/cern.ch/user/r/rmcphers/scratch0/8.8.0/esd/run2100180.root"]
```

Either copy the PoolFileCatalog.xml from this directory to your RecExTB/*/run directory

Or run:

```
pool_insertFileToCatalog /
    afs/cern.ch/user/r/rmcphers/scratch0/8.8.0/esd/run2100180.root
```

For a castor file, prepend rfio:/castor/cern.ch/etc….

TBAnalysis_topOptions.py 2

```
####### TBAnalysis algorithm configuration ########
theApp.Dlls += [ "TBAnalysis" ]
theApp.TopAlg += [ "TBAnalysisSkeleton/TBAna" ]
TBAna = Algorithm( "TBAna" )
TBAna.OutputLevel=DEBUG
#TBAna.ClustersName = "CaloTopoClusterEM"
TBAna.ClustersName="LArClusterTBEM"   # clusters to be analysed
#cells to be skipped in em, element of identifier:
#   barrel/endcap sample region ieta iphi
TBAna.InputIDs=["1 1 0 69 2", "1 2 0 9 8"]
```

TBAnalysis_topOptions.3

Typical configuration flags

```
doCBNT=True  # rebuild CBNT ntuple
useROOTNtuple=True
doLAr=True
doTile=True
doTruth=False
doTileNtuple=False
doDetailedNtuple=False
doCaloCluster=False
doEmCluster=False
doEMTBCluster=True
doCaloTopoCluster=True
#typical if statement
if doCBNT:
        NTupleSvc = Service( "NTupleSvc" )
      etc.. Etc…
```

WARNING : in python, blocks are defined by indentation (fortunately emacs knows about it, let him do it)

Some CBNT blocks switched off:

```
CBNT_LArCell.Enable = False # no lar cell
CBNT_TileCell.Enable = False # no tile cell
```

FAQ

How do I know what objects are available in StoreGate?

- StoreGateSvc= Service (StoreGateSvc)
- Add StoreGateSvc.Dump=True (StoreGateSvc.OutputLevel=INFO if necessary) (at the end) and run again

⇒ long list of object type and keys

⇒ Documentation on ntuple variables: Software→Software Domain →Reconstruction → Reconstruction in Athena → CBNT_Athena → Variable list http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/domains/Reconstruction/packages/CBNT_Athena/CBNT_variables.htm

Documentation about STL (C++ vector and more): http://www.sgi.com/tech/stl/

Exercise 1a

Looping on clusters, printing basic quantities, fill simple histogram

You can start modifying TBAnalysisSkeleton.cxx, where « Exercises code to be inserted here » is indicated

```
// Retrieve clusters
const CaloClusterContainer* theClusterContainer;
sc=m_eventStore->retrieve(theClusterContainer,m_clustersName);
if(sc != StatusCode::SUCCESS){
   logStream << MSG::ERROR << " Could not retrieve clusters " <<
    m_clustersName << endreq ;
   return StatusCode::SUCCESS ;   }
```

Exercise 1b

loop on clusters and print energy

```
 typedef CaloClusterContainer::const_iterator ClusterIter;
 ClusterIter itrCluster;
 ClusterIter itrClusterBeg = theClusterContainer->begin();
 ClusterIter itrClusterEnd = theClusterContainer->end();
 float nCluster = 0.;
 for ( itrCluster=itrClusterBeg ; itrCluster != itrClusterEnd; ++itrCluster) {
   const CaloCluster* theCluster = *itrCluster;
   nCluster += 1.0;
   double theEnergy=theCluster->energy();
   logStream << MSG::INFO << "Original cluster energy :" << theEnergy << endreq ;
}
```

Fill energy and n cluster histogram. Insert at the right place:

```
 // fill cluster energy histogran
   m_eCluster->fill(theEnergy/GeV,1.);  // note that GeV=1000

// fill n cluster histogran
  m_nCluster->fill(nCluster,1.);
```

Histograms were booked in initialize() method with (you do not need to do it) (do not type this in, it is already there):

```
 m_eCluster =histoSvc()->book(basepath + "/eCluster","ecluster",m_NEClusterBins,
                              m_minEnergy/GeV,m_maxEnergy/GeV);
```

Exercise 1c

Compile (in TBAnalysis) and run again (in RecExTB) (note that you can now remove the DEBUG cell printout) changing DEBUG to INFO in:

```
TBAna.OutputLevel=DEBUG
```

Browse root file:

```
root
TBrowser b;
```

In the left hand window, select ntuple.root and histosESD.root

Then click on ROOT folder (still in left hand window)

Navigate to histos and double click

Exercise 2

Recompute cluster energy from its cells

In the cluster loop, insert:

```
CaloCluster::cell_iterator itrCell = theCluster->cell_begin();
double eSum=0.;
for (;itrCell!=theCluster->cell_end(); ++itrCell) {
    eSum+=(*itrCell)->energy();
}

logStream << MSG::INFO << "Recomputed cluster energy :" <<
    eSum << endreq ;
```

Recompile, rerun: compare in the log file the original and recomputed cluster energy: are they equal ?

Exercise 3

Skip selected cells from cluster energy computation

A method to compute a weight is provided at the bottom of TBAnalysisSkeleton.cxx:

- this->computeWeight(*itrCell)
- It return 0 for cell which identifier is provided in jobOption, 1 otherwise.

TBAna.InputIDs=["1 1 0 69 2", "1 2 0 9 8"]

Use this method to skip « noisy » cells

Compile and run, now check wether recomputed energy is different

Exercise 4a

Build new clusters from the previous one, skipping selected cells, put new cluster in CBNT, compare in ntuple old and new cluster

Before cluster loop, create a new cluster container

```
CaloClusterContainer * outputClusterContainer = new CaloClusterContainer();
```

and record it (m_clustersOutputName is a property already defined,"false" indicates it cannot be modified by another algorithm)

```
 sc=m_eventStore->record
   (outputClusterContainer,m_clustersOutputName,false);
 if(sc != StatusCode::SUCCESS){
   logStream << MSG::ERROR << " Could not record clusters" <<
   m_clustersOutputName << endreq ;
  return StatusCode::SUCCESS ;
}
```

Exercise 4b

Inside the cluster loop, create a new cluster:

```
CaloCluster * newCluster= new CaloCluster();
```

…and put it in the container

```
outputClusterContainer->push_back(newCluster);
```

Inside the cluster cell loop, fill the new cluster with cells from the old (weight is what is returned by computeWeight(…):

```
newCluster->addCell(cellContainer,*itrCell,weight);
```

Outside cluster cell loop print the new cluster energy

Compile, run compare the energies printed out…

Exercise 4c

Activate CBNT_myCluster:

```
CBNT_Athena.Members += [ "CBNT_CaloCluster/CBNT_myCluster"]
CBNT_myCluster = Algorithm( "CBNT_myCluster" )
CBNT_myCluster.ClusterColl = "myClusters"
CBNT_myCluster.EMOnly = True
CBNT_myCluster.Suffix = "_mine"
CBNT_myCluster.Enable = True # switch to true to activate
```

Request more events for the ntuple:

```
theApp.EvtMax = ???
```

Browse root ntuple, compare old and new energies: cl_e_tb_em cl_e_mine

Notice you can now readily also compare the energy per layer cl_eemb0_mine, the position: cl_eta1_mine, cl_phi2_mine etc…

Ultimate Exercise 4d

With zero code modification nor recompilation, only with few mods of the jobOption:

- you can now do the same thing for topo clusters
  - Just need to change to

TBAna.ClustersName = "CaloTopoClusterEM «

TBAna.ClustersOutputName=« myTopoClusters »

TBAna.histoDir=« /CaloClusterTopo»   # histogram dir

- You can even have both your modified TB clusters and modified topo clusters simultaneously in the ntuple
  - Just need to create TBAnaTopo and CBNT_myClustersTopo and configure them appropriately

Setting up TBAnalysis for a >9.0.0

In order to run with a more recent AtlasRelease few changes are necessary:

check out later tags for TBAnalysis and check which other packages to check out

remove the line
include("AthenaCommon/SystemOfUnits.py")

copy section about GlobalFlags from RecExTB_combined_2004_jobOptions.py

add database patch at end of jobOptions.py file

now you are set !