Analysis in CMSSW

Benedikt Hegner

DESY Hamburg

Outline

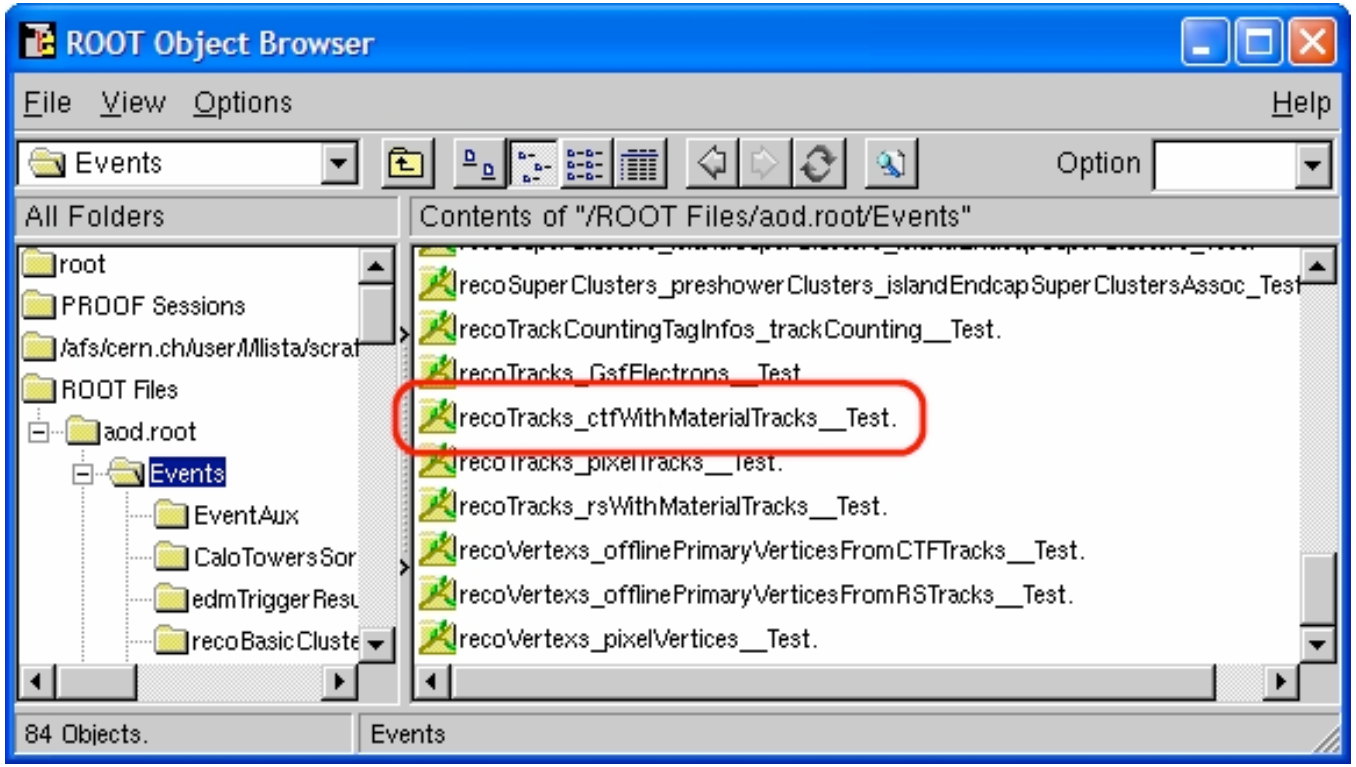
- Bare ROOT access
- FWLite access
- FWLite in python
- Full framework

Bare ROOT

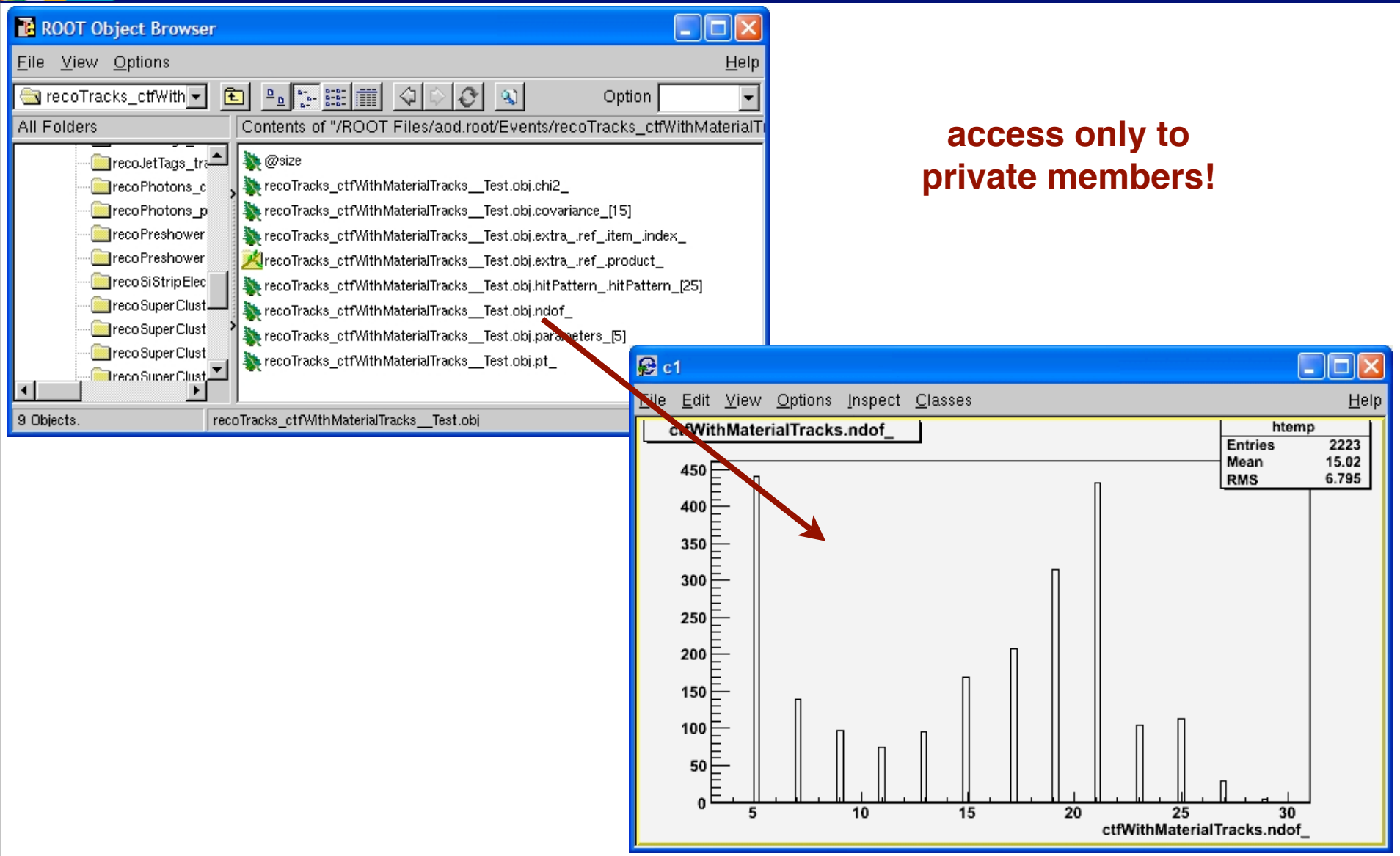
- without CMSSW installation
- very basic analysis
- only access to object members

bare ROOT

```
root.exe
[] TFile f("aod.root")
[] new TBrowser()
```



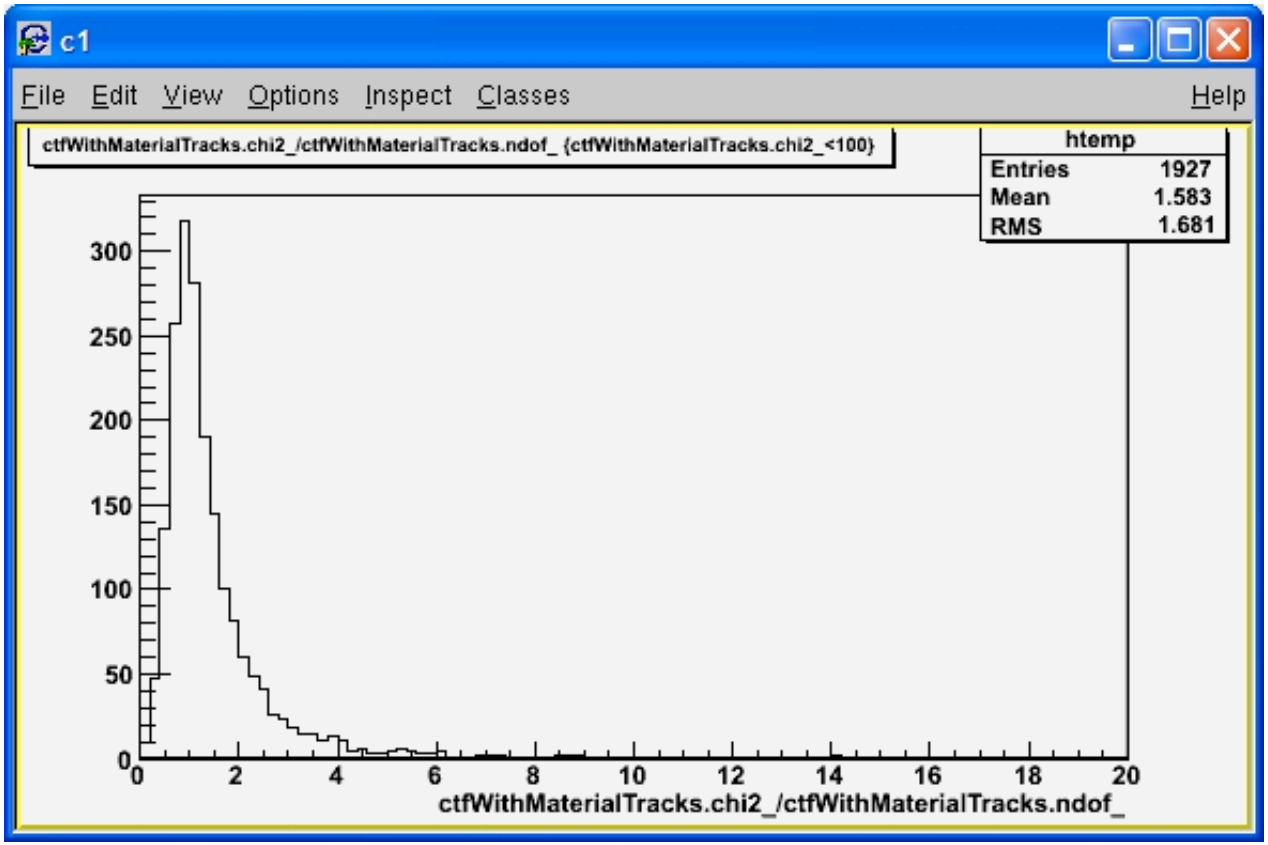
bare ROOT



access only to private members!

from ROOT prompt

```
[] Events.Draw("ctfWithMaterialTracks.ndof_")
[] Events.Draw("ctfWithMaterialTracks.chi2_/ctfWithMaterialTracks._ndof", "ctfWithMaterialTracks.chi2<200")
```



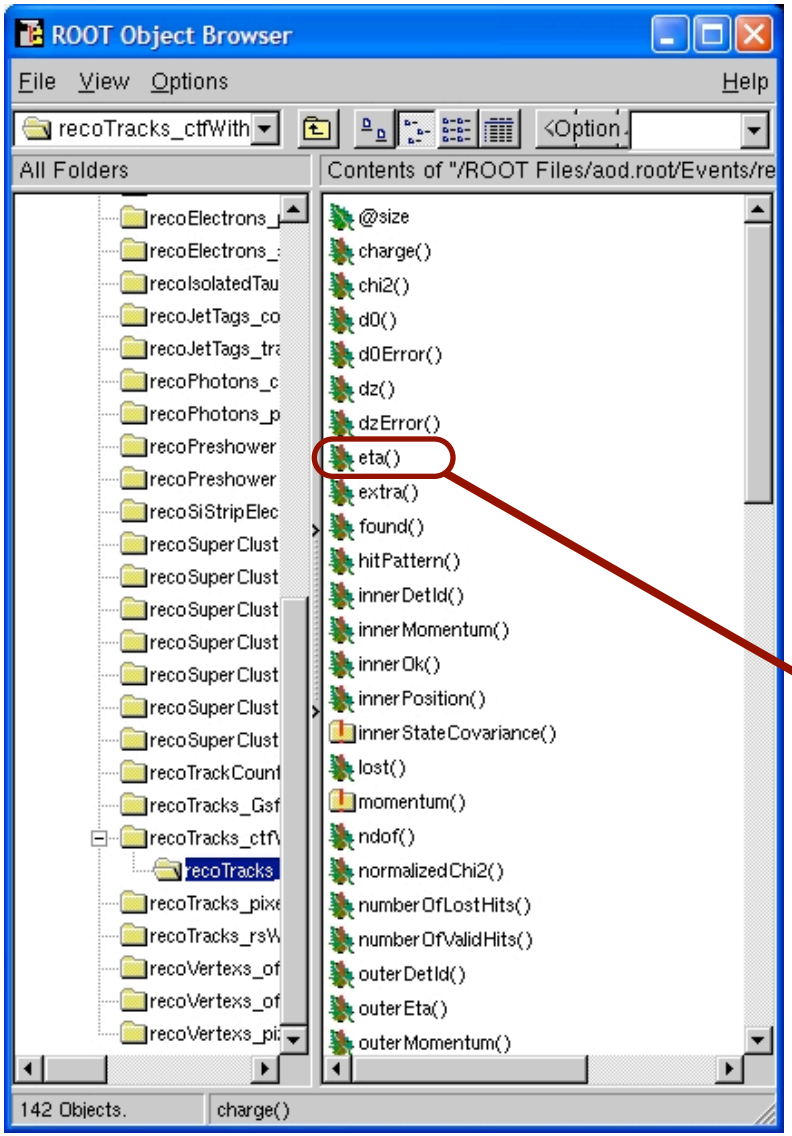
Using FWLite

Preparing the environment

```
 $ scramv1 project CMSSW CMSSW_1_2_0
 [...]
 $ cd CMSSW_1_2_0/src

setting runtime variables
 $ eval `scramv1 run -csh`     (C-Shell)
 $ eval `scramv1 run -sh`      (BASH)

accessing the cvs server
 $ cmscvsroot CMSSW
 $ cvs login                   (password: 98passwd)
```

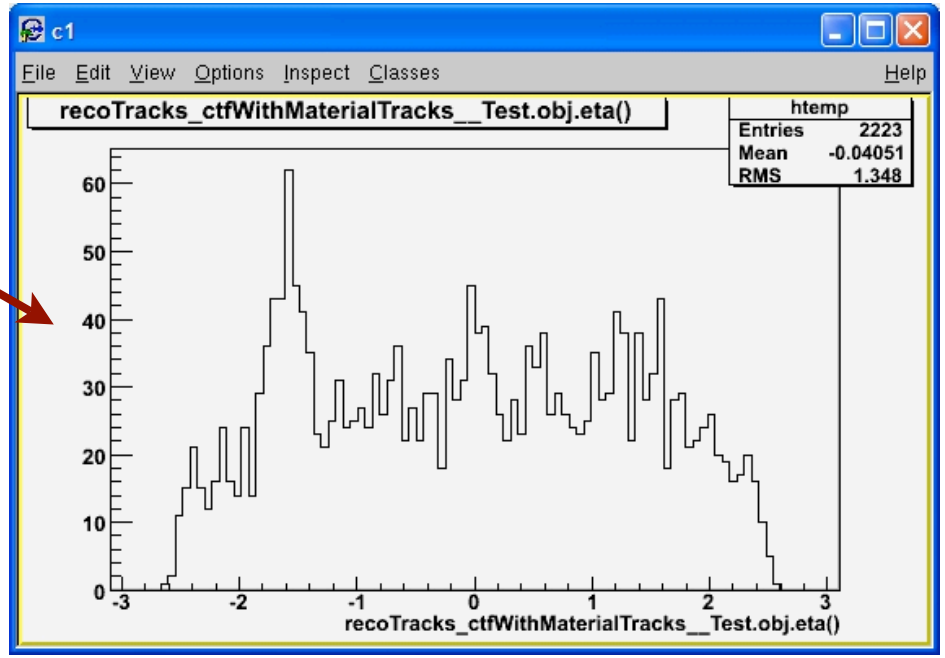
FWLite

- reading produced root files
- full access to the class methods
- no need to write full-blown framework modules



Automatic library loading

```
[] gSystem->Load("libFWCoreFWLite");
[] AutoLibraryLoader::enable();
[] new TBrowser();
```



reading collections of objects

aliases

```
TFile theFile("generatorOutput.root");

TTree *events = (TTree*)theFile.Get("Events");
events->GetEntry();

edm::HepMCProduct prod;
HepMC::GenEvent *genEvent;
TBranch *sourceBranch = events->GetBranch(events->GetAlias("source"));
sourceBranch->SetAddress(&prod);

for( unsigned int index = 0; index < events->GetEntries(); index++)
{
   sourceBranch->GetEntry(index);

   events->GetEntry(index,0);
   genEvent = prod.GetEvent();
   ...
```

Using FWLite and Python together

```
$ cvs co PhysicsTools/PythonAnalysis/python
$ export PYTHONPATH=$CMSSW_BASE/src/PhysicsTools/PythonAnalysis/python:$PYTHONPATH
$ python -i start.py

Preparing CMS tab completer tool...
Loading FWLite dictionary...
>>>
```

start.py

```
from ROOT import *
from cmstools import *

gSystem.Load("libFWCoreFWLite.so")
AutoLibraryLoader.enable()
```

One short example - MCTruth2.py

```
from ROOT import *
from cmstools import *

### prepare the FWLite autoloading mechanism
gSystem.Load("libFWCoreFWLite.so")
AutoLibraryLoader.enable()

# load the file with the generator output
theFile = TFile("generatorOutput.root")

# access the event tree
events = EventTree(theFile.Get("Events"))

# access the products inside the tree
# aliases can be used directly
sourceBranch = events.branch("source")

# loop over the events
for event in events:
    genEvent = sourceBranch().GetEvent()
    ...
```

no need to know the type beforehand

more examples in PhysicsTools/PythonAnalysis/test

interactive use - help command

```
>>> help(HepMC.GenParticle)
```

```
Help on class HepMC::GenParticle in module __main__:

class HepMC::GenParticle(ROOT.ObjectProxy)
 |  Method resolution order:
 |      HepMC::GenParticle
 |      ROOT.ObjectProxy
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  BeginDaughters(...)
 |      int GenParticle::BeginDaughters()
 |
 |  CollisionNumber(...)
 |      int GenParticle::CollisionNumber()
 |
 |  CreationVertex(...)
 |      CLHEP::HepLorentzVector GenParticle::CreationVertex()
 |
 |  DecayVertex(...)
 |      CLHEP::HepLorentzVector GenParticle::DecayVertex()
 |
 |  EndDaughters(...)
:
```

How to get a list of aliases

```
>>> ...
>>> theFile = TFile("generatorOutput.root")
>>> events = EventTree(theFile.Get("Events"))
>>> for alias in events.getListOfAliases():
...     alias.Print()
[...]
OBJ: TNamed     CaloHitsTk      PCaloHit_r_CaloHitsTk.obj
OBJ: TNamed     CastorBU        PCaloHit_r_CastorBU.obj
OBJ: TNamed     CastorFI        PCaloHit_r_CastorFI.obj
OBJ: TNamed     CastorPL        PCaloHit_r_CastorPL.obj
OBJ: TNamed     CastorTU        PCaloHit_r_CastorTU.obj
OBJ: TNamed     EcalHitsEB      PCaloHit_r_EcalHitsEB.obj
OBJ: TNamed     EcalHitsEE      PCaloHit_r_EcalHitsEE.obj
OBJ: TNamed     EcalHitsES      PCaloHit_r_EcalHitsES.obj
[...]
```

how to loop over event data objects

python objects and most of the root ojects

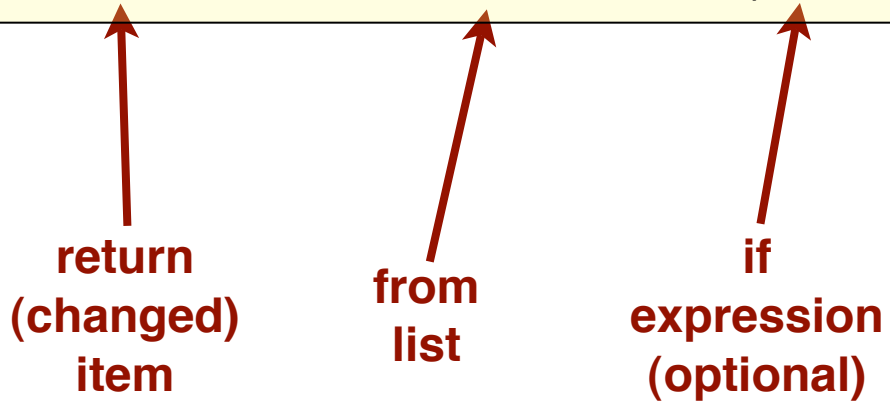
```
for item in iterable:
    doSomething
```

no direct access to the container itself
but to iterators

```
begin = genEvent.particles_begin()
end = genEvent.particles_end()
for particle in loop(begin,end):
    doSomething
```

how to do filtering - list comprehensions

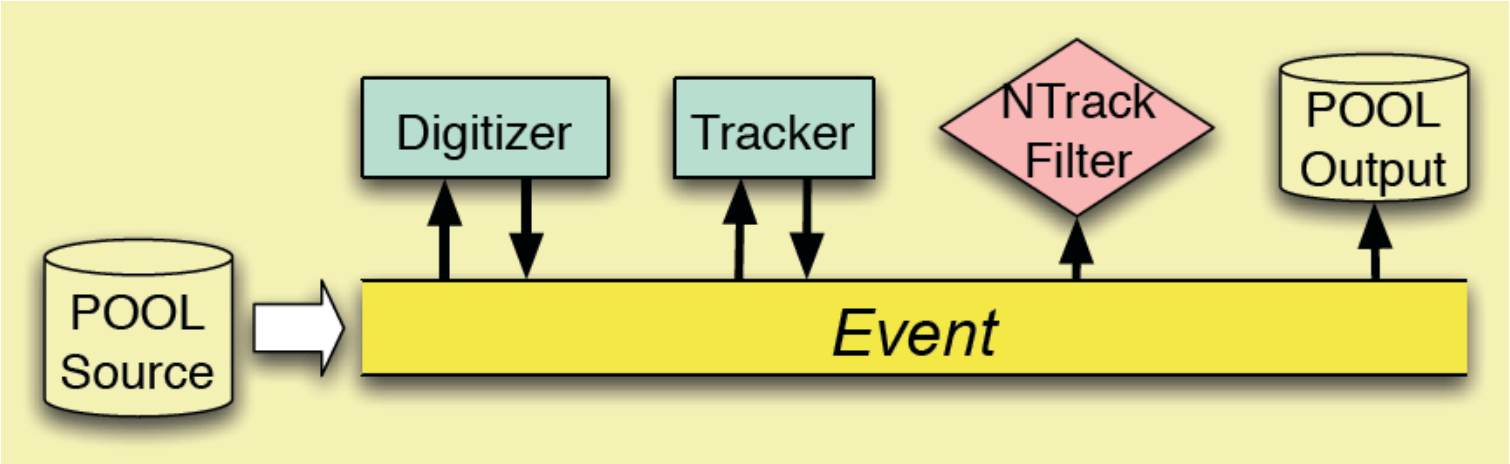
```
resulting list = [ value for item in iterable (if expression) ]
```




```
selectedLeptons = [l for l in theLeptons if (l.pt>20 and l.eta<2.5) ]
```

```
...
# register the interesting branches
source = events.branch("VtxSmeared")
recoJets = events.branch("iterativeCone5CaloJets")
globalMuons = events.branch("globalMuons")

# loop over the events
for event in events:
    genEvent = sourceBranch().GetEvent()
    begin = genEvent.particlesBegin()
    end = genEvent.particlesEnd()

    myMCParticles = [ p for p in loop(begin,end) if p.status!=3 ]

    myJets = [ jet for jet in recoJets if jet.energy>20 ]
    ...
```

Full CMSSW framework



- Source
- EDProducer
- EDFilter
- EDAnalyzer
- OutputModule
- ...

```
process EXAMPLE = {
        source = EmptySource { untracked int32 maxEvents = 2}
        module int = IntProducer { int32 ivalue = 2 }

        module test = IntTestAnalyzer {
                untracked int32 valueMustMatch = 2
                untracked string moduleLabel = "int"
        }
        service = Tracer {}
        module dump = EventContentAnalyzer {}

        path p = {int, test, dump}
}
```

very useful!

```
$ cmsRun example.cfg
```

CMSSW file input and output

input

```
source = PoolSource
{
  untracked vstring fileNames = {"file:input.root"}
  untracked int32 maxEvents = -1
}
```

many other options possible!

PROCESS

output

```
module Out = PoolOutputModule
{
  untracked string fileName = "output.root"
}
```

```
untracked vstring outputCommands =
  {
    "keep *",
    "drop *_*_*_HLT",
    "keep FEDRawDataCollection_*_*_*"
  }
```

Accessing event data I

```
# by module and default product label
 Handle<TrackVector> trackPtr;
 iEvent.getByLabel("tracker", trackPtr );

# by module and product label
 Handle<SimHitVector> simPtr;
 iEvent.getByLabel("detsim", "pixel" ,simPtr );

# by type
 vector<Handle<SimHitVector> > allPtr;
 iEvent.getByType( allPtr );

# by Selector
 ParameterSelector<int> coneSel("coneSize",5);
 Handle<JetVector> jetPtr;
 iEvent.get( coneSel, jetPtr );
```

there are some more...

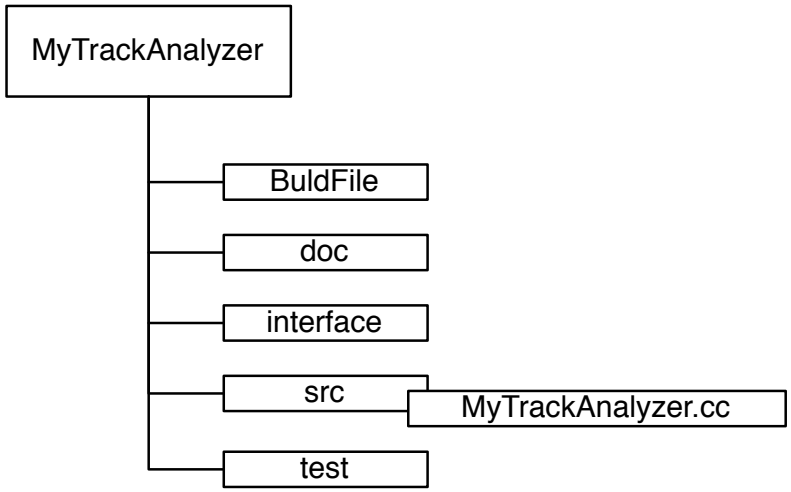
Accessing event data II

```
Handle< HepMCProduct > EvtHandle;
iEvent.getByLabel("source", EvtHandle);  // or "VtxSmeared"!

const HepMC::GenEvent* genEvent = EvtHandle->GetEvent() ;

for ( HepMC::GenEvent::particle_const_iterator
      p=genEvent->particles_begin();
      p!=genEvent->particles_end(); p++)
{
    std::cout << (*p)->status() << std::endl;
}
```

HepMC online documentation:
http://mdobbs.web.cern.ch/mdobbs/HepMC/html_reference/index.html

Creating an EDAnalyzer

```
$ cd CMSSW_1_0_4/src
$ mkdir Tutorial
$ cd Tutorial
$ mkedanlzr -list
$ mkedanlzr -track MyTracks
```



EDAnalyzer source code

```
private:
   void beginJob( const edm::EventSetup& );
   void analyze( const edm::Event&, const edm::EventSetup& );
   void endJob();
```

```
// ------------ method called to for each event  ------------
void
MyTracks::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
   using namespace edm;
   using reco::TrackCollection;

   Handle<TrackCollection> tracks;
   iEvent.getByLabel(trackTags_,tracks);

   for(TrackCollection::const_iterator itTrack = tracks->begin();
       itTrack != tracks->end(); ++itTrack)
   {
      int charge = itTrack->charge();
   }
```

```
DEFINE_FWK_MODULE(MyTracks);
```

Building the example analyzer I

```
$ cd MyTrackAnalyzer
$ scramv1 build
```

BuildFile

```
<use name=FWCore/Framework>
<use name=Foundation/PluginManager>
<use name=FWCore/ParameterSet>
<flags SEAL_PLUGIN_NAME="TutorialMyTracks">
<use name=DataFormats/TrackReco>
<export>
   <lib name=TutorialMyTracks>
   <use name=FWCore/Framework>
   <use name=Foundation/PluginManager>
   <use name=FWCore/ParameterSet>
   <use name=DataFormats/TrackReco>
</export>
```

Building the example analyzer II

```
$ cd MyTrackAnalyzer
$ scramv1 build
```

```
Parsing BuildFiles
..
[...]

Entering Package Tutorial/MyTracks
Nothing to be done for MyTracks/interface:
Entering library rule at Tutorial/MyTracks
>> Compiling /afs/cern.ch/user/h/hegner/dev/CMSSW_1_0_4/src/Tutorial/MyTracks/src/MyTracks.cc
/afs/cern.ch/user/h/hegner/dev/CMSSW_1_0_4/src/Tutorial/MyTracks/src/MyTracks.cc: In member function
`virtual void MyTracks::analyze(const edm::Event&, const edm::EventSetup&)':
/afs/cern.ch/user/h/hegner/dev/CMSSW_1_0_4/src/Tutorial/MyTracks/src/MyTracks.cc:98: warning: unused
variable `int charge'
>> Building shared library tmp/slc3_ia32_gcc323/src/Tutorial/MyTracks/src/libTutorialMyTracks.so
Copying tmp/slc3_ia32_gcc323/src/Tutorial/MyTracks/src/libTutorialMyTracks.so to productstore area:
@@@@ Checking shared library for missing symbols:
@@@@ ----> OK, shared library FULLY-BOUND (no missing symbols)
@@@@ Checking SEAL plugin:
@@@@ ----> OK, SEAL plugin loaded successfully --
--- Registered SEAL plugin TutorialMyTracks
Leaving library rule at Tutorial/MyTracks
Nothing to be done for MyTracks/doc:
Leaving Package Tutorial/MyTracks
>> Package MyTracks built
```

Common mistake - missing entry in BuildFile

```
@@@@ Checking shared library for missing symbols:
tmp/slc3_ia32_gcc323/src/L1Fake/L1FakeAnalyzer/src/libL1FakeL1FakeAnalyzer.so:
undefined symbols
  _ZN12HcalTBTiming8setTimesEddddd
        needed by
/afs/cern.ch/cms/sw/slc3_ia32_gcc323/cms/cmssw/CMSSW_0_8_0/lib/slc3_ia32_gcc323/
libRecoTBCaloHcalTBObjectUnpacker.so
          needed by
tmp/slc3_ia32_gcc323/src/L1Fake/L1FakeAnalyzer/src/libL1FakeL1FakeAnalyzer.so
  _ZN12HcalTBTiming7setHitsERKSt6vectorIdSaIdEES4_S4_S4_S4_S4_
        needed by
/afs/cern.ch/cms/sw/slc3_ia32_gcc323/cms/cmssw/CMSSW_0_8_0/lib/slc3_ia32_gcc323/
libRecoTBCaloHcalTBObjectUnpacker.so
          needed by
tmp/slc3_ia32_gcc323/src/L1Fake/L1FakeAnalyzer/src/libL1FakeL1FakeAnalyzer.so
gmake: ***
[tmp/slc3_ia32_gcc323/src/L1Fake/L1FakeAnalyzer/src/libL1FakeL1FakeAnalyzer.so]
Error 1
```

```
$ cmsfilt.py _ZN12HcalTBTiming8setTimesEddddd
<use name=TBDataFormats/HcalTBObjects>
```

(~hegner/public/cmsfilt.py on lxplus)

The entire gen+sim+digi+reco-chain

```
module myModule = MyTracks {}
path p = {..., myModule, ... }
```

```
[...]

# create MC events
include "IOMC/GeneratorInterface/data/PythiaHZZ4mu.cfi"

# vertex smearing
include "IOMC/GeneratorInterface/data/PythiaHZZ4mu.cfi"

# simulate the energy deposition in the detector via Geant
include "SimG4Core/Application/data/SimG4Object.cfi"

# compute the detector response (digis)
include "SimG4Core/Application/data/DigiWithEcalZeroSuppression.cff"

# do reconstruction
include "Configuration/Examples/data/RECO.cff"

# define execution order
sequence smear_and_geant = { VtxSmeared,  g4SimHits}
sequence digitization = {doAllDigiWithEcalZeroSup}
path p = {smear_and_geant, digitization, reconstruction}

[...]
```

Where to get information

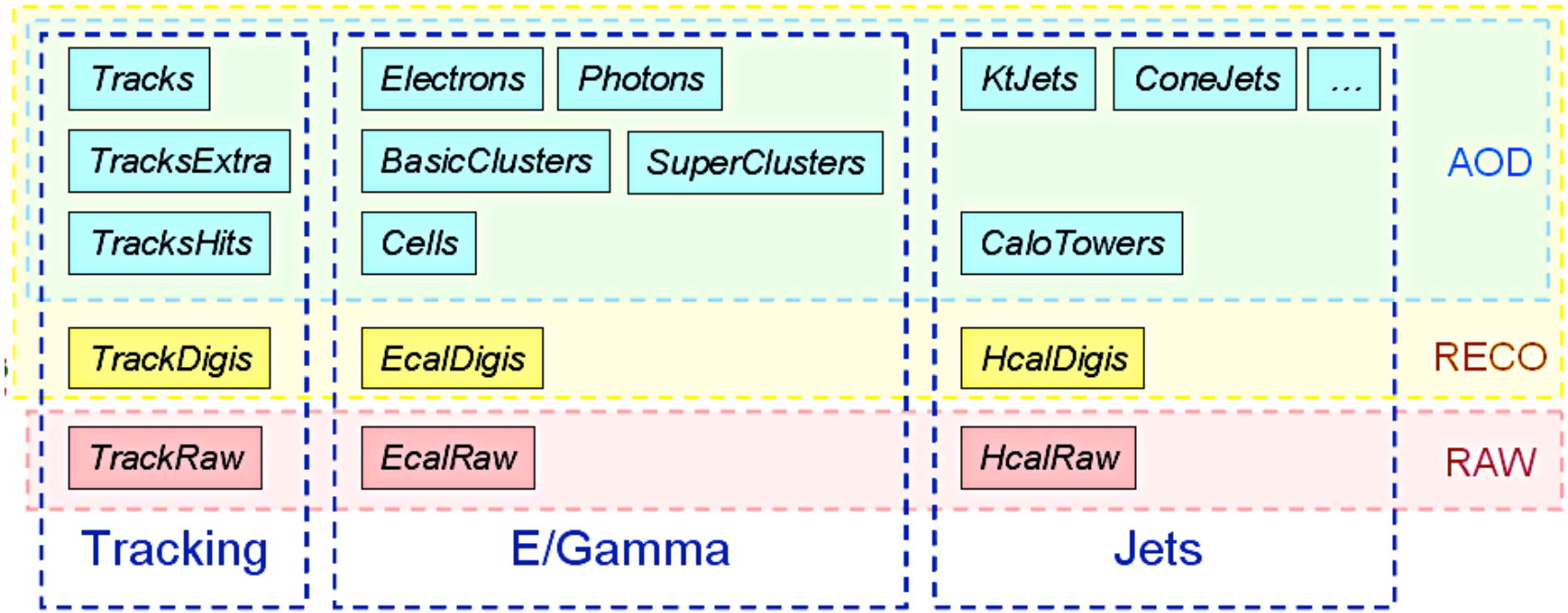
- Software General page:
  http://cmsdoc.cern.ch/cms/cpt/Software/html/General/
- CMS Workbook:
  https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook
- Reference manual:
  http://cmsdoc.cern.ch/cms/sw/slc3_ia32_gcc323/cms/cmssw/CMSSW_1_0_4/doc/html/index.html
- WEBcvs:
  http://cmsdoc.cern.ch/swdev/viewcvs/viewcvs.cgi/?cvsroot=CMSSW
- LXR:
  http://cmslxr.fnal.gov/lxr/
- python interactive help

BACKUP

What is stored in the event files?

Different file content types

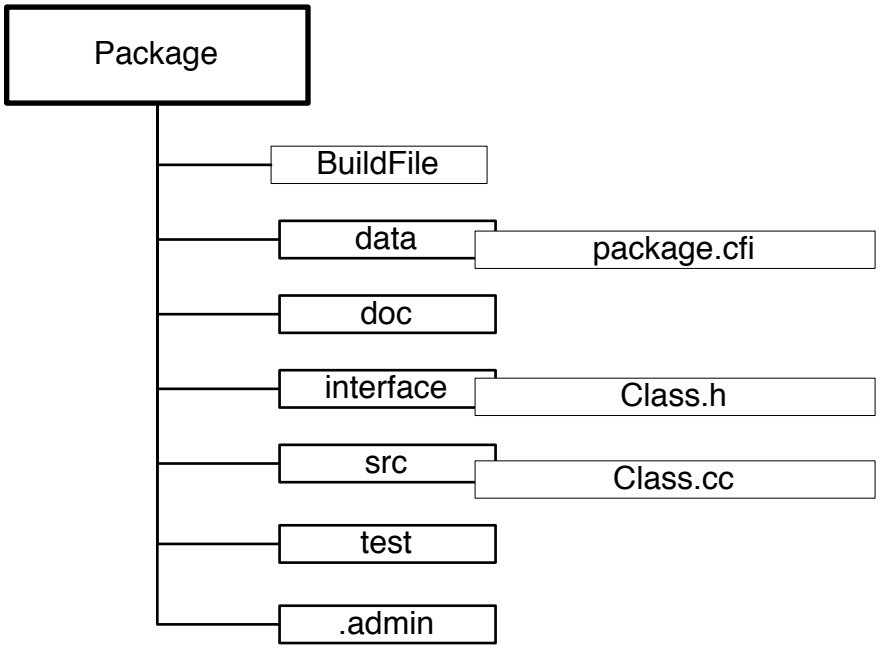
- FEVT: Full EVenT
- RECO: RECOnstruction
- RECOSIM: RECOnstruction + selected simulation information
- AOD: Analysis Object Data (a compact subset of RECO format)
- AODSIM: AOD + generator information



http://cmsdoc.cern.ch/Releases/CMSSW/latest_release/doc/html

The release area

```
Project/subproject/package

e.g. CMSSW_1_0_4/GeneratorInterface/ToprexInterface
```



Writing EDProducers

let's do some

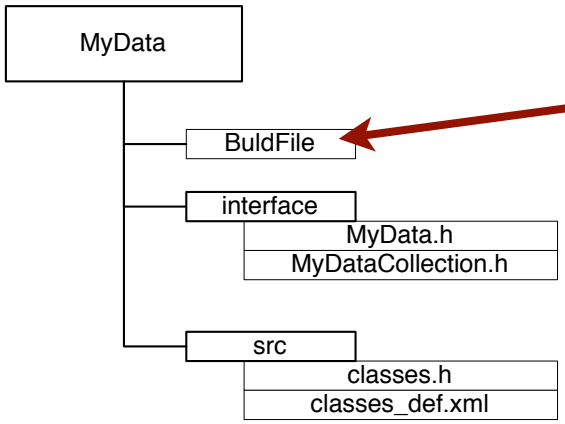
VOODOO

Making your data persistent I

```
$ cd src/DataFormats
$ mkdir MyData
$ cd MyData
$ mkdir interface
$ mkdir src
```

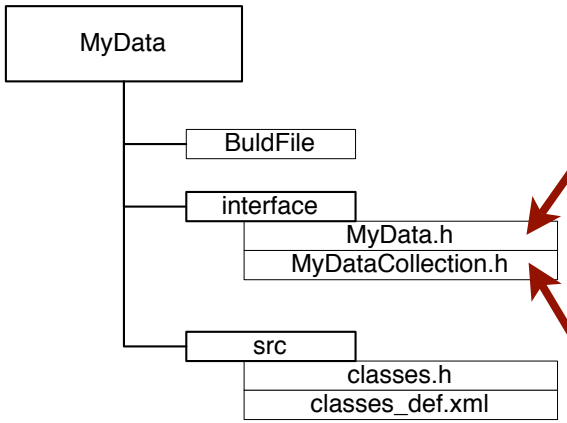
can be placed in any Subsystem (e.g. your analysis directory)

```
<use name=DataFormats/Common>
<export>
 <lib name=DataFormatsMyData>
 <use name=DataFormats/Common>
</export>
```



https://twiki.cern.ch/twiki/bin/view/CMS/CreatingNewProducts

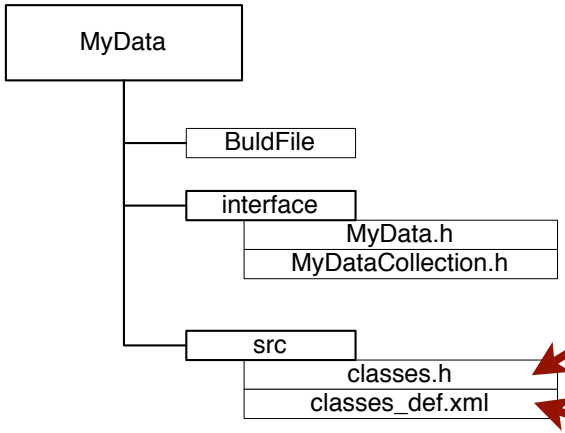
Making your data persistent II




```
#ifndef DATAFORMATS_MYDATA_H
#define DATAFORMATS_MYDATA_H

class MyData {
public:

  MyData() {}
  MyData(unsigned int tracks): numTracks_(tracks) {}

  inline unsigned int GetNumberOfTracks() { return numTracks_; }
  inline void SetNumberOfTracks(unsigned int tracks) { numTracks_ = tracks; }

private:

  unsigned int numTracks_;
};

#endif // DATAFORMATS_TRACKUTIL_H
```

```
#ifndef DATAFORMATS_MYDATACOLLECTION_H
#define DATAFORMATS_MYDATACOLLECTION_H

#include <vector>
#include "DataFormats/MyData/interface/MyData.h"

typedef std::vector<MyData> MyDataCollection;

#endif // DATAFORMATS_MYDATACOLLECTION_H
```

Making your data persistent III




```
#ifndef MYDATA_CLASSES_H
#define MYDATA_CLASSES_H

#include "DataFormats/MyData/interface/MyDataCollection.h"
#include "DataFormats/Common/interface/Wrapper.h"
#include <vector>

namespace {
  namespace {
    edm::Wrapper<MyDataCollection> MyDataCollectionWrapper;
  }
}

#endif // MYDATA_CLASSES_H
```

```
<lcgdict>
  <class name="MyData"/>
  <class name="std::vector<MyData>"/>
  <class name="edm::Wrapper<std::vector<MyData> >"/>
</lcgdict>
```

space!

Putting products in an event

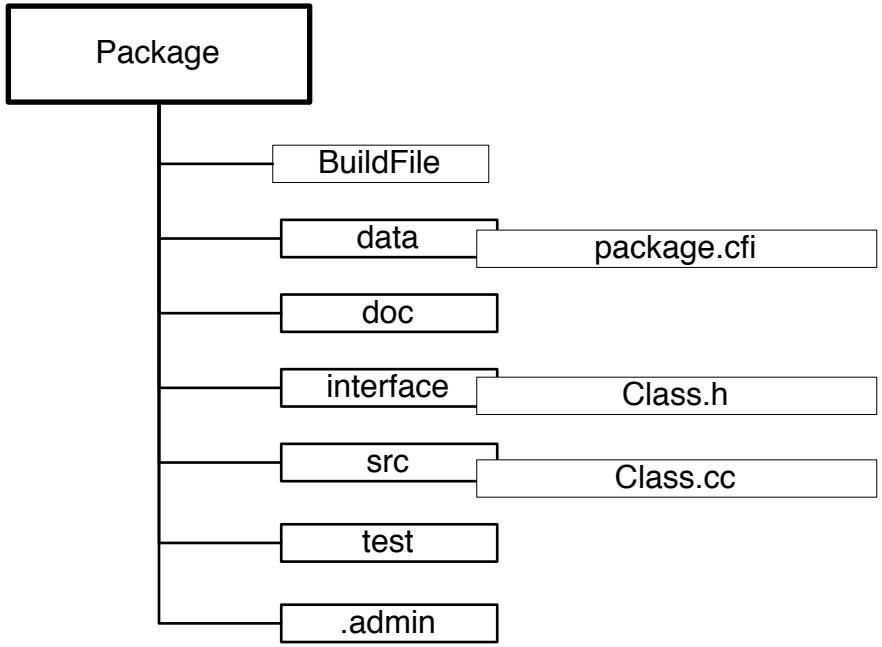
```
$ cd src/Tutorial
$ mkedprod MyProducer
```

```
MyProducer::MyProducer(const edm::ParameterSet& iConfig)
{
    produces<MyData>();
}

void
MyProducer::produce(edm::Event& iEvent, const edm::EventSetup&)
{
    std::auto_ptr<MyData> myData( new MyData );
    [...]
    iEvent.put(myData);
}
```

```
scramv1 list

scramv1 project

eval `scramv1 run -csh`

scramv1 build

scramv1 clean

...
```

SCRAM = Software release and management tool

```
Project/subproject/package

CMSSW_1_0_4/GeneratorInterface/ToprexInterface
```



```
class Particle(object):
    pass


# main routine
if __name__ == "__main__":

    aParticle = Particle()
    aParticle.px = 3.0
    aParticle.py = 3.0
...
```

struct like objects

“real” objects

```
class Analysis(object):

    def __name__(self):
        pass


    def printStats(self, verbosity):
        print ‘some text’
```

how to handle std::vectors and similar

C++

```
std::vector<HepMC::GenParticle> genParticles;
```

python long

```
GenParticleVector = vector(HepMC.GenParticle)
genParticles = GenParticleVector()
```

python short

```
genParticles = vector(HepMC.GenParticle)()
```

```
genParticles = vector(genParticle)()

genParticles.append(aMuon)

print genParticles[1].pt

for particle in loop(genParticles):
    print particle.pt
```

wrapper for std::vector

standard python

```
genParticles = []

genParticles.append(aMuon)

print genParticles[1].pt

for particle in genParticles:
    print particle.pt
```

One example of long PyROOT syntax

```
from cmstools import *

### open root file and access TIB SimHit branch
file = TFile('simevent.root')
events = file.Get('Events')
simHitBranch = events.GetBranch('PSimHit_r_TrackerHitsTIBLowTof.obj')

simHit = vector(PSimHit)()
simHitBranch.SetAddress(simHit)


histo = TH1F('tofhits', 'Tof of hits', 100, -0.5, 50)

### loop over all events and simHits
for event in loop(events):
    simHitBranch.GetEntry(event)
    for hit in loop(simHit):
        histo.Fill(hit.timeOfFlight())

### plot the histogram
c = TCanvas()
histo.Draw()
```

FW Plugins in Python

The event wrapper

```
ConstEventWrapper::ConstEventWrapper(const edm::Event& iEvent): event_(&iEvent) {}

void ConstEventWrapper::getByLabel(std::string const& iLabel, edm::GenericHandle& oHandle) const
{
  if(event_) {
    event_->getByLabel(iLabel,oHandle);
  }
}
```

```
bool PythonAnalyzer::analyze(edm::Event& iEvent, const edm::EventSetup& iSetup){
    using namespace boost::python;
    object main_module(( handle<>(borrowed(PyImport_AddModule("__main__"))) ));
    object main_namespace = main_module.attr("__dict__");
    main_namespace["event"] = object(edm::python::ConstEventWrapper(iEvent) );
    command = pyObjectName + ".filter(event)"
    object result((handle<>(PyRun_String(command.c_str(),
                                         Py_eval_input,
                                         main_namespace.ptr(),
                                         main_namespace.ptr()))));
}
```

How to load your analyzer

the cfg-file fragment

```
module myAnalyzer = PythonAnalyzer { className = "simpleAnalyzer" }
```

python analyzer

```
class simpleAnalyzer(object):

    # constructor
    def __init__(self, config):
        print "Init of simpleAnalyzer"

    # for the event loop
    def analyze(self, event, setup):
        event.getByLabel("foo")
        print "analyze loop"

    # called at destruction of the object
    def finalize(self)
        print "finalize"
```

Python interpreter singleton

```
namespace edm::python {

  class PythonInterpreter

  {
    public:
      void Method ();

    protected:
      PythonInterpreter ();

    private:
      PythonInterpreter (const Singleton& rSource)  { }

      PythonInterpreter& operator = (const PythonInterpreter& rSource) { }

      static PythonInterpreter * instance ();
      static void releaseInstance ();

      static PythonInterpreter * m_pInstance;
      static unsigned long m_ulReferenceCounter;

      friend SingletonHandleT <PythonInterpreter> :: SingletonHandleT ();
      friend SingletonHandleT <PythonInterpreter> :: SingletonHandleT
                (const SingletonHandleT&);
      friend SingletonHandleT <PythonInterpreter> :: ~SingletonHandleT ();
  };

} //namespace end
```

The exception handler

```
namespace edm::python {

  static void ErrorHandler(const std::string& iType){
     using namespace boost::python;
     PyObject *exc, *val, *trace;
     PyErr_Fetch(&exc,&val,&trace);
     handle<> hExc(allow_null(exc));
     if(hExc) { object oExc(hExc); }
     handle<> hVal(allow_null(val));
     handle<> hTrace(allow_null(trace));
     if(hTrace) { object oTrace(hTrace); }
     if(hVal) {
        object oVal(hVal);
        handle<> hStringVal(PyObject_Str(oVal.ptr()));
        object stringVal( hStringVal );

        //PyErr_Print();
        throw cms::Exception(iType) <<"PythonErrorHandle: "<< PyString_AsString \
                                      (stringVal.ptr())<<"\n";
     } else { throw cms::Exception(iType)<<" PythonErrorHandle: unknown reason\n"; }
  }

} //namespace end
```