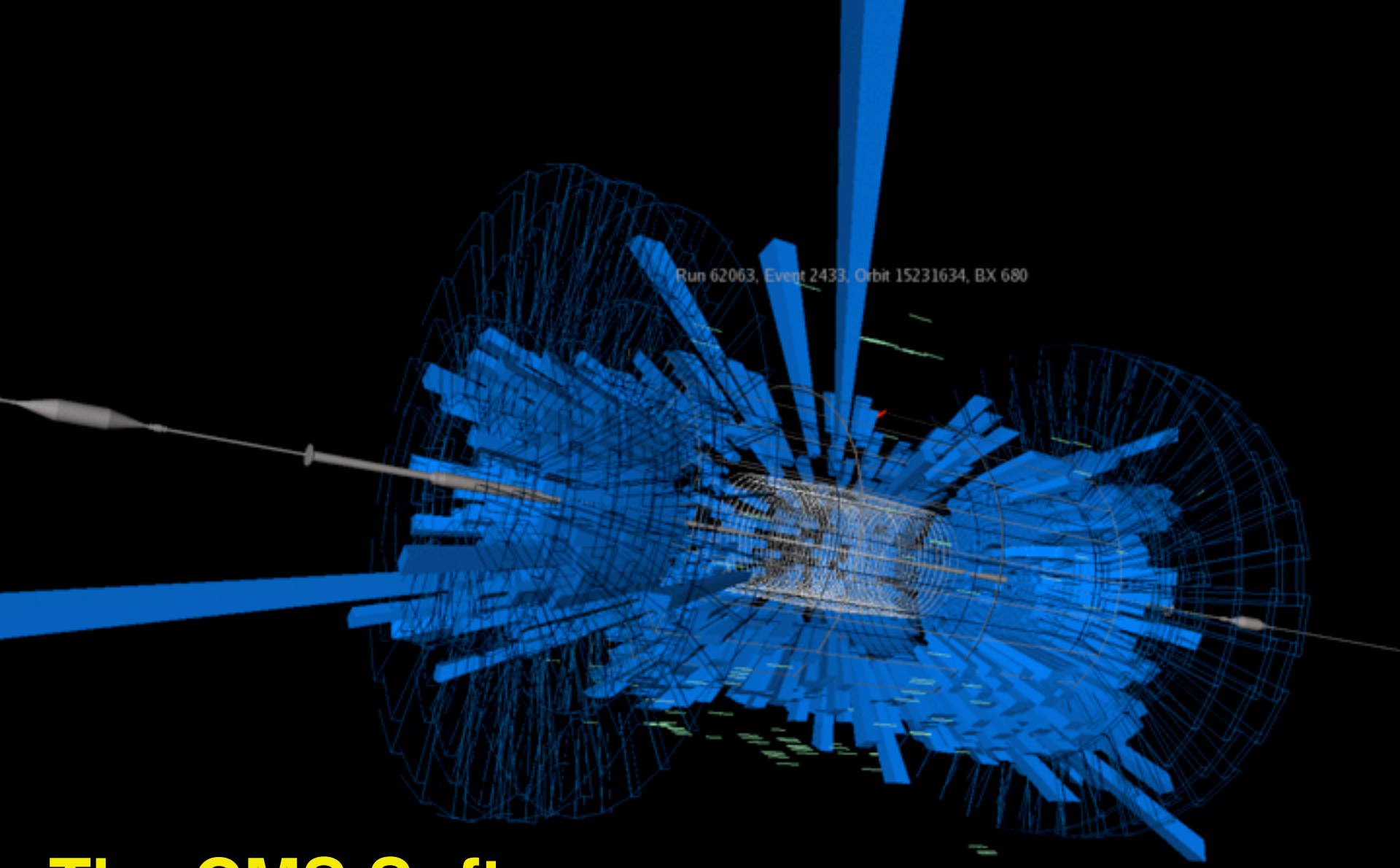


The CMS Software Framework CMSSW

Benedikt Hegner

Outline

- Framework basics
- The Data format
- Looking at Files
- Writing a framework module
- Running the module

- Places to look for information

- If there are things not covered:
  Feel free to ask in the discussion session

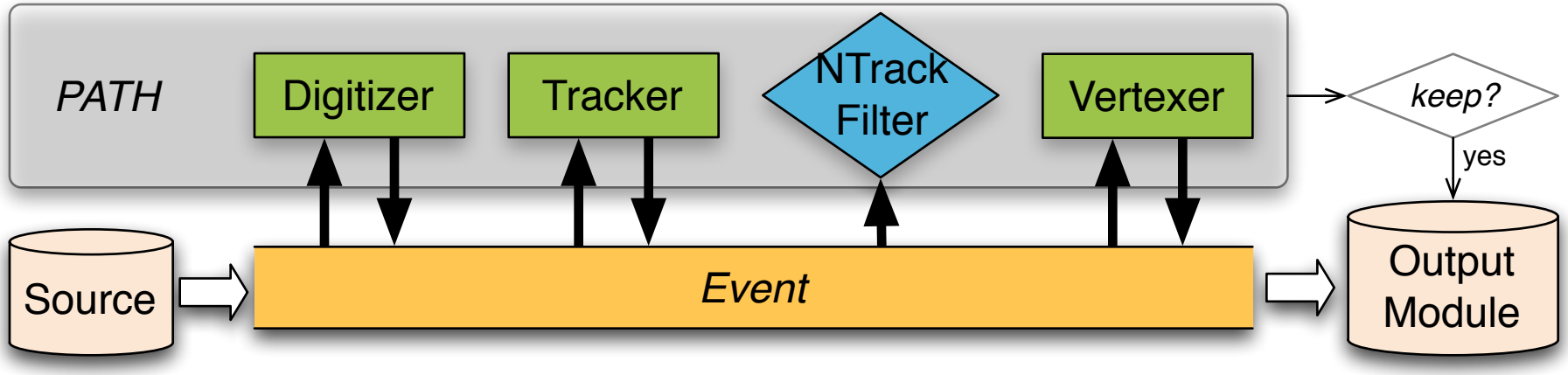
The CMSSW framework

Basic concepts

- One executable: *cmsRun*
- Event Data Model (EDM) based on the event:
  - Single entity in memory: edm::event container
  - Modular content
- Modular architecture

  Module: component to be plugged into cmsRun as unit of clearly defined event-processing functionality
- Steered via Python job configurations

Flow of the Data



- Source
- EDProducer
- EDFilter
- EDAnalyzer
- OutputModule

- Sequence
- Path
- EndPath
- Schedule

Communication among modules **only** via the event

Framework Modules

- EDAnalyzer
  - analyse objects from the event
- EDProducer
  - adds objects into the event
- EDFilter
  - can stop an execution path and put objects into the event

- EDLooper
  - For multi-pass looping over events (e.g. for alignment)
- OutputModule
  - Write events to a file. Can use filter decisions

Services and Setups

Examples of Framework Services and setups are
- Geometry, Calibration, MessageLogger

• Types are:
- Service
  Constantly available
- ESSource
  Provides data which have an IOV (Interval of Validity)
- ESProducer
  Creates Products when IOV changes

Topic of its own. No further details in this tutorial!

Steering file for cmsRun

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("EXAMPLE")
process.source = cms.Source("EmptySource")
process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(100) )

process.int = cms.EDProducer("IntProducer", ivalue = cms.int32(2) )

process.test = cms.EDAnalyzer("IntTestAnalyzer",
                              valueMustMatch = cms.untracked.int32(2)
                             )

process.Tracer = cms.Service("Tracer")
process.p = cms.Path( process.int * process.test)
```

```
$ cmsRun example_cfg.py
```

CMSSW File input and output

input

```
process.source = cms.Source("PoolSource",
  fileNames = cms.untracked.vstring("file:input.root")
)
```

PROCESS

output

```
process.out = cms.OutputModule("PoolOutputModule",
     fileName = cms.untracked.string("output.root")
)
```

```
outputCommands = cms.untracked.vstring(
      "keep *",
      "drop *_*_*_HLT",
      "keep FEDRawDataCollection_*_*_*"
)
```

The Data Formats

What is stored in the event files?

- RAW:
  Data like they come from the detector

- RECO (Reconstruction):
  Output of the event reconstruction

- AOD (Analysis Object Data):
  Subset of data needed for
  standard analysis

- RAWSIM, RECOSIM, AODSIM:
  with additional simulation information

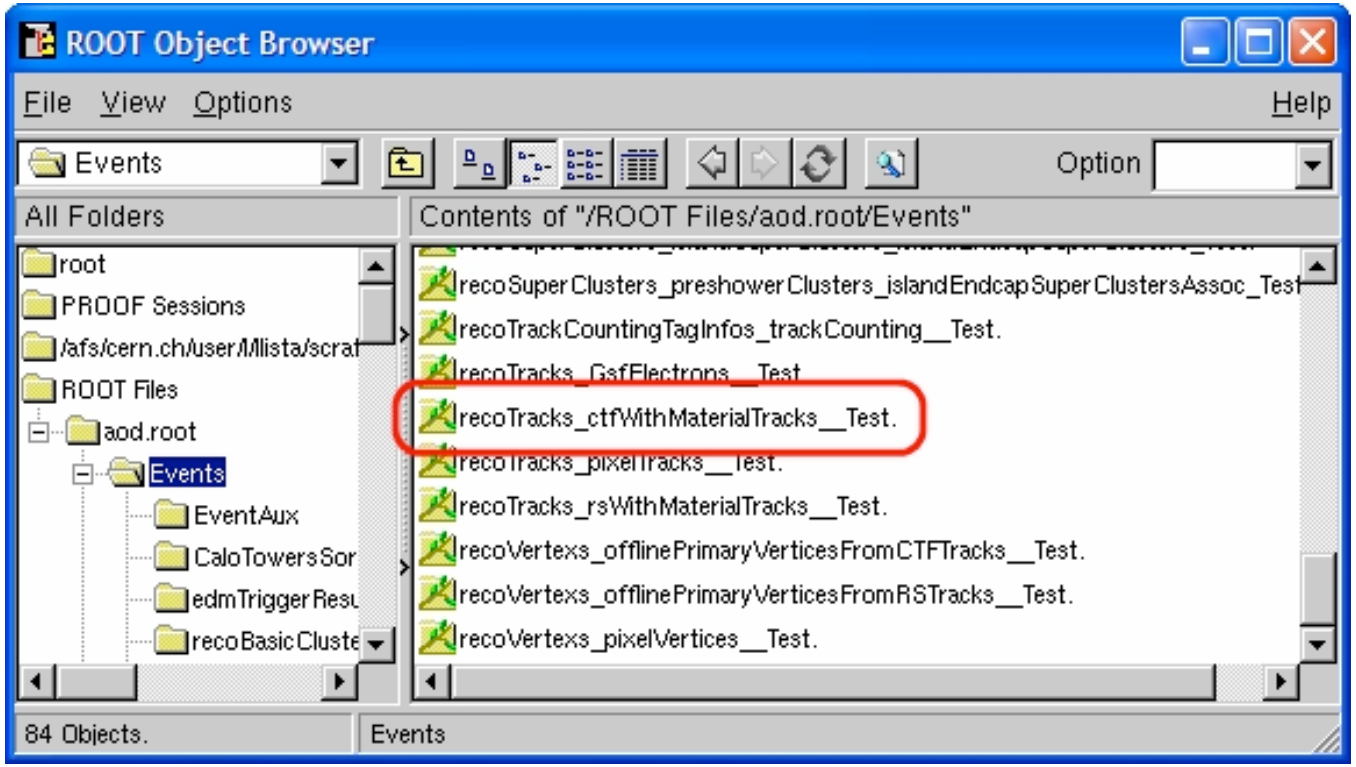
Processing steps

There are quite a few steps in simulation

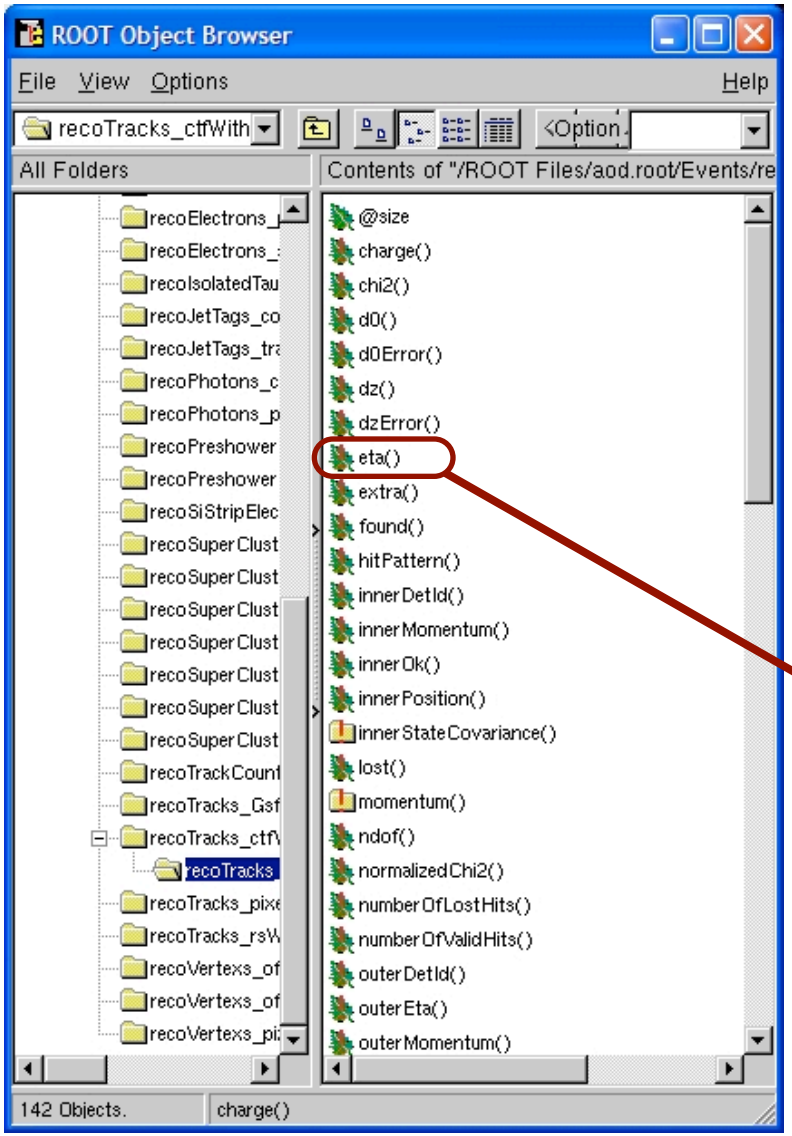
- GEN (generator + generator level objects like genJets)
- SIM (energy deposits in the detector material)
- SIMDIGI (actual detector response + noise)
- L1 (Level1 trigger)
- DIGI2RAW (conversion into RAW data format)
- HLT (High Level Trigger)
- RAW2DIGI (conversion from RAW to digi data format)
- RECO (reconstruction)

Files can be opened with ROOT

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

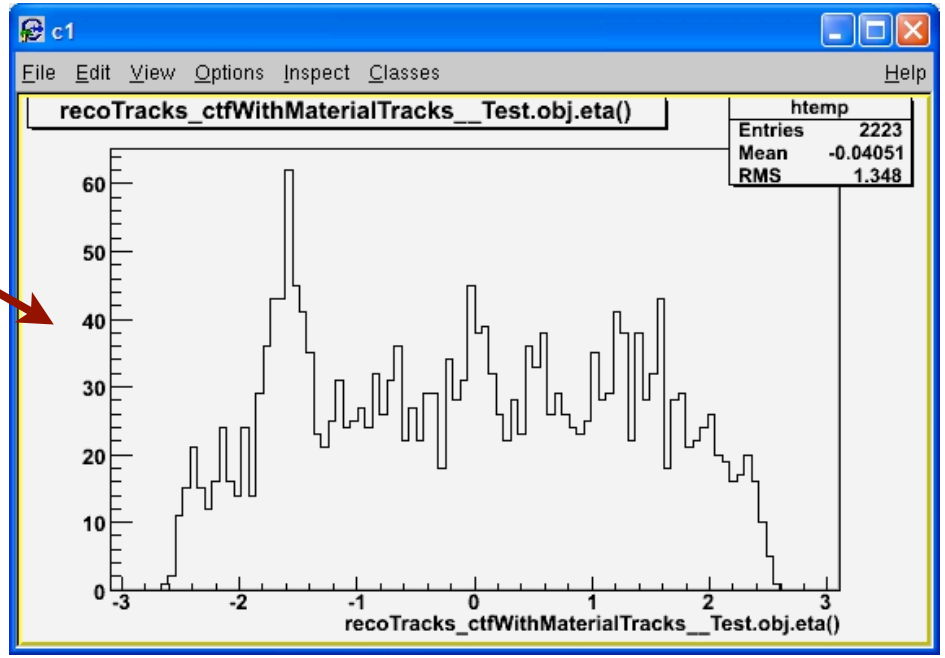


FWLite gives access to classes



Automatic library loading

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



Accessing Event Data I

Data inside the event are called “Product”

moduleLabel : productInstanceLabel : processName

```
# 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 );


...
```

Accessing Event Data II

Output of module EventContentAnalyzer:

```
++Event     0 contains 161 products with friendlyClassName, moduleLabel and productInstanceName:
...
++recoElectrons "siStripElectronToTrackAssociator" "siStripElectrons"
++recoGenJets "Fastjet10GenJets" ""
++recoGenJets "Fastjet10GenJetsNoNu" ""
++recoGenJets "Fastjet6GenJets" ""
++recoGenJets "Fastjet6GenJetsNoNu" ""
++recoGenJets "Kt10GenJets" ""
++recoGenJets "Kt10GenJetsNoNu" ""
++recoGenJets "iterativeCone5GenJets" ""
++recoGenJets "iterativeCone5GenJetsNoNu" ""
++recoGenJets "iterativeCone7GenJets" ""
++recoGenJets "iterativeCone7GenJetsNoNu" ""
++recoGenMETs "genMet" ""
++recoGenMETs "genMetNoNu" ""
...
```

```
Handle<GenJetCollection> genJets;
Event.getByLabel("FastJet6GenJets",genJets );
```

another option:
EdmDumpEventContent <filename>

Provenance Tracking

The history of each single product in the event is stored in the "provenance"

```
...

Module: caloTowers Rec
 PSet id:e03ccfff88a2fd4ed3c2b9bd8261000b
 products: {
  recoCandidatesOwned_caloTowers__Rec.
}
 parameters: {
  @module_label: string tracked  = 'caloTowers'
  @module_type: string tracked  = 'CaloTowerCandidateCreator'
  minimumE: double tracked  = -1
  minimumEt: double tracked  = -1
  src: InputTag tracked  = towerMaker::
}

...
```

edmProvDump <filename>

Visualization Tools

Visualization Tools

In CMS there are two complementary visualization tools

- *IGUANA* is the full blown visualization using the full framework and all available conditions and geometry information
- Used in commissioning, algorithm development and for nice plots

  `https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookEventDisplay`

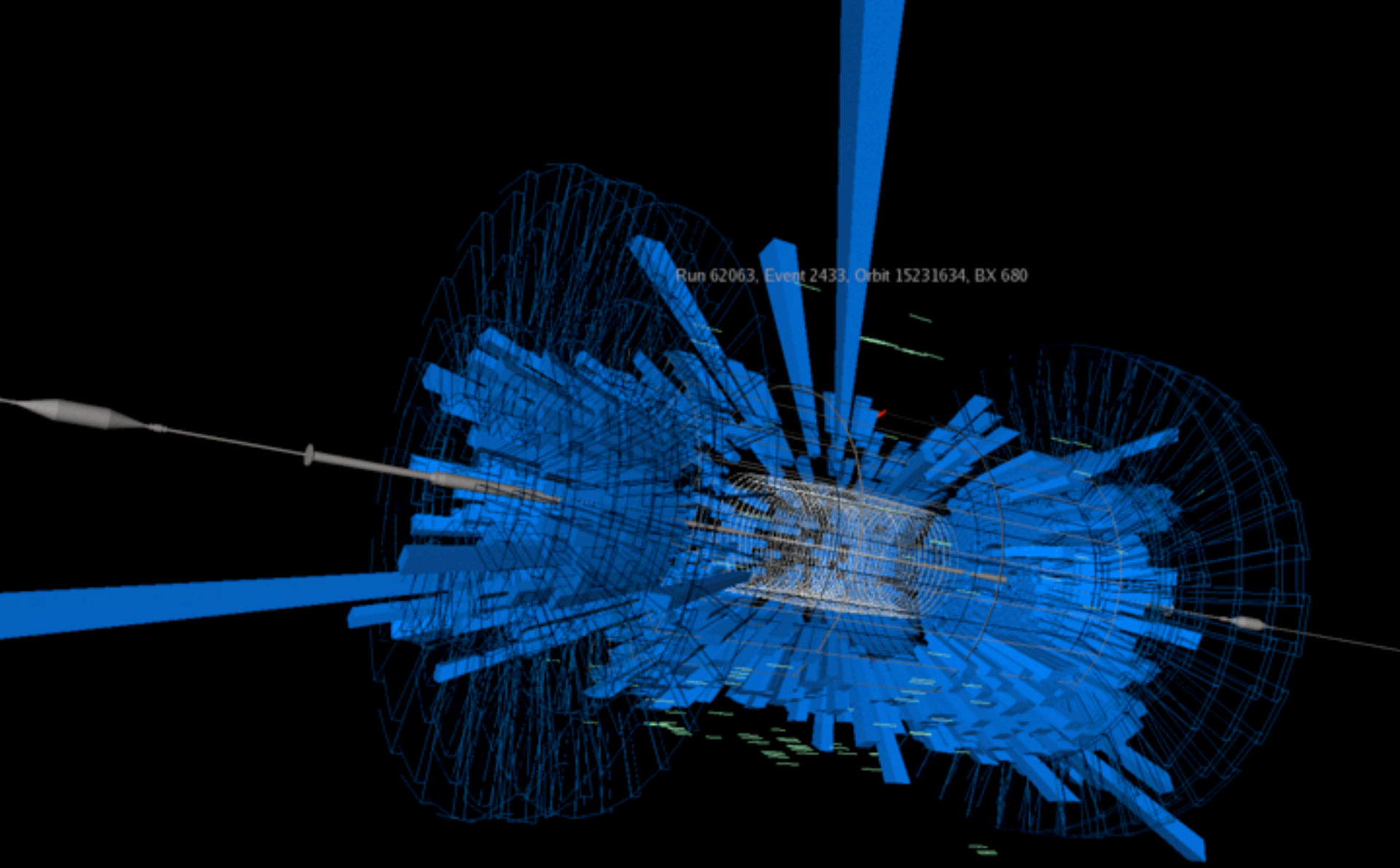
- *Fireworks* is the light weight event display for analysis which can be installed on your laptop:

  `https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookFireworks`

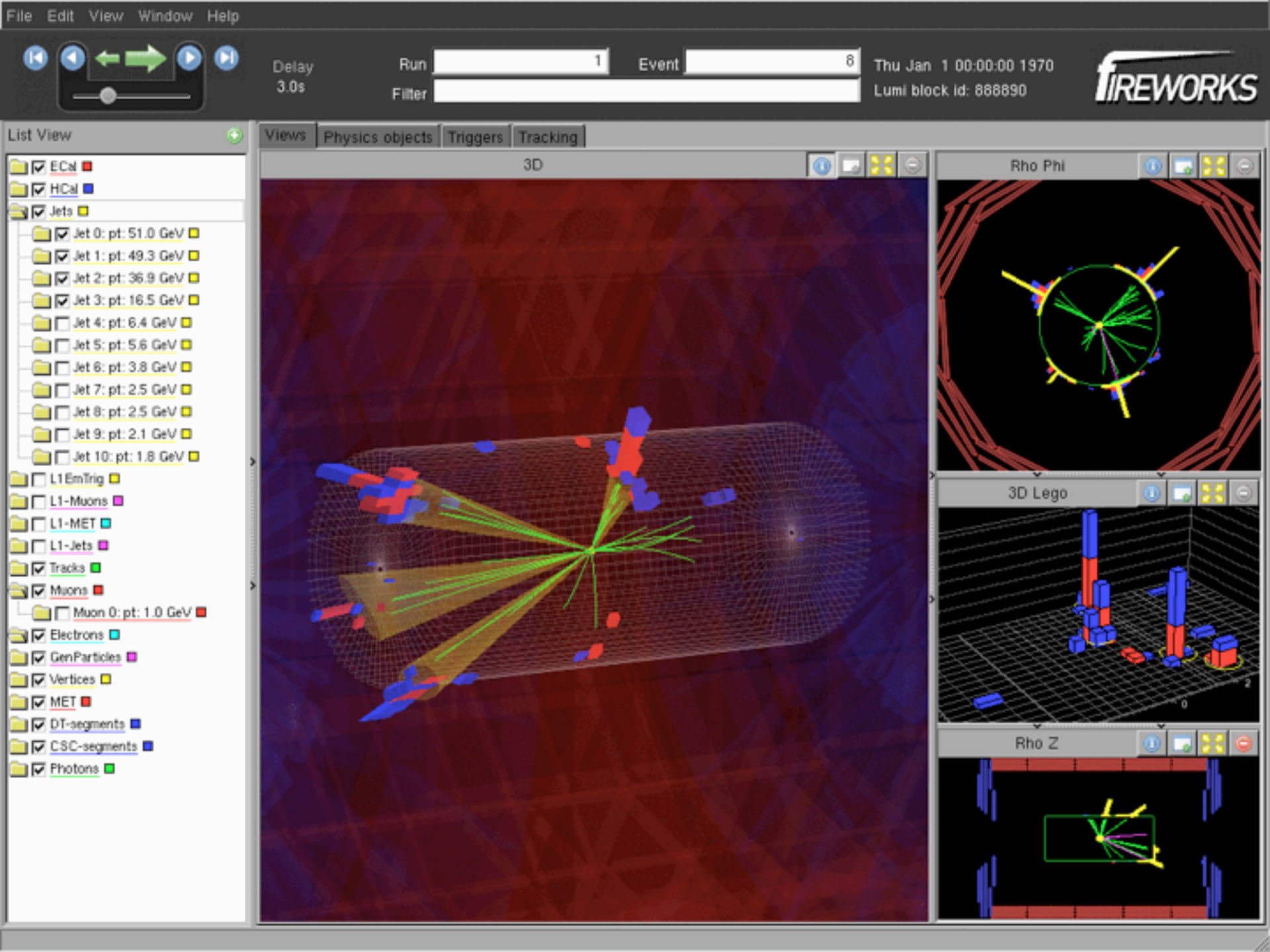
- Try out the video tutorial!

  `http://cern.ch/cms-sdt/fireworks/demo.mov`

- If in doubt, you probably want to use Fireworks!



IGUANA


File Edit View Window Help
Delay
3.0s
Run
1
Event
8
Filter
Thu Jan 1 00:00:00 1970
Lumi block id: 888890
FIREWORKS
List View
ECal
HCal
Jets
Jet 0: pt: 51.0 GeV
Jet 1: pt: 49.3 GeV
Jet 2: pt: 36.9 GeV
Jet 3: pt: 16.5 GeV
Jet 4: pt: 6.4 GeV
Jet 5: pt: 5.6 GeV
Jet 6: pt: 3.8 GeV
Jet 7: pt: 2.5 GeV
Jet 8: pt: 2.5 GeV
Jet 9: pt: 2.1 GeV
Jet 10: pt: 1.8 GeV
L1EmTrig
L1-Muons
L1-MET
L1-Jets
Tracks
Muons
Muon 0: pt: 1.0 GeV
Electrons
GenParticles
Vertices
MET
DT-segments
CSC-segments
Photons
Views
Physics objects
Triggers
Tracking
3D
Rho Phi
3D Lego
Rho Z

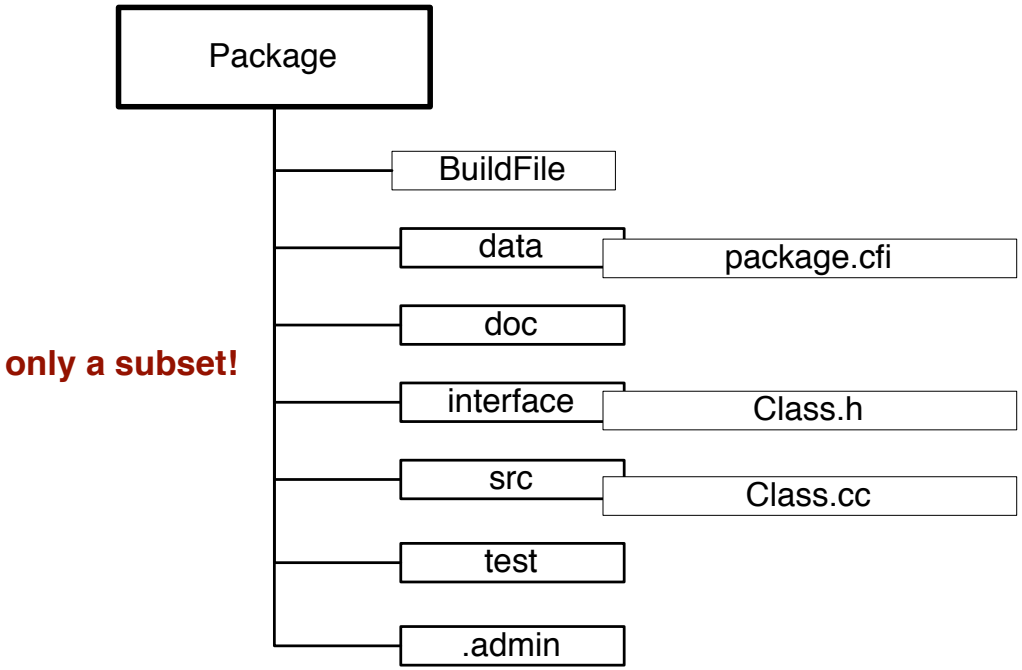
Writing your own framework module

The Release Area

SCRAM = Software release and management tool

```
Project/Subproject/Package

CMSSW_2_2_13/src/GeneratorInterface/ToprexInterface
```



only a subset!

Preparing the environment

```
creating your local area
 $ cmsrel CMSSW_2_2_13
 [...]
 $ cd CMSSW_2_2_13/src

setting runtime variables
 $ cmsenv

accessing the cvs server
 $ addpkg Subproject/Package
```

What module type to write

EDAnalyzer

- Reading data only
- Creating histograms
- the standard use case

EDProducer

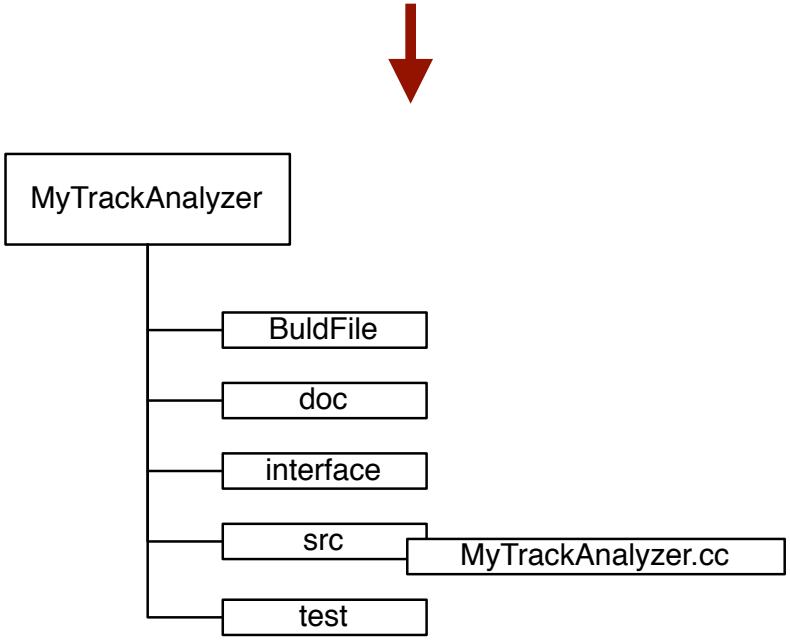
- You want to create new products
- You want to share your reconstruction code with others
- You want to make different algorithms plugable

EDFilter

- you want to know if an object could be produced
- you want to control the analysis flow or make skimming

Writing an EDAnalyzer

```
$ cd CMSSW_2_2_13/src
$ mkdir Tutorial
$ cd Tutorial
$ mkedanlzr -list
$ mkedanlzr -track MyTrackAnalyzer
```



The 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("modulelabel",tracks);

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

```
DEFINE_FWK_MODULE(MyTracks);
```

Building the Example I

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

BuildFile

```
<use name=FWCore/Framework>
<use name=FWCore/PluginManager>
<use name=FWCore/ParameterSet>
<flags EDM_PLUGIN=1>
<use name=DataFormats/TrackReco>
<export>
   <lib name=TutorialMyTrackAnalyzer>
   <use name=FWCore/Framework>
   <use name=FWCore/PluginManager>
   <use name=FWCore/ParameterSet>
   <use name=DataFormats/TrackReco>
</export>
```

Building the Example II

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

```
Reading cached build data
Scanning src/Tutorial/MyTrackAnalyzer/BuildFile

Entering Package Tutorial/MyTrackAnalyzer
Entering library rule at Tutorial/MyTrackAnalyzer
>> Compiling /build/hegner/CMSSW_1_6_12/src/Tutorial/MyTrackAnalyzer/src/MyTrackAnalyzer.cc
>> Building shared library tmp/slc4_ia32_gcc345/src/Tutorial/MyTrackAnalyzer/src/
TutorialMyTrackAnalyzer/libTutorialMyTrackAnalyzer.so
/usr/bin/ld: skipping incompatible /usr/lib64/libnsl.so when searching for -lnsl
...
/usr/bin/ld: skipping incompatible /usr/lib64/libc.a when searching for -lc
@@@@ Checking shared library for missing symbols: libTutorialMyTrackAnalyzer.so
/usr/bin/ld: skipping incompatible /usr/lib64/libm.so when searching for -lm
...
/usr/bin/ld: skipping incompatible /usr/lib64/libc.a when searching for -lc
@@@@ ----> OK, shared library FULLY-BOUND (no missing symbols): libTutorialMyTrackAnalyzer.so
@@@@ Checking shared library load: libTutorialMyTrackAnalyzer.so
@@@@ ----> OK, shared library loaded successfully: libTutorialMyTrackAnalyzer.so
Leaving library rule at Tutorial/MyTrackAnalyzer
--- Registered EDM Plugin: TutorialMyTrackAnalyzer
Leaving Package Tutorial/MyTrackAnalyzer
>> Package MyTrackAnalyzer built
```

Building other code skeletons

```
# Create a code skeleton
 $ mkskel <name>
 $ mkedanlzr <name> / mkedanlzr <-template> <name>
 $ mkedprod <name>
 $ mkedfltr <name>

# Other useful tools
 $ edm*
```

TFileService

Opening a ROOT file inside cmsRun is very error prone.
But there is an alternative...

```
// access the TFileService
edm::Service<TFileService> fs;

// create your histogram
TH1F * h_pt = fs->make<TH1F>( "pt"  , "p_{t}", 100,  0., 100. );

// fill it
h_pt->Fill( pt );

// create subdirectories if you like
TFileDirectory subDir = fs->mkdir( "mySubDirectory" );
```

```
# make the TFileService known to the config
process.TFileService = cms.Service("TFileService",
                                   fileName = cms.string("histo.root")
                                  )
```

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

HistoAnalyzer

Getting histograms without writing C++ code

```
plotJets = cms.EDAnalyzer("CandViewHistoAnalyzer",
    src = cms.InputTag("iterativeCone5CaloJets"),
    histograms = cms.VPSet(
      cms.PSet(
        itemsToPlot = cms.untracked.int32(5),  # plots the first 5 jets
        min = cms.untracked.double(0.0),
        max = cms.untracked.double(200),
        nbins = cms.untracked.int32(50),
        name = cms.untracked.string("jet %d E_{T} [GeV/c]"),
        description  = cms.untracked.string("jet_%d_et"),
        plotquantity = cms.untracked.string("et")
      )
    )
)
```

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

CMSSW Config Files

Definition of terms: configuration file

- Controls the final job to be run
- Written in Python
- Contains a **`cms.Process`** object named **`process`**
- Usually placed in a package's **`python/`** or **`test/`**
- Can be checked for completeness doing

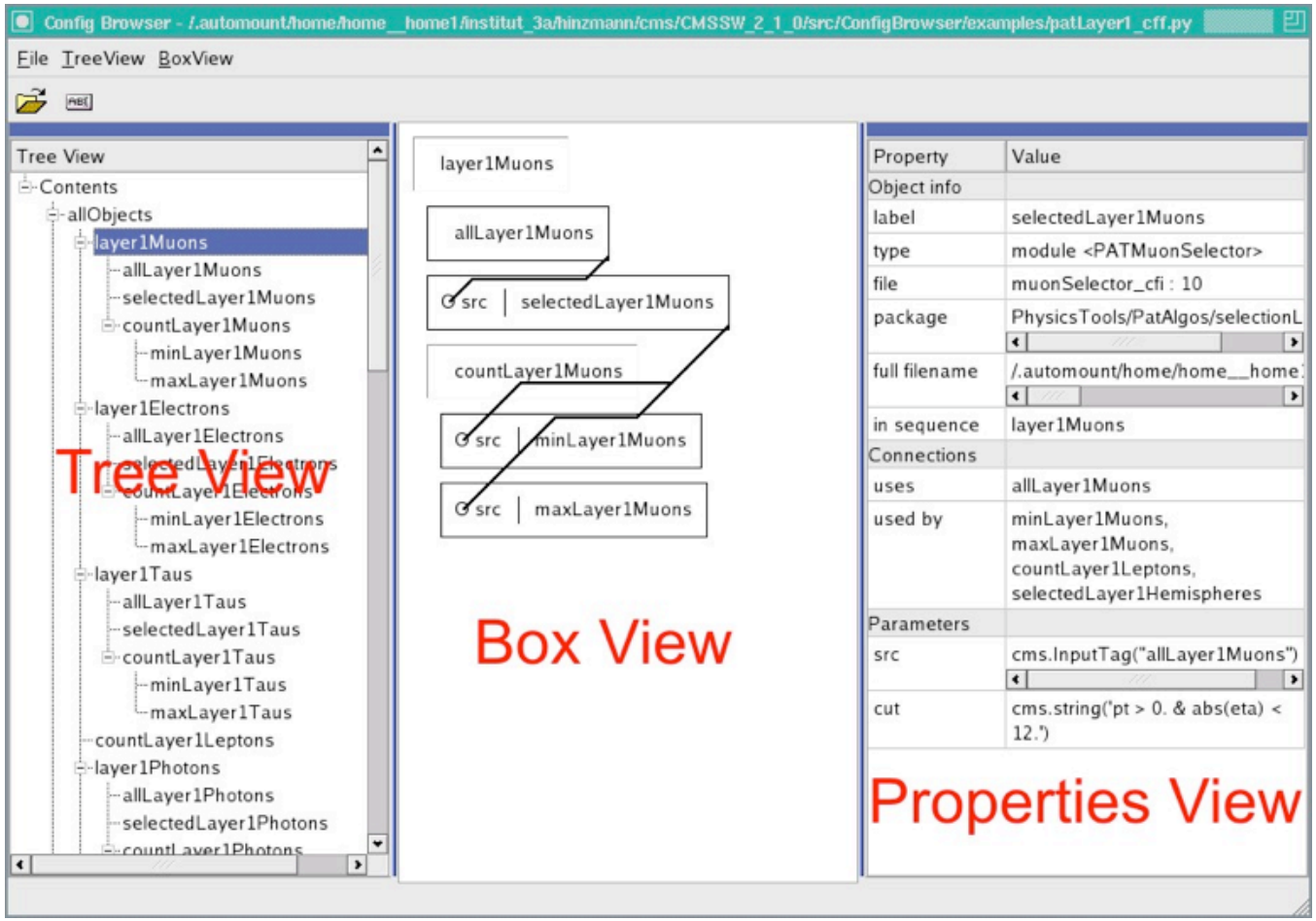
```
python myExample_cfg.py
```
(*Python interpreter*)

- Can be run using cmsRun

```
cmsRun myExample_cfg.py
```

Process Object:
"the protagonist" of the configuration

ConfigBrowser



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

Definition of terms: Python module

- A python file that is meant to be included by other files
- Placed in **`Subsystem/Package/python/`** or a subdirectory of it

- Naming conventions
  - Definition of a single object: **`_cfi.py`**
  - A configuration fragment: **`_cff.py`**
  - A full process definition: **`_cfg.py`**

- To make your module visible to other python modules:
  - Be sure your SCRAM environment is set up
  - Go to your package and do scram b or scram b python
  - Needed only once

- Correctness of python config files is checked on a basic level every time scram is used.

How to import objects

- To fetch all modules from some other module into local namespace

  ```
  from Subsystem.Package.Foo_cff import *
  ```

  (looks into Subsystem/Package/python/Foo_cff.py)

- To load everything from a python module into your process object you can say:

  ```
  process.load('Subsystem.Package.Foo_cff')
  ```

- Don't forget that all imports create references, not copies:

changing an object at one place
changes the object at other places

Cloning

- Sometimes you need to add a module which has almost the same parameter as another one
- You can copy the module and change the parameters that need to be modified

```
import ElectroWeakAnalysis.ZReco.zToMuMu_cfi as zmumu
    zToMuMuGolden = zmumu.zToMuMu.clone(
    massMin = cms.double(40)
    )
```

- Changing while cloning should be preferred wrt clone + later replace as it is a much safer practice.

Sequences and Paths

Sequence:

• Defines an execution order and acts as building block for more complex configurations and contains modules or other sequences.

```
trDigi = cms.Sequence(siPixelDigis + siStripDigis)
```

Path:

• Defines which modules and sequences to run.

```
p1 = cms.Path(pdigi * reconstruction)
```

EndPath:

• A list of analyzers or output modules to be run after all paths have been run.

```
outpath = cms.EndPath(myOutput)
```

Sequence Operators

"+" as 'follows':

- Use if the input of the previous module/sequence is not required

```
trDigi = cms.Sequence(siPixelDigis + siStripDigis)
```

"*" as 'depends on':

- If module depends on previously created products

```
p1 = cms.Path(pdigi * reconstruction)
```

- Enforced and checked by scheduler

Combining:

- By using () grouping is possible

```
(ecalRecHits + hcalRecHits) * caloTowers
```

Filters in Paths

- Each path corresponds to a trigger bit
- When an EDFilter is in a path, returning *False* will cause the path to terminate
- Two operators ~ and - can modify this.

1. ~ means not. The filter will only continue if the filter returns *False*.
2. - means to ignore the result of the filter and proceed regardless

```
jet500_1000 = cms.Path(
  ~jet1000filter + jet500filter + jetAnalysis
  )
```

The Output

- Standard event contents are defined centrally:

```
Configuration.EventContent.EventContent_cff
```

- Output files are written via the PoolOutputModule

```
cms.OutputModule("PoolOutputModule",
    outputCommands = RECOEventContent.outputCommands,
    fileName = cms.untracked.string('TTbar_cfi_GEN_SIM_DIGI.root'),
    SelectEvents = cms.untracked.PSet(
        SelectEvents = cms.vstring('*Electron:HLT')
    )
)
```

- Details on selectEvents in the SWGuide:

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

Helper utilities

Finding Code

A usual annoyance - where the he... is *this* defined?

- If you already know where to look - cvs browser:

  `http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/CMSSW`

- In all other 99% of the cases - lxr browser:

  `http://cmslxr.fnal.gov/lxr/`

- or simply at command line:

  `cmsglimpse <string>`

Handling Source Code

- Add a package from the cms repository

```
addpkg PhysicsTools/Utilities  [tag]
```

- If no tag is given the default one from the release is taken
- List which packages are there

```
showtags -r
```

```
Test Release based on: CMSSW_2_1_7
Base Release in: /afs/cern.ch/cms/sw/slc4_ia32_gcc345/cms/cmssw/CMSSW_2_1_7
Your Test release in: /build/hegner/CMSSW_2_1_7
--- Tag ---     --- RelTag --- -------- Package --------
V08-09-16       V08-09-15       FWCore/ParameterSet
V02-08-04       V02-08-04       RecoJets/JetProducers
--------------------------------------
total packages: 2 (2 displayed)
```

What's going on?

- Quick'n'dirty way:

```
std::cout << "here I am" << std::endl;
```

- There is a much better way:

```
cms.Service('Tracer')
```

```
++++source
 Begin processing the 1st record. Run 1, Event 1, LumiSection 1 at
 09-Sep-2008 10:30:22 CEST
 ++++finished: source
 ++++ processing event:run: 1 event: 1 time:5000000
 ++++++ processing path:generation_step
 ++++++++ module:randomEngineStateProducer
 ++++++++ finished:randomEngineStateProducer
 ++++++++ module:VtxSmeared
 ++++++++ finished:VtxSmeared
 ...
```

EventContentAnalyzer

- At every position in the schedule one can inspect what products can be accessed:

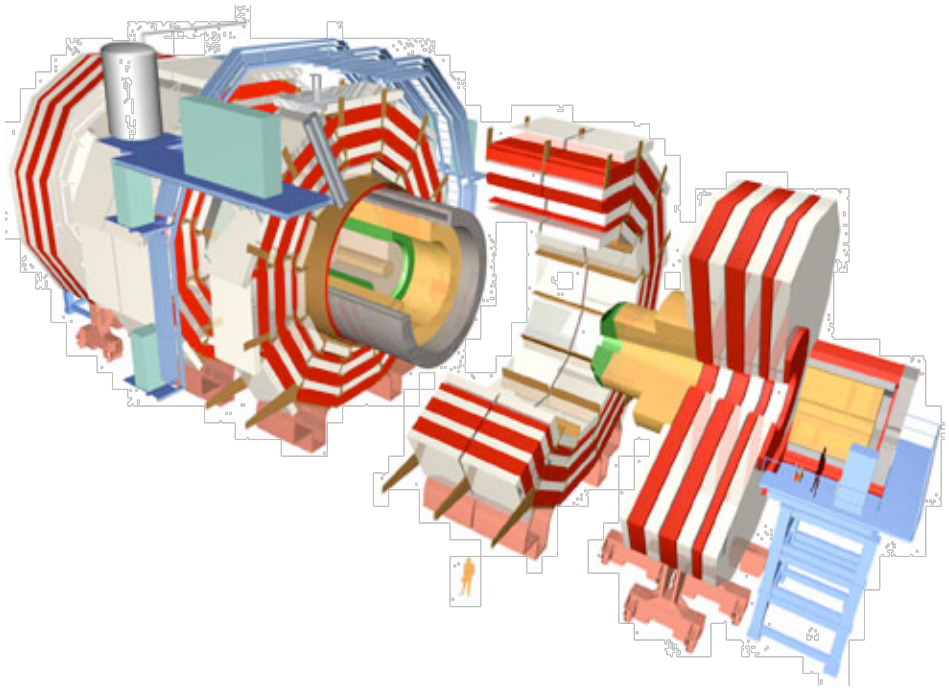
```
process.dump =
cms.EDAnalyzer('EventContentAnalyzer')
process.p = cms.Path(... + dump + ...)
```

- Given a file a similar thing can be done at command line:

```
edmDumpEventContent <filename>
```

Support and Documentation

- There is the WorkBook:
  https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook
- The SWGuide:
  https://twiki.cern.ch/twiki/bin/view/CMS/SWGuide
- LXR:
  http://cmslxr.fnal.gov/lxr/
- Many, many hypernews lists
- CMSSW “office hours”:
  Wednesday, 14:00-16:00
  40-3-B20



Questions?
Comments?
Need a break?

BACKUP SLIDES

Timing Problems

- You can use the framework to quickly identify how fast your process and single modules are

  `cms.Service('Timing')`

- Useful information if you plan your private production

- Further details:

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

- Don't optimise before you *really* have a problem

Tips'n'tricks

Starting easy

Working with CMSSW starts with eval `scram ...` an, no?

- Setting up your project area:

```
cmsrel CMSSW_2_1_17
```

- Setting up your environment:

```
cd CMSSW_2_1_17/src
cmsenv
```

- These are only simple shortcuts for the much longer scram commands

Getting development up to speed

The compilation is too slow!

- Most machines have multiple processors and cores, so let's use them:

```
scram b -jN
```

- In first order the compiler then uses N cores
- If you are sharing the machine with other users - take care that you don't accidentally block the whole machine for yourself!

Some numbers

- 2.6 GHz * 8 Cores translates into 2.5 h for the whole CMSSW

Code quality and runtime inspection

Ignominy tools

- ignominy for checking dependencies and complexity

  https://ignominy.web.cern.ch/ignominy/

- igprof for memory and performance measurements

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

- igtrace for memory allocations, traces etc

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

hit by dependencies...

- Sometimes you change a package. It compiles nicely, but cmsRun crashes with segmentation violations because it cannot find symbols.
- Or you compile and scram returns something like:

```
tmp/slc4_ia32_gcc345/src/PhysicsTools/RecoUtils/plugins/PhysicsToolsRecoUtilsPlugins/
libPhysicsToolsRecoUtilsPlugins.so: undefined reference to
`edmplugin::PluginFactory<EventSelector* ()(edm::ParameterSet const&)>::get()'
collect2: ld returned 1 exit status
gmake: *** [tmp/slc4_ia32_gcc345/src/PhysicsTools/RecoUtils/plugins/PhysicsToolsRecoUtilsPlugins/
libPhysicsToolsRecoUtilsPlugins.so] Error 1
```

- This means you made a major change to a package and didn't recompile the packages which would be affected by these changes.
- There is a tool to help with this:

```
/build/hegner/CMSSW_2_1_7/src > checkdeps
Packages to check out and compile:
None
```

Python and ROOT

```
from PhysicsTools.PythonAnalysis import *
from ROOT import *

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

events = EventTree("reco.root")

# book a histogram
histo = TH1F("photon_pt", "Pt of photons", 100, 0, 300)

# event loop
for event in events:
      photons = event.photons  # uses aliases
      print "  Number of photons in event %i: %i" % (event, len(photons))

      for photon in photons:
          if photon.eta() < 2:
              histo.Fill(photon.pt())
```

A configuration example

A standard config file

```
process = cms.Process('DIGI')

# import of standard configurations
process.load('Configuration/StandardSequences/Services_cff')
    ...

process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(1)
)

process.options = cms.untracked.PSet(
    Rethrow = cms.untracked.vstring('ProductNotFound')
)

# Input source
process.source = cms.Source("PythiaSource", ... )

# Output definition
process.output = cms.OutputModule("PoolOutputModule", ...)
# Path and EndPath definitions
process.generation_step = cms.Path(process.pgen)
    ...

# Schedule definition
process.schedule = cms.Schedule(process.generation_step,...)
```

Load the general setup - 1/2

- The geometry to use (ideal, pilot1, pilot2)

```
GeometryPilot2_cff
```

- The magnetic field (ideal, 0.0, 3.8 or 4 Tesla)

```
MagneticField_38T_cff
```

- The conditions can be fake or frontier (ideal, startup)

```
FrontierConditions_GlobalTag_cff
GlobalTag.globaltag = 'STARTUP_V5::All'

FakeConditions_cff
```

loads from Configuration/StandardSequences

Load the general setup - 2/2

- The message logger defaults

  `FWCore/MessageService/MessageLogger_cfi`

- The standard services as TFileService, DQMStore, particle data table, random number generator

  `Services_cff`

The Generation

- The MC input

```
process.source = ...
```

- The vertex smearing scenario (unsmeared, early collision,...)

```
VtxSmearedEarly10TeVCollision_cff
```

- The generator sequence pgen (genParticles, genJets, ...)

```
Generator_cff
```

- Put it in a path

```
process.generation_step =
cms.Path(process.pgen)
```

The Simulation

- The simulation modules

```
Sim_cff
```

- The pileup scenario (NoPileUp, LowLumiPileUp,...)

```
MixingNoPileUp_cff
```

- Put it in a path

```
process.simulation_step =
cms.Path(process.psim)
```

- All the other steps (DIGI,L1,DIGI2RAW, ...) work similar...

The Output

- Standard event contents are defined centrally:

  `Configuration.EventContent.EventContent_cff`

- Output files are written via the PoolOutputModule

```
cms.OutputModule("PoolOutputModule",
    outputCommands = RECOEventContent.outputCommands,
    fileName = cms.untracked.string('TTbar_cfi_GEN_SIM_DIGI.root'),
    SelectEvents = cms.untracked.PSet(
        SelectEvents = cms.vstring('*Electron:HLT')
    )
)
```

- Details on selectEvents in the SWGuide:

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

Exception handling

- The handling of exceptions can be modified (here: if a product is missing, stop the job):

```
process.options = cms.untracked.PSet(
    Rethrow = cms.untracked.vstring('ProductNotFound')
)
```

Category Name	Default Action
ProductNotFound	Skip Event
NoProductSpecified	Rethrow (stops the job)
InsertFailure	Skip Event
Configuration	Rethrow (stops the job)
LogicError	Rethrow (stops the job)
InvalidReference	Skip Event
EventTimeout	Skip Event
EventCorruption	Skip Event
FileInPathError	Rethrow (stops the job)
NotFound	Skip Event

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

edmPythonSearch

- A “grep”-like syntax to search for identifiers within imported files

```
> edmPythonSearch minPt Reconstruction_cff
```

...
RecoMuon.MuonIdentification.muons_cfi (line: 19) : minPt = cms.double(1.5),
...

edmPythonTree

- Gives an indented dump of which files are included by which files

(initial version for the old configs by Karoly Banicz and Sue Anne Koay)

```
> edmPythonTree Reconstruction_cff
```

```
+ Simulation_cff
    + Configuration.StandardSequences.Digi_cff
            + SimCalorimetry.Configuration.SimCalorimetry_cff
...
```

Python

- The Python interpreter helps you inspecting your configs

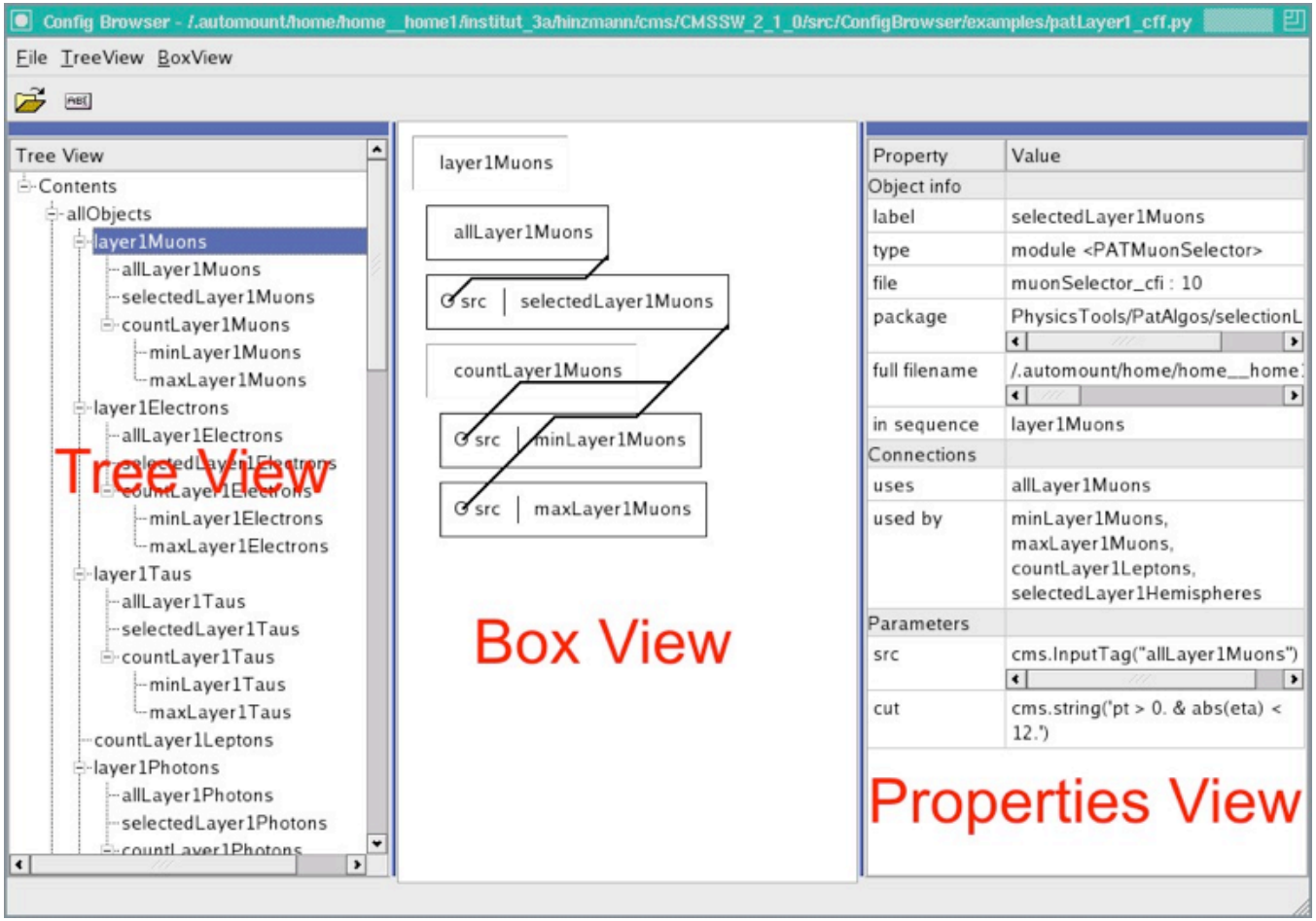
```
> python -i MyJob_cfg
```

- Simple commands will bring you forward

```
>>> process.aModule
>>> process.aModule._filename
>>> process.aModule._lineNumber
>>> print process.dumpPython()
>>> help(process)
```

- Ctrl-D shuts it down

PythonConfigBrowser



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

Using a PSet

- Give the PSet name directly after module type
- Has to happen before the named parameters

```
KtJetParameters = cms.PSet(
    strategy = cms.string("Best")
    )
ktCaloJets = cms.EDProducer("KtCaloJetProducer",
                            KtJetParameters,
                            coneSize = cms.double(0.7)
                            …
```

As parameters get copied a later change of KtJetParameters will not get picked up

Memory Problems

- If you get bad_alloc problems or you just observe increasing memory needs, the framework can help you getting a first idea

```
cms.Service('SimpleMemoryCheck',
            ignoreTotal = cms.untracked.int32(1)
)
```

- One example for such a problem:

```
++++-w MemoryIncrease: CSCRecHit2DProducer:csc2DRecHits
    28-May-2007 08:52:48 CEST Run: 1 Event: 1
Memory increased from VSIZE=763.04MB and RSS=615.809MB
to VSIZE=763.04MB and RSS=615.813MB
```

- For the real big problems use valgrind (http://valgrind.org) or igtools (see later)