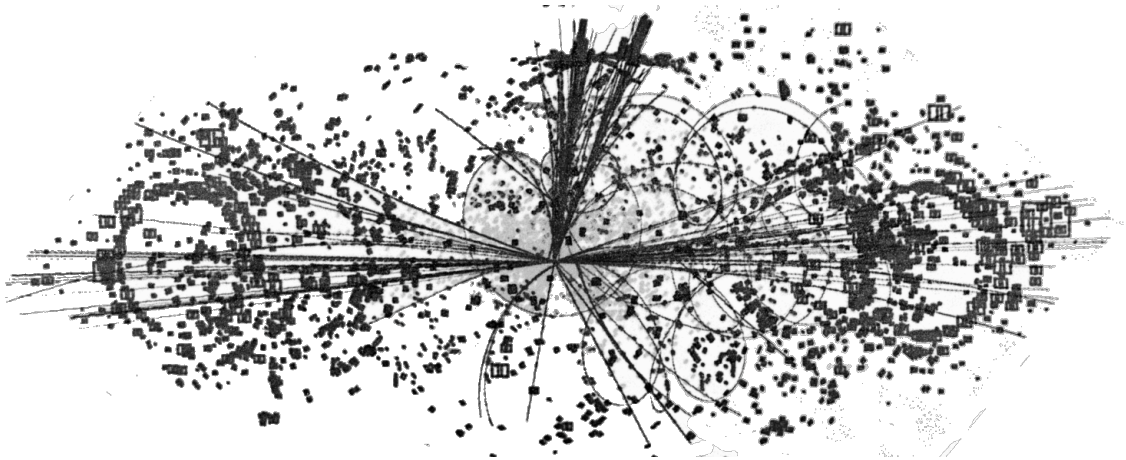


Introduction to the CMS software framework

Benedikt Hegner

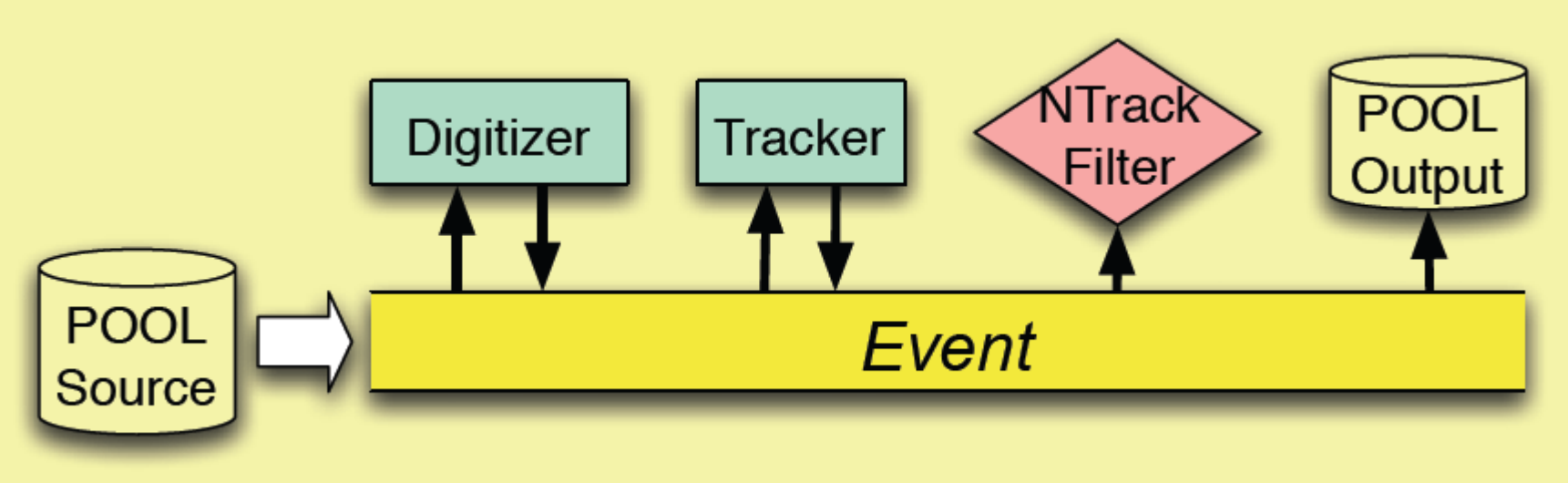
Outline

- Framework basics
- The Data format
- Configuring a CMSSW job
- Writing a framework module
- Tips'n'tricks

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
  - 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

This all gets steered by configuration files

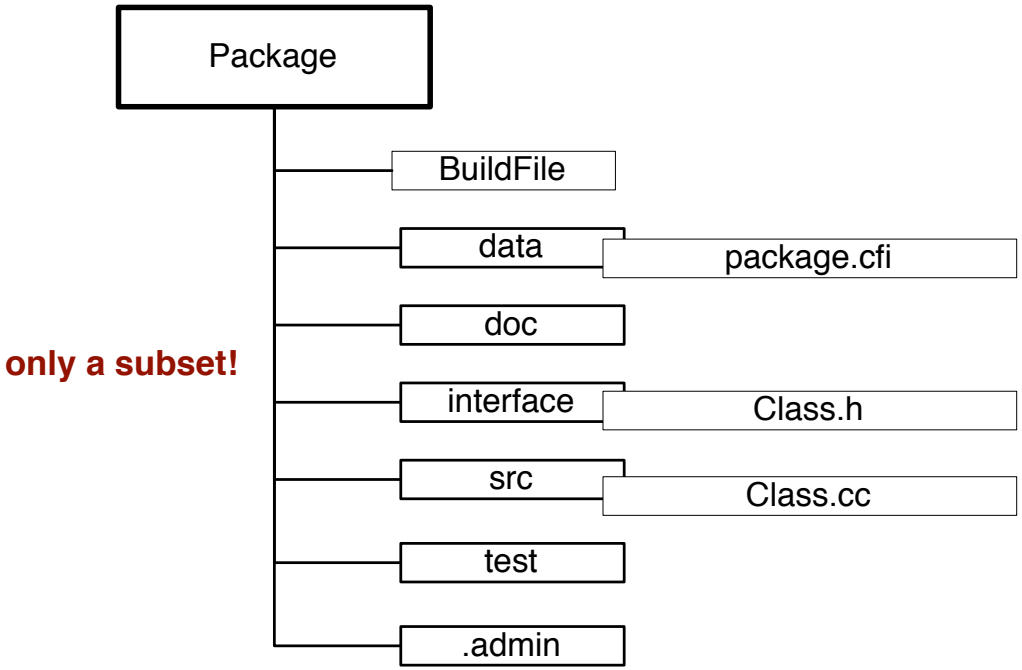
Preparing the environment

```
creating your local area
 $ cmsrel CMSSW_2_2_3
 [...]
 $ cd CMSSW_2_2_3/src

setting runtime variables
 $ cmsenv

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

The Release Area II

SCRAM = Software release and management tool

```
Project/Subproject/Package

CMSSW_2_2_3/src/GeneratorInterface/ToprexInterface
```

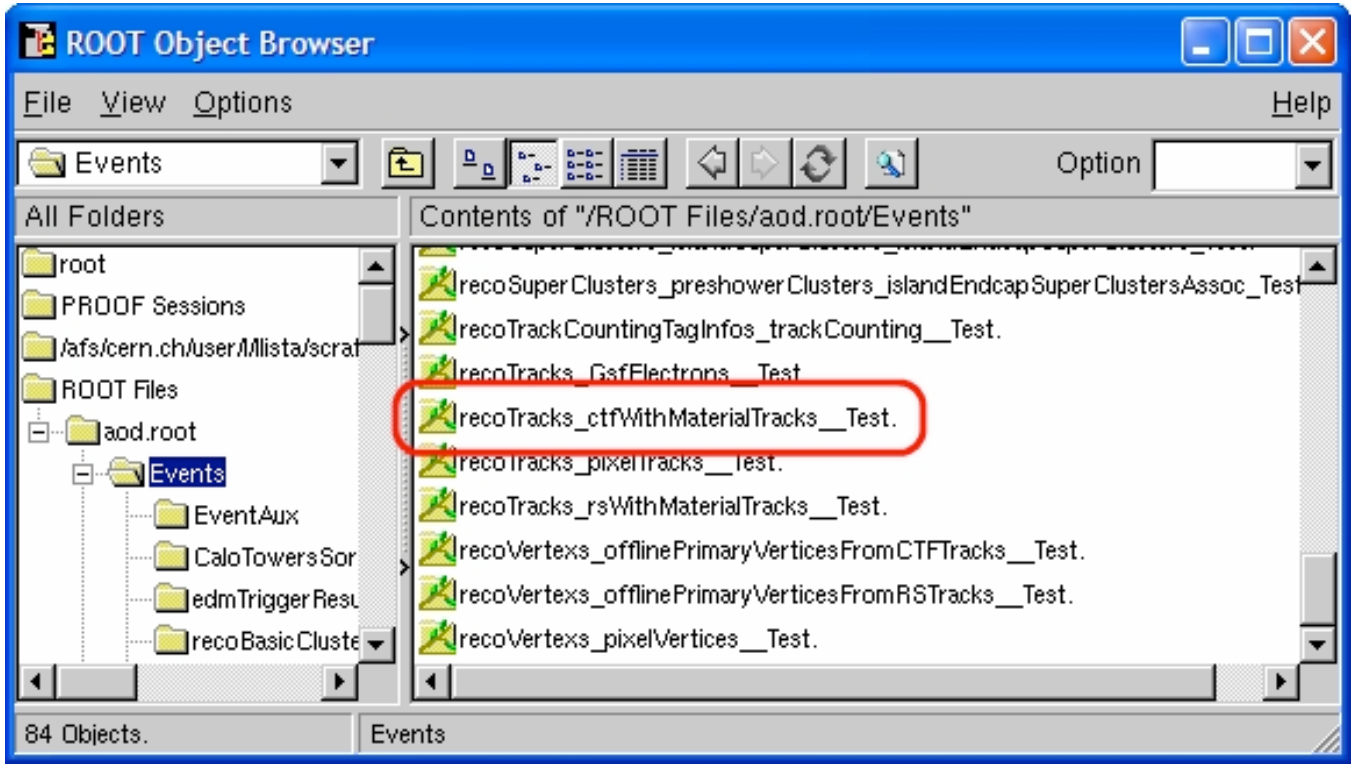


only a subset!

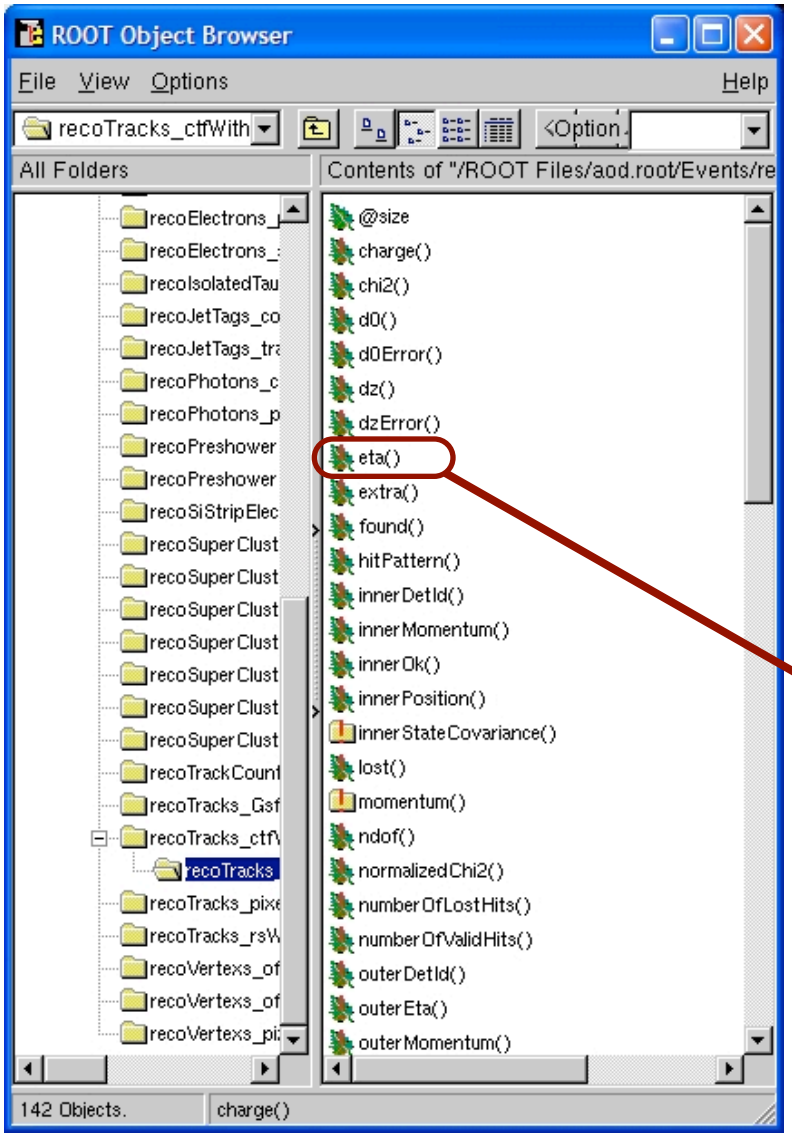
The Data format

Files can be opened with ROOT

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

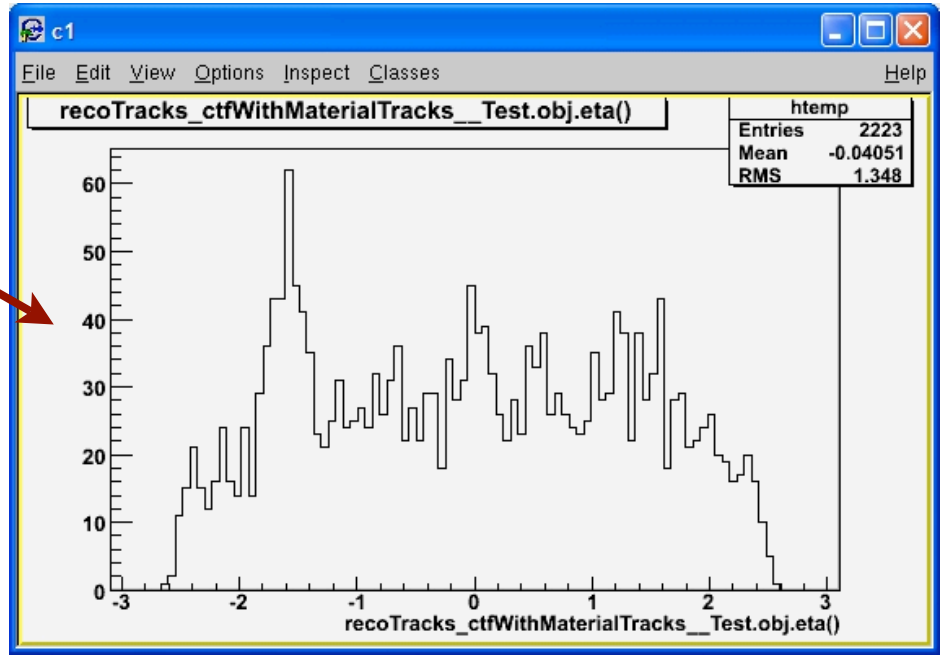


FWLite gives access to the classes



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 );

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

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>

CMSSW Config Files

Configuration 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

The Process Object

A usual process definition

- Process

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

- An input source

```
process.source = cms.Source('PoolSource', ...)
```

- Some modules

```
process.jets = cms.EDProducer('JetProducer')
```

- Some services

```
process.tracer = cms.Service('Tracer')
```

- Some execution paths

```
process.p1 = cms.Path(process.a * process.b)
```

- Maybe output modules

```
process.out = cms.OutputModule(...
```

CMS Python Config Types – 1/2

- Most of the objects you'll create will be of a CMS-specific type. To make them known to the interpreter you do:

```
import FWCore.ParameterSet.Config as cms
```

- Objects are then created with a syntax like:

```
jets = cms.EDProducer('JetReco',
        coneSize = cms.double(0.4),
        debug = cms.untracked.bool(True)
       )
```

Warning:

The comma between the parameters is very important!

Forgetting the comma after line n results in a syntax error reported for line n+1. So not straightforward to track down!

CMS Python Config Types – 2/2

CMS Python Types:

DATA TYPES	COMPOSITES	EXECUTION	MODULES	MODIFIERS
int32	vint32	Process	EDProducer	untracked
uint32	vuint32	Sequence	EDAnalyzer	ESPrefer
int64	vint64	Path	EDFilter	
uint64	vuint64	EndPath	ESProducer	
double	vdouble	Schedule	ESSource	
bool	vstring	SequencePlaceholder	Source	
string	VPSet		SecSource	
PSet	VInputTag		Looper	
InputTag	VEventID		Service	
EventID			OutputModule	
FileInPath				

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

Sequences, Paths and Schedules

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)
```

Schedule:

• Defines the execution order of paths. If not given first all paths, then all endpaths.

```
process.schedule = cms.Schedule(process.p1, process.outpath)
```

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
  )
```

Tracked / Untracked

Tracked parameters:

- Parameters that change the content of the event
- Will be saved in the event data provenance
- Cannot be optional
  - if asked for and not found, exception will be thrown
  - constructs to circumvent this policy are highly discouraged

```
a = cms.double(3.0)
```

Untracked parameters:

- Parameters that don't affect the results, e.g. debug level
- Can have default values

```
a = cms.untracked.double(3.0)
```

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

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)

They cannot be run in the same jobs

- HLT and RECO have various conflicts
- Memory problems
- Official production has two step procedure GEN->HLT + RAW2DIGI,RECO

Writing your own framework module

What module type to write

EDAnalyzer

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

EDProducer

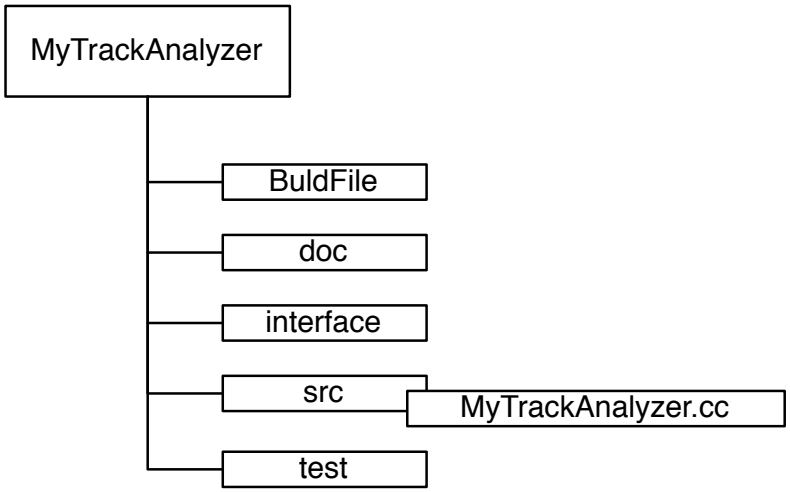
- You want to create new products
- You want to share your reconstruction code with others
- You want to make different algorithms plugable

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_1/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
$ scram 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
$ scram 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
```

Tips’n’tricks

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),
...

Config Tools – 2/4

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
```

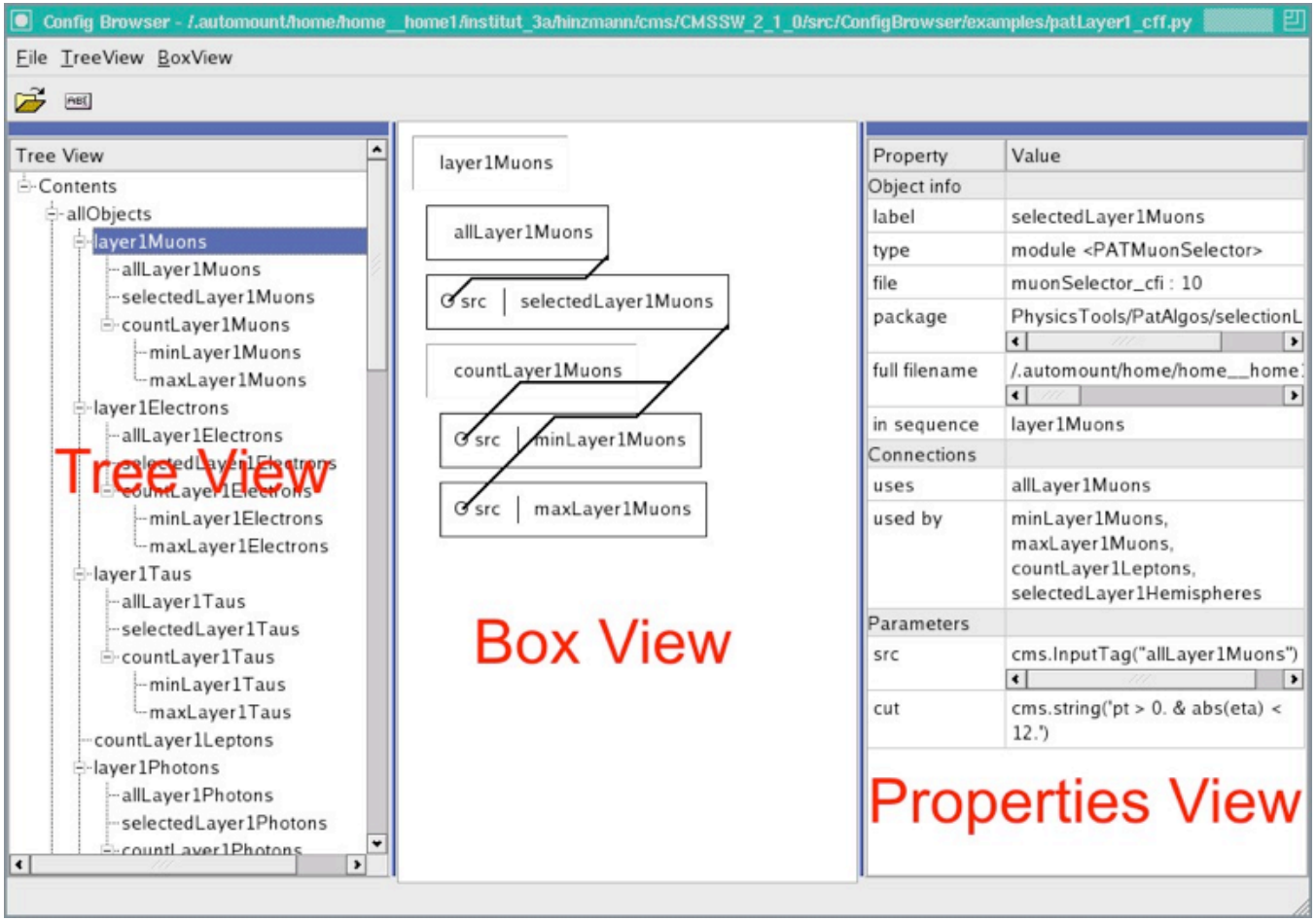
- Simple commands will bring you forward

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

- Ctrl-D shuts it down

Config Tools – 4/4

PythonConfigBrowser



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

Debugging with CMSSW

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

 ...
```

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

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

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!

Finding code

Another 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)
```

- CMS Workbook:
  https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook

- Reference manual:
  http://cmsdoc.cern.ch/cms/cpt/Software/html/General/gendoxy-doc.php

- WEBcvs:
  http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/CMSSW/?cvsroot=CMSSW

- LXR:
  http://cmslxr.fnal.gov/lxr/

- HyperNews:
  https://hypernews.cern.ch/HyperNews/CMS/

Thank you very much
for you attention!