

CMSSW Tutorial – Part 2

Goals of Part 2

- ❑ *Finish the analysis from Part 1.*
- ❑ *Learn how to setup default parameters.*
- ❑ *Learn how to look at the contents of an Event.*
 - *Dumping the Event*
 - *Looking at the ROOT file*
- ❑ *Understand the Configuration File Language.*
 - *So that you can better understand what has been done.*

Last class...

□ *Now we have the number of tracks in each event!*

%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004

Event: 1 number of tracks 4

%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004

Event: 2 number of tracks 4

%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004

Event: 3 number of tracks 4

%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004

Event: 4 number of tracks 3

%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004

Event: 5 number of tracks 1

□ *This is a $H \rightarrow ZZ \rightarrow 4\mu$ event.*

□ *How many charged tracks should it have then...?*

Filtering some events

- ❑ *We want only the events with 4 tracks.*
 - *Remember: tracks are created only by charged particles – the muons, in this case.*
- ❑ *Our DemoAnalyzer should do the job.*
- ❑ *Edit demo.cfg file to add a "minTracks" parameter*
 - *This parameter will then be used inside of the DemoAnalyzer code.*

```
module demo = DemoAnalyzer {  
    untracked uint32 minTracks = 4  
}
```

Filtering some events

- *Edit the DemoAnalyzer.cc source code, and modify it in 3 places:*

```
//-----member data-----  
unsigned int minTracks_;
```

```
DemoAnalyzer::DemoAnalyzer(const edm::ParameterSet& iConfig) :  
minTracks_(iConfig.getUntrackedParameter<unsigned int>("minTracks",0))
```

```
if( minTracks_ <= tracks->size() ) {  
  LogInfo("Demo") << "number of tracks " << tracks->size();  
}
```

- *The first modification declares the minTracks_ variable in our code.*
- *The second one says that minTracks_ will receive its value from the parameter minTracks – notice the missing underscore.*
- *The third one makes that the number of tracks is printed only if there are 4 or more charged tracks in the Event.*

Filtering some events

Rebuild the binaries

– *This step must be done every time you modify a .cc file.*

```
$ scramv1 b
```

Run the job

```
$ cmsRun demo.cfg
```

Now, only events with 4 tracks appear

```
%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004
```

```
Event: 1 number of tracks 4
```

```
%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004
```

```
Event: 2 number of tracks 4
```

```
%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004
```

```
Event: 3 number of tracks 4
```

```
%MSG-i Demo: DemoAnalyzer:demo 18-Apr-2007 14:12:20 CEST Run: 5004
```

```
Event: 6 number of tracks 4
```

What else can be done?

❑ *Create a .cfi file – with default values for the required parameters*

– *Go to the package directory*

```
$ cd Demo/DemoAnalyzer
```

– *Make a new 'data' directory and go inside*

```
$ mkdir data
```

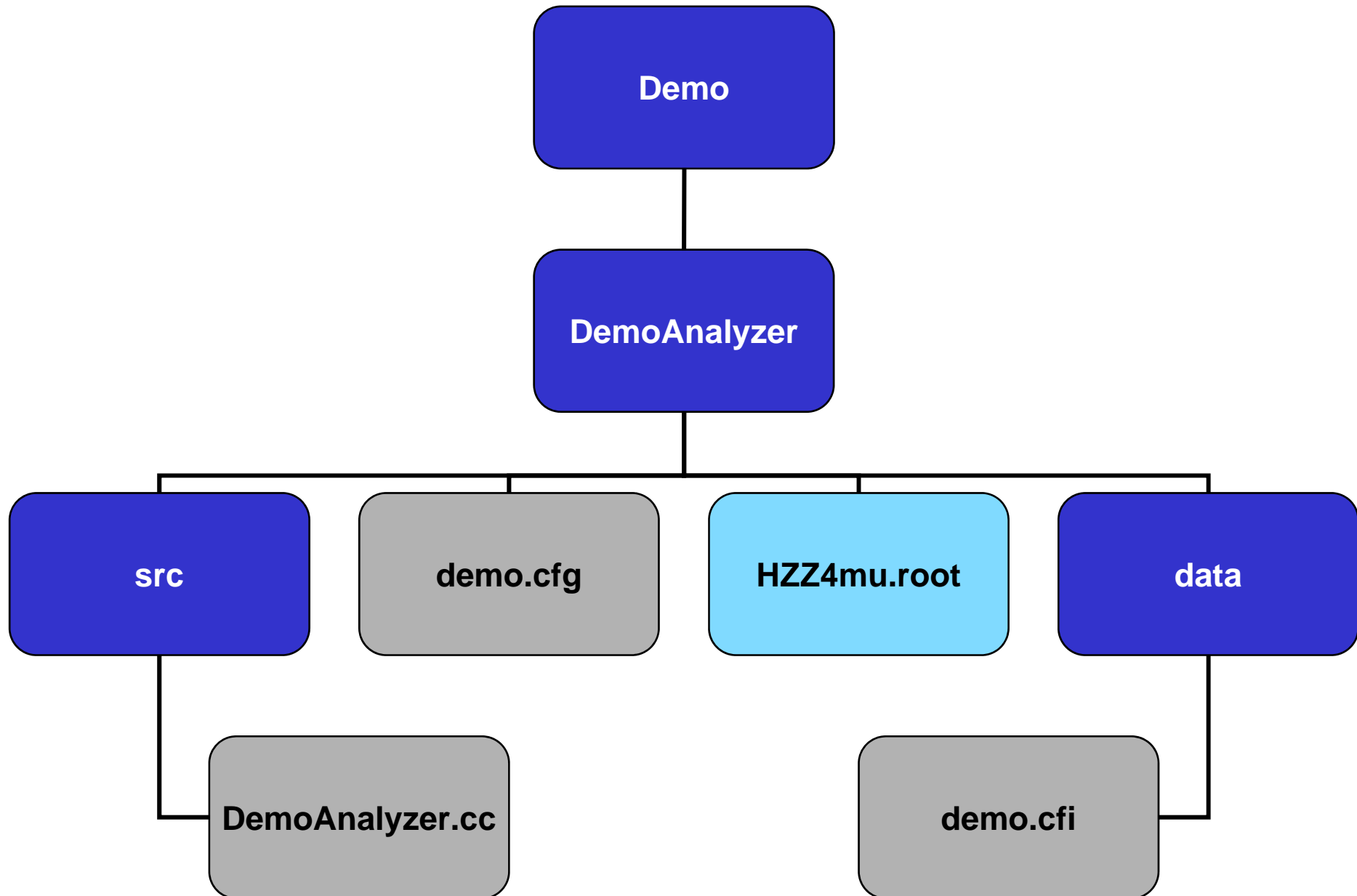
```
$ cd data
```

– *Create the file demo.cfi, with the following content:*

```
module demo = DemoAnalyzer {  
    untracked uint 32 minTracks = 0  
}
```

– *Notice that we're declaring the demo module inside a .cfi file.*

The directory layout



– *So, it must be not declared elsewhere!*

– *Edit the demo.cfg file and replace*

```
module demo = DemoAnalyzer {  
    untracked uint 32 minTracks = 0  
}
```

with

```
include "Demo/DemoAnalyzer/data/demo.cfi"  
replace demo.minTracks = 4
```

(Notice the include statement. That means we're pasting the contents of that file at that point.)

– *Rerun the job*

```
$ cd ..  
$ cmsRun demo.cfg
```

– *The output should not change.*

Event content

- ❑ *Not only charged tracks are inside the Event!*
- ❑ *To see what else is available*

1. One could "dump" the event.

- Edit the demo.cfg file*
- Add the Event Content Analyzer module*
Remember, a module needs a name and a label!
- Add it to the path*

```
module dump = EventContentAnalyzer {}
```

```
...
```

```
path p = {demo & dump}
```

- Add the number of events in PoolSource (in the .cfg file), after the input file name, so to dump one event only.

untracked int 32 maxEvents = 1

- Rerun the job, redirecting output to a file.

cmsRun demo.cfg > output.txt

- Remember the syntax (in red):

TYPE MODULE_NAME INSTANCE_NAME PROCESS

...

```
++EcalRecHitsSorted "ecalRecHit" "EcalRecHitsEB"  
++EcalRecHitsSorted "ecalRecHit" "EcalRecHitsEE"  
++EcalUncalibratedRecHitsSorted "ecalWeightUncalibRecHit"  
"EcalUncalibRecHitsEB"  
++EcalUncalibratedRecHitsSorted "ecalWeightUncalibRecHit"  
"EcalUncalibRecHitsEE"  
++HBHEDataFramesSorted "hcalDigis" ""  
++HBHERecHitsSorted "hbhereco" ""
```

...

Event contents - continued

2. One can also open the ROOT file itself.

– But, ROOT must be instructed on how to open it – specific libraries must be loaded.

– To enable ROOT to load libraries:

```
$ root -l
```

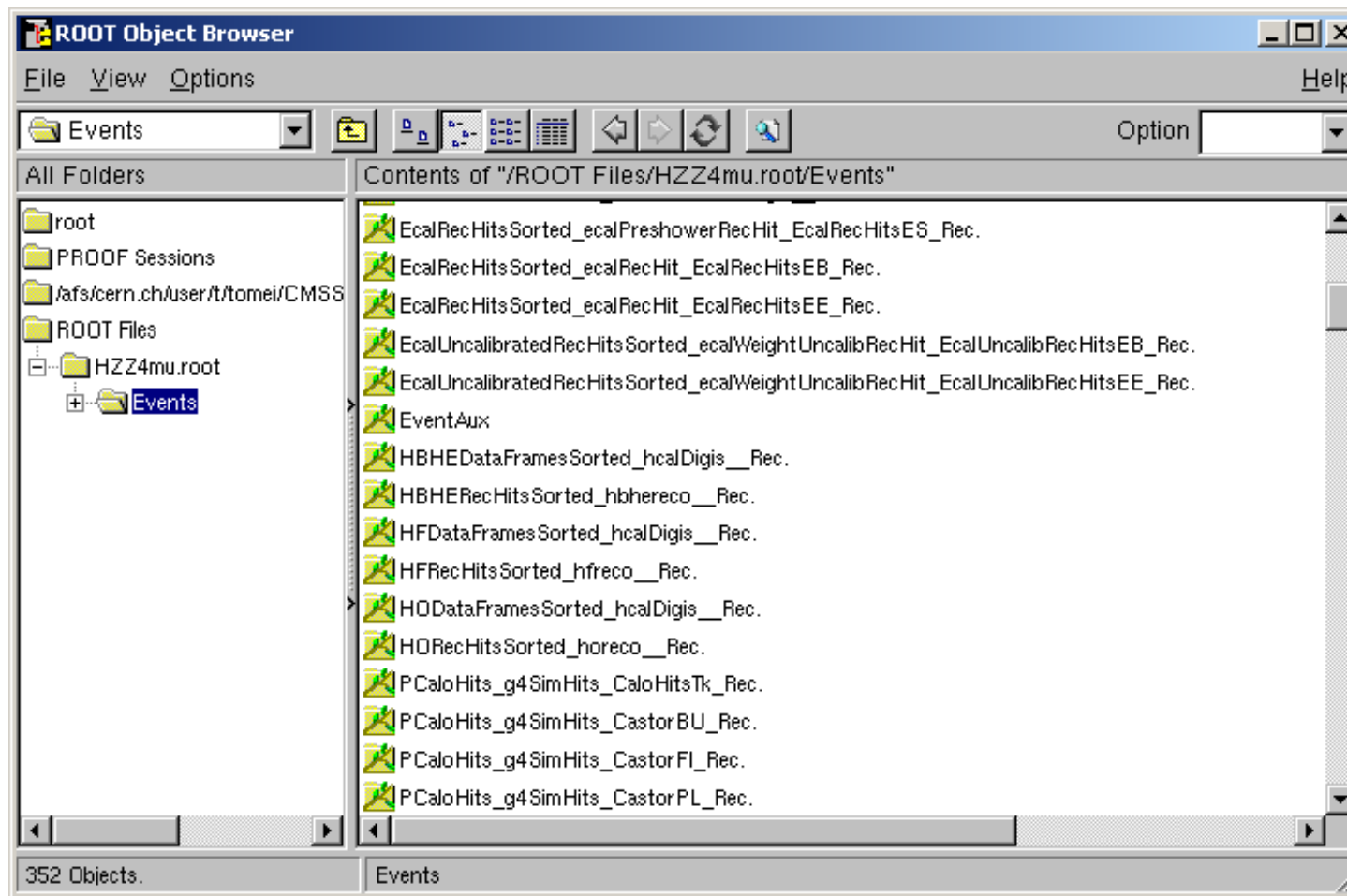
```
root[0] gSystem->Load("libFWCoreFWLite.so");
```

```
root[1] AutoLibraryLoader::enable();
```

```
root[2] TFile *_file0 = TFile::Open("HZZ4mu.root")
```

– See the different objects inside the ROOT file.

Event contents - continued



□ Notice that the contents are the same as per the other method.

The Configuration File Language

- ❑ *The .cfg is a hierarchy of **blocks** , nested or adjacent with respect to each other.*
- ❑ *Each block specifies a component of cmsRun program.*
- ❑ *The block keyword identifies (and possibly names) the component type.*
- ❑ *The contents of the block are contained within the block's scope (delimited by { })*

The process block

- ❑ *It is the top-level block – all other blocks are contained within it.*
- ❑ *Each .cfg file must have exactly process block.*
- ❑ *It must be named – its name gets carried along with the output data and is used to distinguish between similar objects in a given event.*

```
process NAME  
{  
...  
}
```

The source block

- ❑ *The data input source for cmsRun is declared using exactly one unnamed **source** block.*
- ❑ *For simulated data, one could choose PythiaSource, FlatRandomEGunSource, PoolSource (to read from a file of generated data).*
 - *We have been using PoolSource until now.*
- ❑ *Notice that Pythia is a program on its own, but can be used as a plugin module for CMSSW.*

```
source = PythiaSource
{
...
}
```

Modules

- ❑ *Modules are the "workers" – components that actually do something with the Event data.*
- ❑ *Three types of modules:*
 - *EDProducer: creates new data to be placed in the Event*
 - *EDFilter: decides if processing should continue for an Event*
 - *EDAnalyzer: studies properties of the Event*

```
module filter = PythiaFilter {  
    untracked string moduleLabel = `PythiaSource`  
    untracked double MinMuonPt = 20.  
}
```

Services

- ❑ *Services are a facilities that perform well-defined tasks which are globally acessible and do not affect physics results.*
- ❑ *Services can go unnamed.*

```
service = RandomNumberGeneratorService  
{  
    untracked uint32 sourceSeed = 54321  
}
```

Parameter Set (PSet)

- ❑ *Parameter Set are used to define a list of parameters.*
- ❑ *Can be declared globally (inside the process block itself) to define a list of parameters that are used to configure different modules.*
- ❑ *Having a Parameter Set allow you to have a central maintenance point for all parameters defined.*

```
PSet my_parameters =  
{  
...  
}
```

Module sequences

- ❑ *Module sequences are used to define an ordered group of modules.*
- ❑ *Useful to group modules which you want to use together, or to switch them in and out from the execution path.*

sequence trDigi = { pixdigi, stripdigi }

sequence calDigi = { ecaldigi, hcal digi }

sequence muDigi = { muoncscdigi, muondtdigi }

sequence allDigi = { trDigi, calDigi, muonDigi }

Paths and the endpath

- ❑ *Paths define groups of ordered modules which are to be executed for each Event.*
- ❑ *Endpaths are executed after all other named paths have been run.*
- ❑ *Same syntax as for the sequences:*

path p = { allDigi, reco }

endpath end = { output }

Include and replace

- ❑ *The **include** statement injects the text from another file in the place where it was used.*
- ❑ *Mainly used with .cfi files.*

```
include "IOMC/.../PythiaHZZ4mu.cfi"  
replace PythiaSource.maxEvents = 20
```

- ❑ *You can use the **replace** statement to replace standard parameters for modules defined in those included files*