

CMSSW Tutorial – Part 1

Goals of Part 1

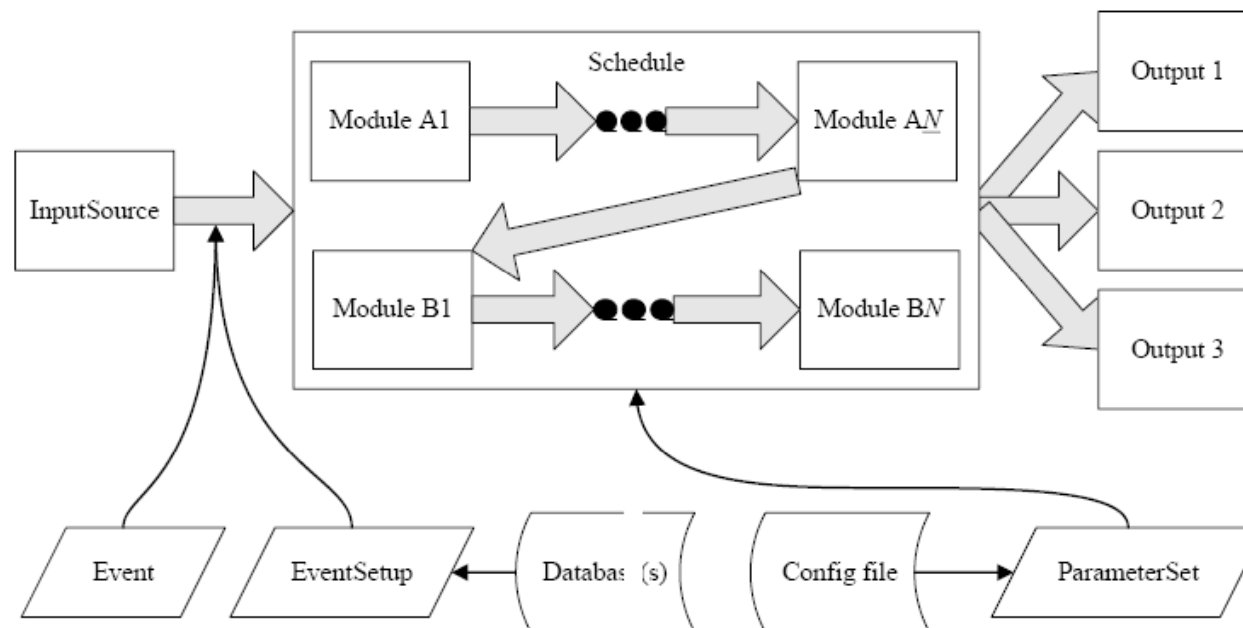
- ❑ *Understand the Framework concept.*
- ❑ *Understand the (modular) architecture.*
 - *Six types of modules.*
- ❑ *Understand the Event concept.*

- ❑ *Learn how to set up your computer environment.*
- ❑ *Baby steps in setting up the config file.*
- ❑ *Baby steps in setting up an analysis.*

Understand the Framework concept

□ *As we have seen:*

- *Software bus model*
- *One main executable: cmsRun*
- *Plugin modules – user code is one of them!*
- *Config file .cfg – configures cmsRun execution at runtime.*



Modular architecture

- ❑ *A module is a piece of CMSSW code that can be plugged into the CMSSW executable cmsRun.*
- ❑ *When preparing an analysis job:*
 - *The user selects which module(s) to run.*
 - *Specifies a ParameterSet for each via the .cfg file.*
 - *The modules are called for every event, in the order given by the path statement in the .cfg*
- ❑ *Six types of modules:*
 - *Source (not quite a module, but...)*
 - *EDProducer*
 - *EDFilter*
 - *EDAnalyzer*
 - *EDLooper*
 - *OutputModule*

Module Syntax

□ *In the .cfg file:*

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

- *module label (also called tag)*
- *module name*
- *module parameters*

□ *When you label a module, you make no references to its name in the code anymore.*

A quick example

```
process Demo = {  
  #keep the logging output to a nice level  
  include "FWCore/MessageLogger/data/MessageLogger.cfi"  
  source = PoolSource  
  {  
    untracked vstring fileNames = {"file:HZZ4mu.root"}  
  }  
  module demo = DemoAnalyzer  
  {  
    untracked uint32 minTracks = 4  
  }  
  path p = {demo}  
}
```

Remember:
- *module label* -
module name -
parameters

Event Data Model

- ❑ *An Event starts as a collection of the RAW data from detector or MC event.*
- ❑ *In software terms: an Event is a single entity in memory, a C++ type-safe container called*

`edm::Event`

- ❑ *Data within the Event are uniquely identified by four quantities:*
 - *C++ class type of the data*
 - *module label*
 - *product instance label (usually empty string)*
 - *process name*

`reco::TrackCollection_TrackProducer__PROD`

class type **module label** **process name**

Getting data from the Event

- ❑ *To hold an access result, all Event data access methods use*

`edm::handle<type>`

where type is the C++ type of the datum.

- ❑ *To request data from an Event, use a form of the following:*

- *get which either returns one object or throws a C++ exception.*

- *getMany which returns a list of zero or more matches to the data request.*

- ❑ *After get or getMany , indicate how to identify the data, e.g getByLabel , and then use the name associated with the handle type.*

Another quick example

❑ *In the analyzer.cc file:*

```
void DemoAnalyzer::analyze(edm::Event const& e, edm::EventSetup
const& iSetup) {
    edm::Handle<reco::TrackCollection> trk_hits;
    e.getByLabel("TrackProducer", trk_hits);
    // Do some (hopefully) useful analysis here
}
```

❑ *Analysis modules are written as C++ code.*

❑ *The analyzer.cc file must have three methods:*

- beginJob(const edm::EventSetup&);
- analyze(const edm::Event&, const edm::EventSetup&);
- endJob();

Writing our own module

□ *Set CMSSW up*

`$ scramv1 project CMSSW CMSSW_1_3_3` → *setup CMSSW 1.3.3*

`$ cd CMSSW_1_3_3/src`

`$ eval `scramv1 runtime -sh`` → *setup shared libs*

`$ project CMSSW` → *setup CVS*

□ *Prepare a Demo area*

`$ mkdir Demo`

`$ cd Demo`

`$ mkedanlizr DemoAnalyzer` → *creates the skeleton of our module*

`$ cd DemoAnalyzer`

`$ scramv1 b` → *builds the executables*

Start building the .cfg and .cc files

- *Let's take a first look at the skeleton analyzer*

\$ emacs -nw src/DemoAnalyzer.cc

– You are not required to work in Emacs – vim, and pico are also available at SPRACE.

- *Create the demo.cfg file in the the DemoAnalyzer directory*

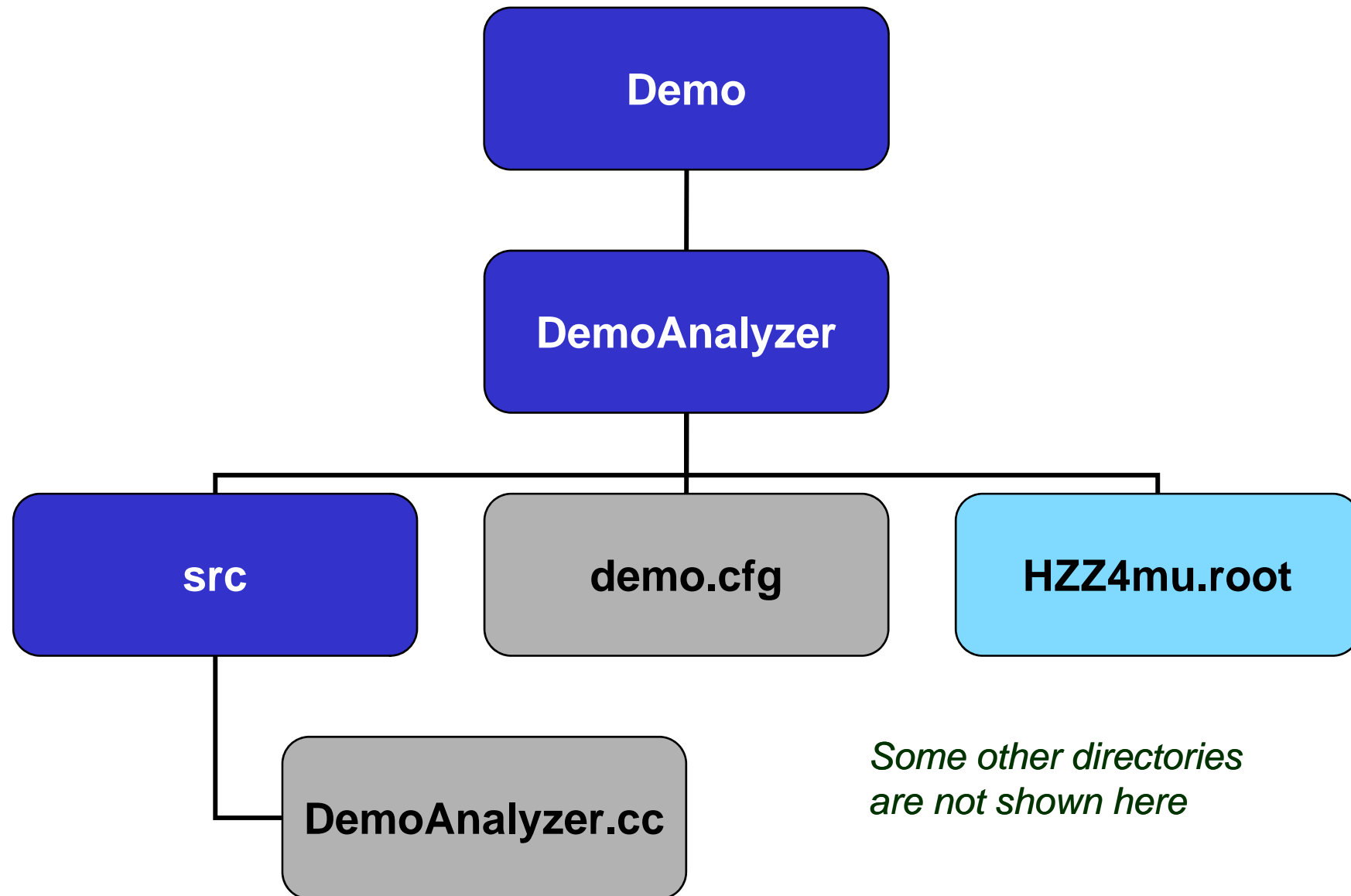
\$ touch demo.cfg

– We'll be editing it right away.

- *Link the data file in the DemoAnalyzer directory*

\$ ln -s /home/guest/sampledData/HZZ4mu.root HZZ4mu.root

The directory layout



*Some other directories
are not shown here*

□ *Include these contents in the demo.cfg file:*

```
process Demo = {  
  #keep the logging output to a nice level  
  include "FWCore/MessageLogger/data/MessageLogger.cfi"  
  source = PoolSource  
  {  
    untracked vstring fileNames = {"file:HZZ4mu.root"}  
  }  
  module demo = DemoAnalyzer  
  {  
  
  }  
  path p = {demo}  
}
```

Running the job

❑ *Run the job with*

\$ cmsRun demo.cfg

❑ *As of now, our DemoAnalysis does no analysis at all!*

– That was expected – the skeleton analyzer does nothing.

❑ *To change that, edit the DemoAnalyzer.cc file:*

\$ emacs –nw DemoAnalyzer.cc

– As a reminder, text already there will be in black, and text which you have to add will be in red .

Doing something useful

□ *Include two other header files:*

– *The track.h gives you tools for dealing with tracks.*

```
#include "FWCore/Framework/interface/Frameworkfwd.h"  
#include "FWCore/Framework/interface/EDAnalyzer.h"  
#include "DataFormats/TrackReco/interface/Track.h"  
#include "FWCore/MessageLogger/interface/MessageLogger.h"
```

□ *Edit the method analyze() :*

```
DemoAnalyzer::analyze(const edm::Event& iEvent, const edm::EventSetup&  
iSetup)
```

```
Handle<reco::TrackCollection> tracks;  
iEvent.getByLabel("ctfWithMaterialTracks", tracks);  
LogInfo("Demo") << "number of tracks " << tracks->size();
```

□ *Compile and run the code*

```
$ scramv1 b
```

```
$ cmsRun demo.cfg
```

Useful information!

□ *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...?*