Tutorial 1:
Basic LArSoft intro

Andrzej Szelc

Intro

- In this tutorial you will set up your working directory, and compile some LArSoft code.
- This should prepare you to run some light simulations tomorrow morning (or this afternoon if things go swimmingly).
- We will just do the very basic commands so that everyone is able to set up.

Logging in to Your Computing Account

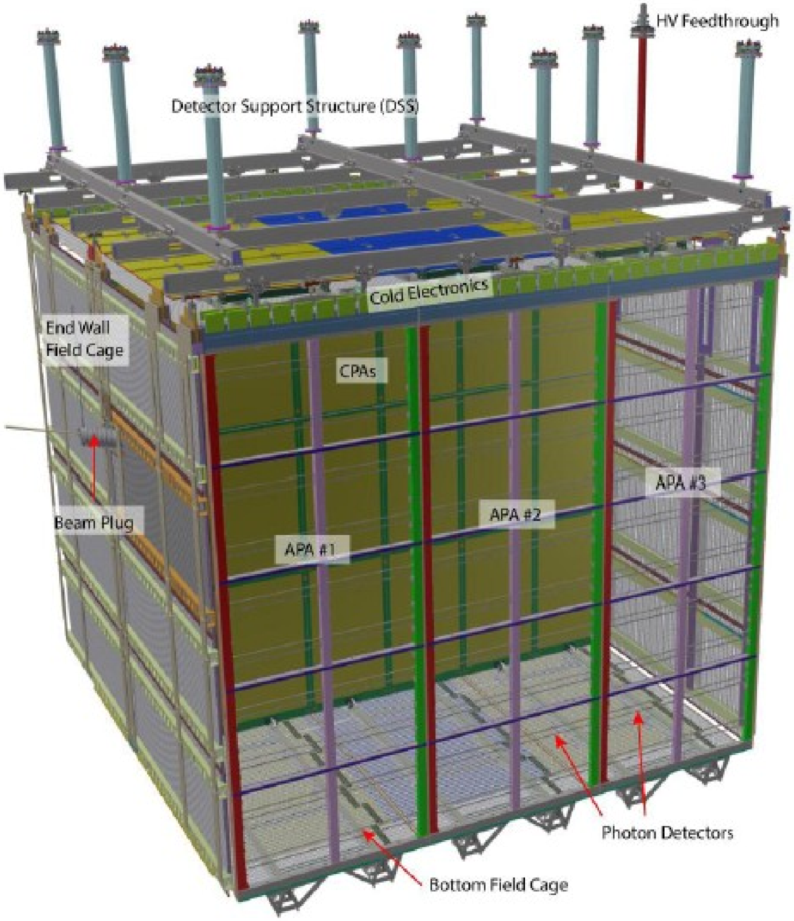
- ssh -X -Y user01@18.231.121.254
- Password is written on the whiteboard
- Most files you will need will be in /home/andrzej/workshop_files/

What we will be doing

- We will do the bare minimum to:
  - Setup a working LArSoft environment.
  - Run and view a simple “TPC” event.
  - Setup a working LArSoft directory
  - Download some code.
  - Start a compilation.
- If you have used LArSoft before, you may be bored. Sorry about that.

The protoDUNE detector

- We will use this for this part of the workshop, because it is smaller.
- Currently running at CERN.
- Has APAs (anode plane assemblies), and 6 TPCs. Although LArSoft thinks it's 12.



1. The simplest LArSoft job(s)

- `source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh`
  - Setup the repository
- `setup dunetpc v07_11_00 -q e17:prof`
  - Activate the dune specific code.
- `lar -c protoDUNE_gensingle_wkshop.fcl -n 5`
  - Generate particles (through event generation, LArG4, detsim – will explain later)
- `lar -c evd_protoDUNE_noped.fcl gen_protoDune_pion_2GeV_mono.root`
  - Launch event display

This file is in my workshop_files Directort only.

Familiarize yourself with The event display.

Take a look at the events.

2. Let's make a working Dir.

- `source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh`
  - Setup the repository
- `setup dunetpc v07_11_00 -q e17:prof`
  - Activate the dune specific code.

If you have not logged out
You do **not** need to do these

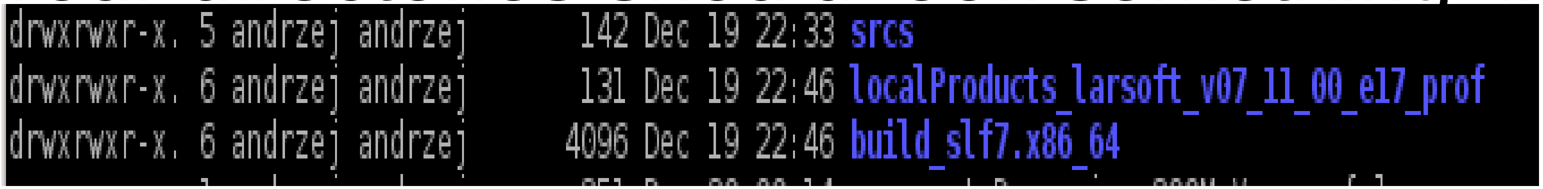
- `mkdir <MyWorkingDir>; cd <MyWorkingDir>`
  - Make a working directory
- `mrb newDev`
  - Create the framework for a LarSoft installation
- `source localProducts_larsoft_v07_11_00_e17_prof/setup`
  - Add local repository to a source of products. Check $PRODUCTS now!

We haven't really done
Anything much, except
prepare ourselves.

Next two slides, give a
quick description of what
happened

LarSoft/MRB (1)
-directory structure

- Your directories should look something like this:



```
drwxrwxr-x. 5 andrzej andrzej      142 Dec 19 22:33 srcs
drwxrwxr-x. 6 andrzej andrzej      131 Dec 19 22:46 localProducts_larsoft_v07_11_00_e17_prof
drwxrwxr-x. 6 andrzej andrzej     4096 Dec 19 22:46 build_slf7.x86_64
```

- srcs: this is where the source code lives.
- build: this is where the code is compiled/built
- localProducts.../ : this is where you install the result of the compilation, i.e. a local version of your **product(s)**

For reference:
Log out/log in.

-
  ```
  source /cvmfs/dune.opensciencegrid.org/products/dune/setup_dune.sh
  ```
  - Setup the repository

-
  ```
  cd <MyWorkingDir>
  ```
  - Go to your working directory

-
  ```
  source localProducts_larsoft_v07_11_00_e17_prof/setup
  ```
  - Add local repository to a source of products.

-
  ```
  setup dunetpc v07_11_00 -q e17:prof
  ```
  #if you haven't compiled dunetpc yet. Otherwise:
  ```
  mrbslp
  ```

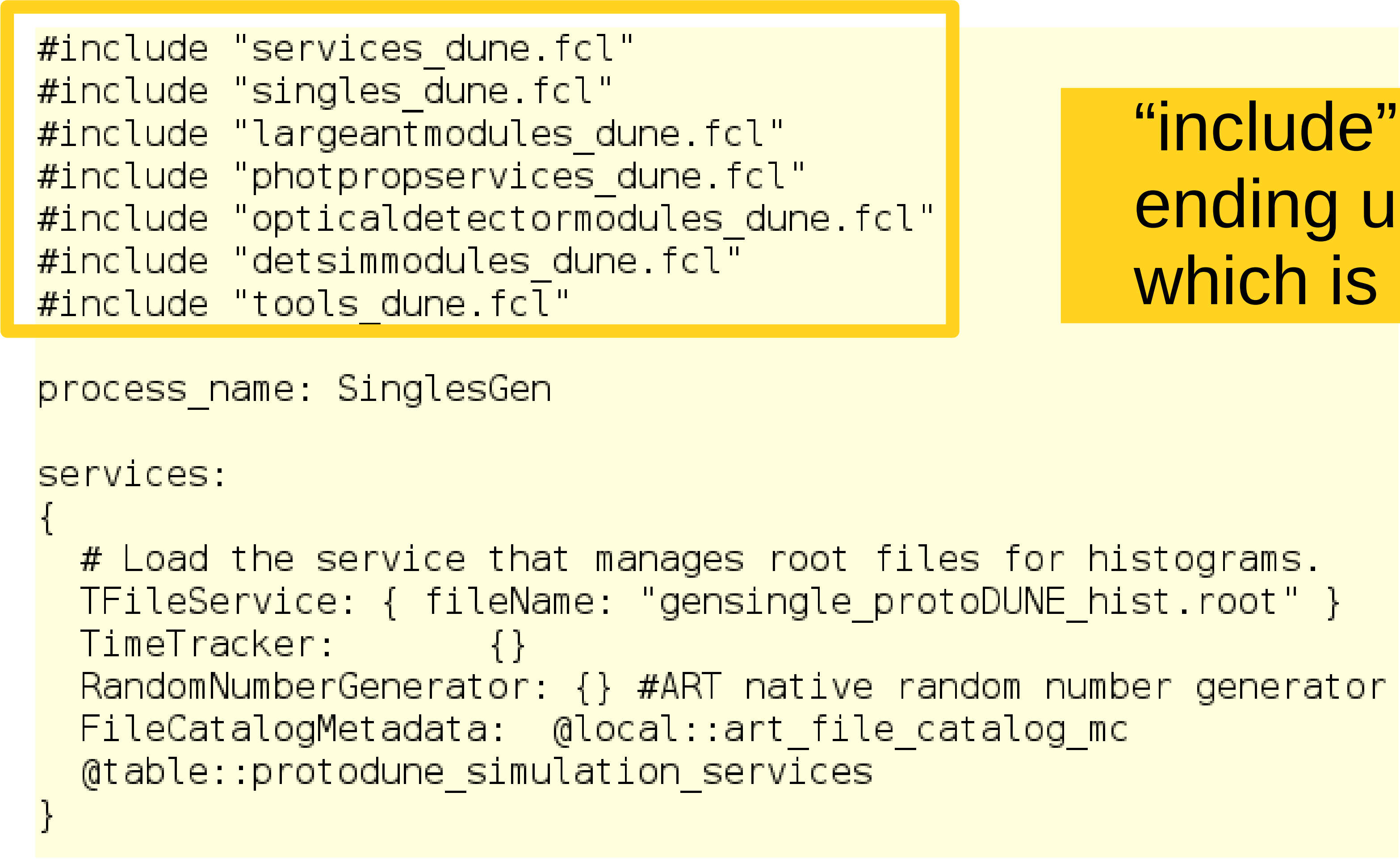
- This is to make sure your code is setup.

3. Locating “active” .fcl files

- Take a look at the .fcl file we used.
  Use “less” or “cat”.
- They have a lot of info, but also include source .fcl files.
- Use fhicl-expand to see the rest of the files.
- Let's try to locate the original files. We need find-fhicl.sh (you can either get it from my directory: /home/andrzej/workshop_files) or:
  - `source /cvmfs/uboone.opensciencegrid.org/products/setup_uboone.sh`
  - `setup ubutil v07_11_00 -q e17:prof`
  - `find_fhicl.sh evd_dune.fcl`
  - `unsetup ubutil` //optional, you won't have access to find_fhicl.sh anymore

The find_fhicl.sh, and other .fcl files are also in:
`/home/andrzej/workshop_files/`

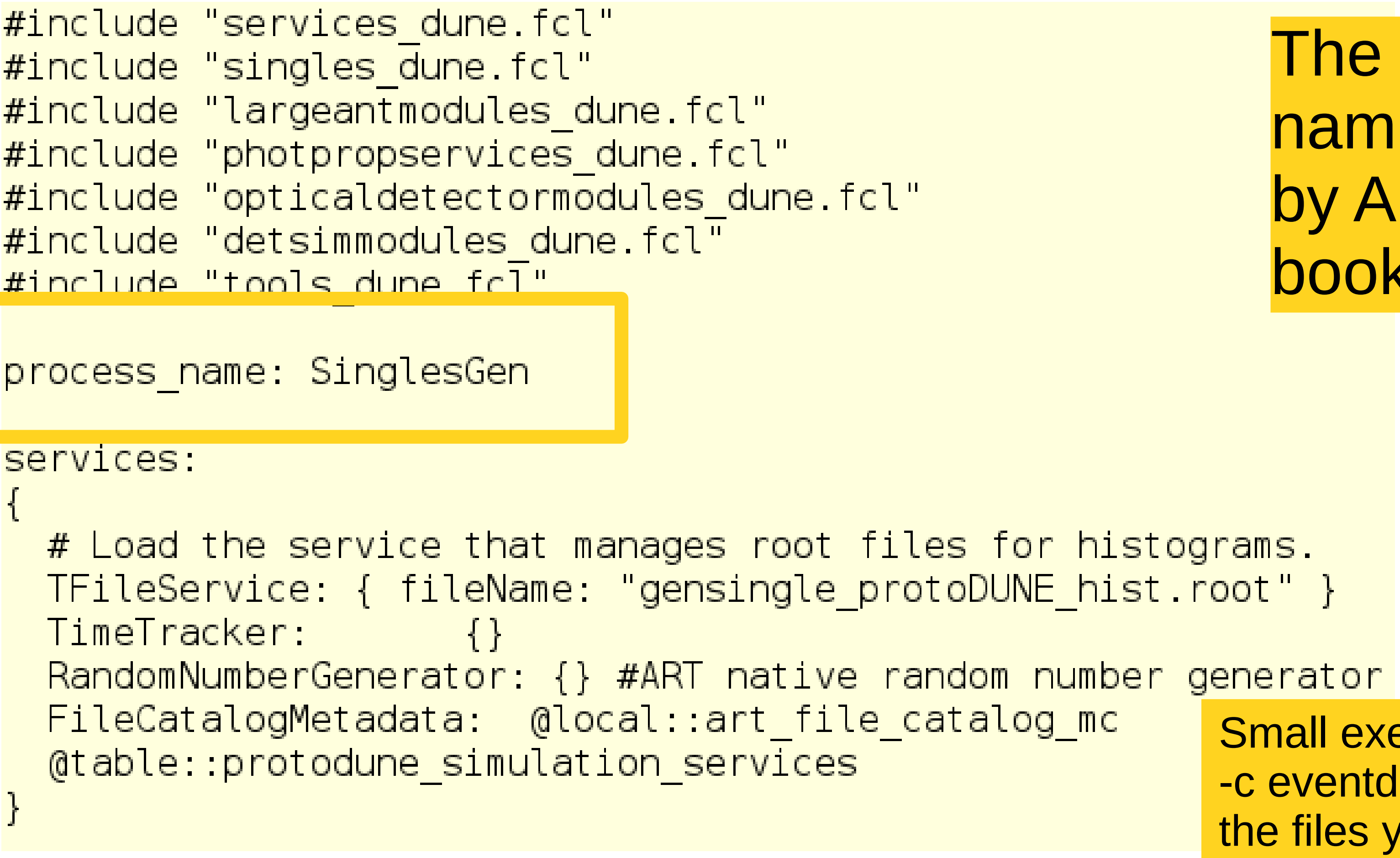
ProtoDUNE_gensingle.fcl



```
#include "services_dune.fcl"
#include "singles_dune.fcl"
#include "largeantmodules_dune.fcl"
#include "photpropservices_dune.fcl"
#include "opticaldetectormodules_dune.fcl"
#include "detsimmodules_dune.fcl"
#include "tools_dune.fcl"

process_name: SinglesGen

services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "gensingle_protoDUNE_hist.root" }
  TimeTracker:       {}
  RandomNumberGenerator: {} #ART native random number generator
  FileCatalogMetadata:  @local::art_file_catalog_mc
  @table::protodune_simulation_services
}
```

“include” files (also ending up with .fcl, which is confusing)

ProtoDUNE_gensingle.fcl



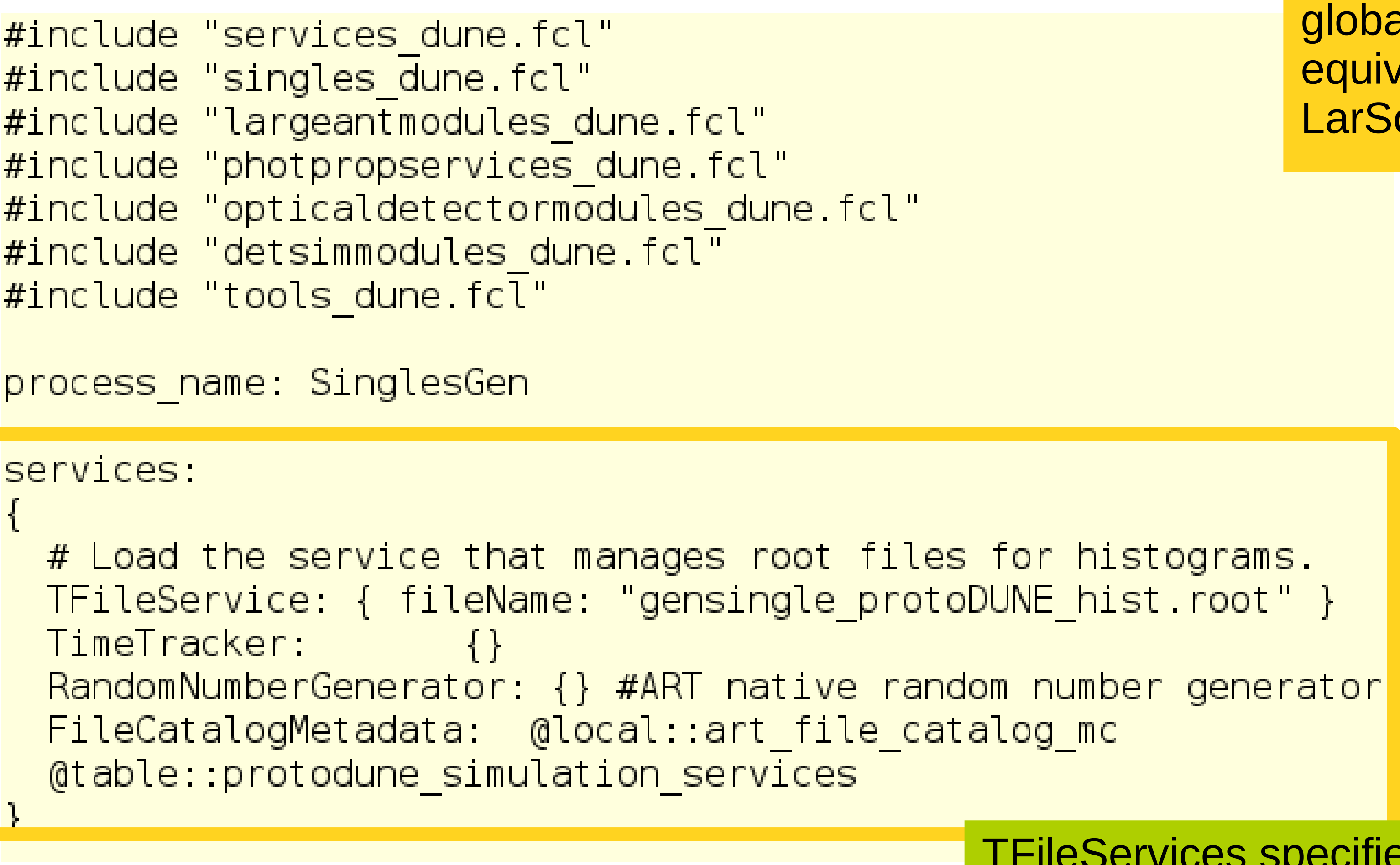
```
#include "services_dune.fcl"
#include "singles_dune.fcl"
#include "largeantmodules_dune.fcl"
#include "photpropservices_dune.fcl"
#include "opticaldetectormodules_dune.fcl"
#include "detsimmodules_dune.fcl"
#include "tools_dune.fcl"

process_name: SinglesGen

services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "gensingle_protoDUNE_hist.root" }
  TimeTracker:       {}
  RandomNumberGenerator: {} #ART native random number generator
  FileCatalogMetadata:  @local::art_file_catalog_mc
  @table::protodune_simulation_services
}
```

The process name, used by ART for bookkeeping.

Small exercise, run lar -c eventdump.fcl on the files you generated in the first tutorial. There should be 3 process names.

ProtoDUNE_gensingle.fcl

Services, these are global variable equivalents used by LarSoft.

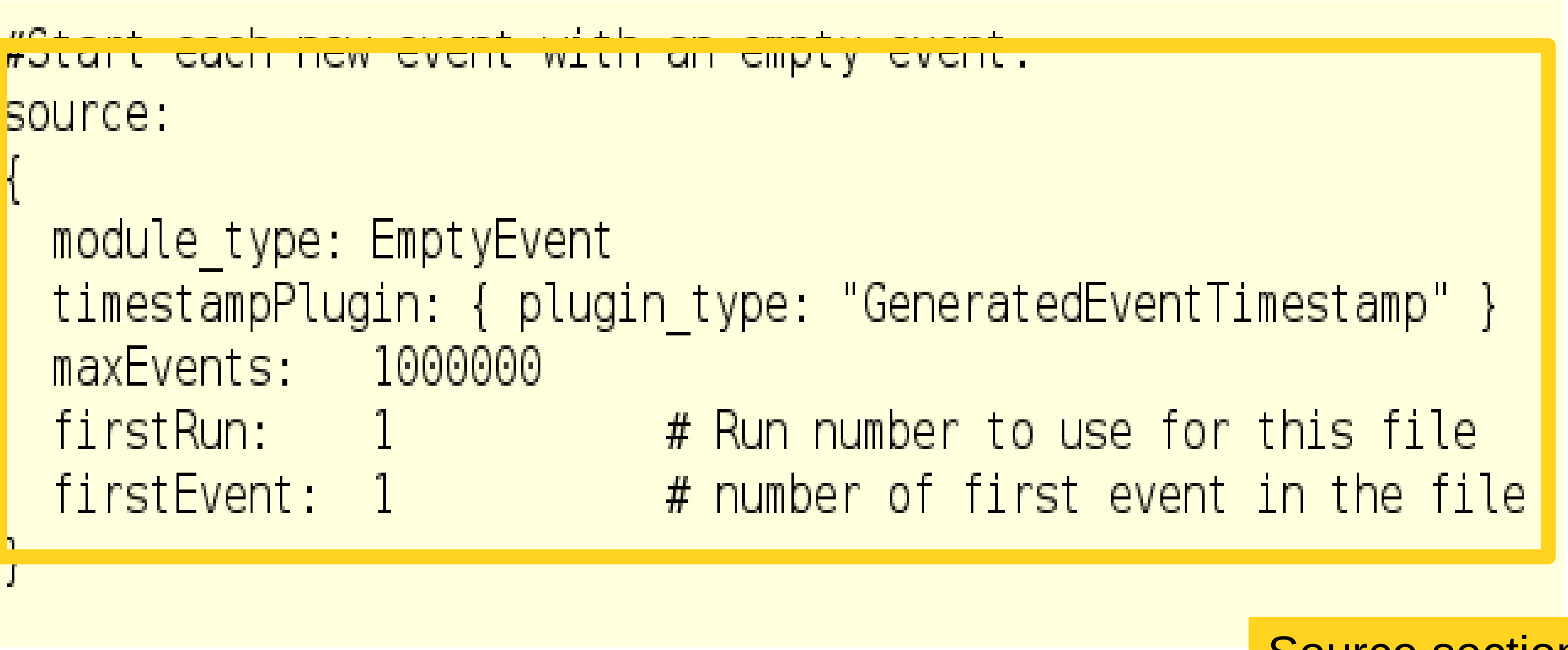


```
#include "services_dune.fcl"
#include "singles_dune.fcl"
#include "largeantmodules_dune.fcl"
#include "photpropservices_dune.fcl"
#include "opticaldetectormodules_dune.fcl"
#include "detsimmodules_dune.fcl"
#include "tools_dune.fcl"

process_name: SinglesGen

services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "gensingle_protoDUNE_hist.root" }
  TimeTracker:       {}
  RandomNumberGenerator: {} #ART native random number generator
  FileCatalogMetadata:  @local::art_file_catalog_mc
  @table::protodune_simulation_services
}
```

TFileServices specifies what is the name of the root output file name.

The other one is a list of services defined as a table. See included: services_dune.fcl

ProtoDUNE_gensingle.fcl



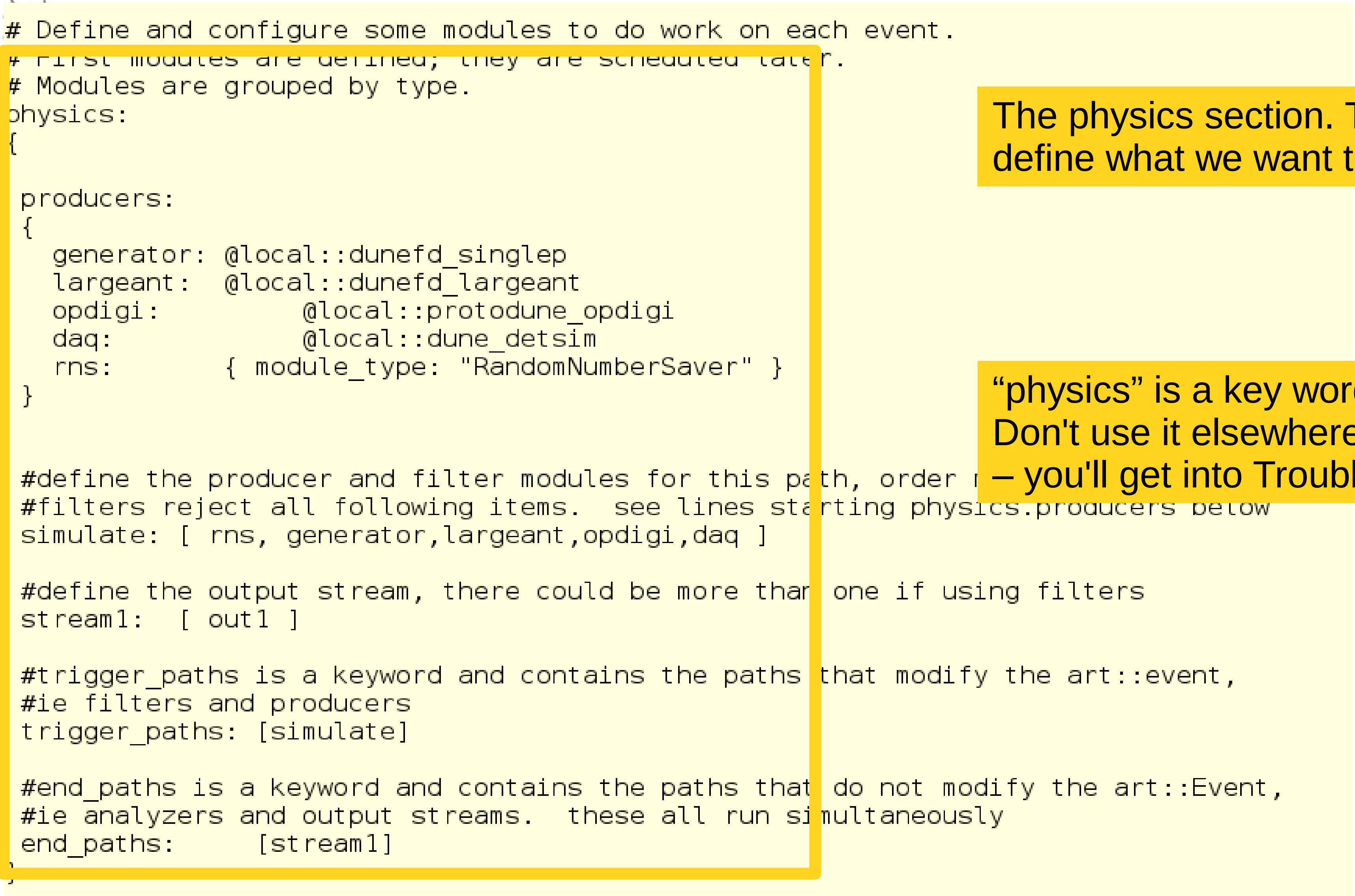
```
#Start each new event with an empty event.
source:
{
  module_type: EmptyEvent
  timestampPlugin: { plugin_type: "GeneratedEventTimestamp" }
  maxEvents:   1000000
  firstRun:    1           # Run number to use for this file
  firstEvent:  1           # number of first event in the file
}
```

Source section.
Defines if there is a source,
Needed for subsequent jobs.

We specified how many events we want (1000000 in this case).

Don't try that. Please. ;-)

ProtoDUNE_gensingle.fcl



```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:       { module_type: "RandomNumberSaver" }
 }


 #define the producer and filter modules for this path, order m
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ rns, generator,largeant,opdigi,daq ]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

The physics section. This is where we define what we want to run.

“physics” is a key word defined by ART Don't use it elsewhere in a .fcl file – you'll get into Trouble.

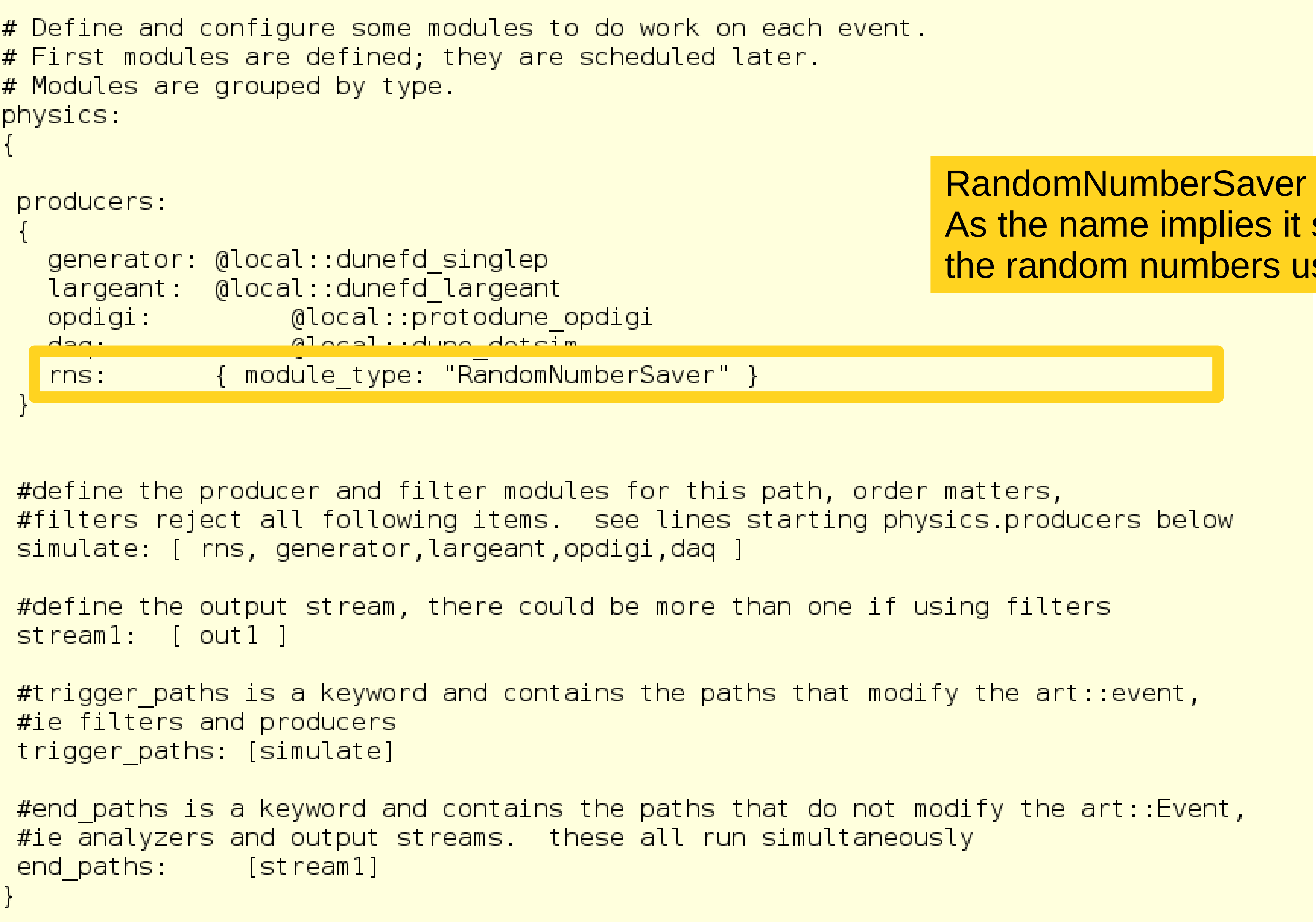
ProtoDUNE_gensingle.fcl

```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:        { module_type: "RandomNumberSaver" }
 }

 #define the producer and filter modules for this path, order matters,
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ rns, generator,largeant,opdigi,daq ]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

Producers: this is where we define
The producer modules we
want to run.

“producers” is a key word defined by ART
Don't use it elsewhere in .fcl
– you'll get into trouble.

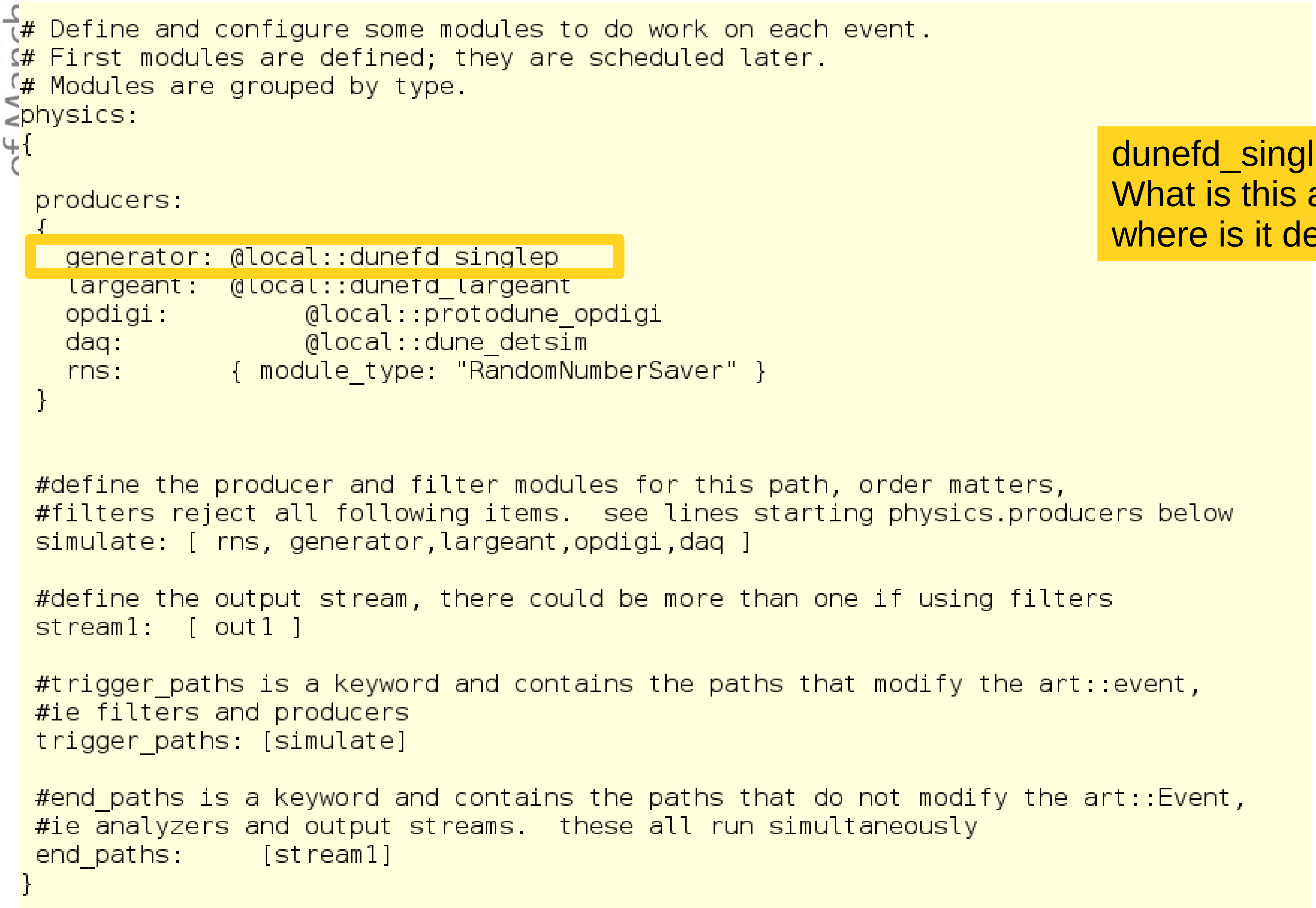
ProtoDUNE_gensingle.fcl



```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:       { module_type: "RandomNumberSaver" }
 }


 #define the producer and filter modules for this path, order matters,
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ rns, generator,largeant,opdigi,daq ]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

RandomNumberSaver module.
As the name implies it saves
the random numbers used by the job.

ProtoDUNE_gensingle.fcl



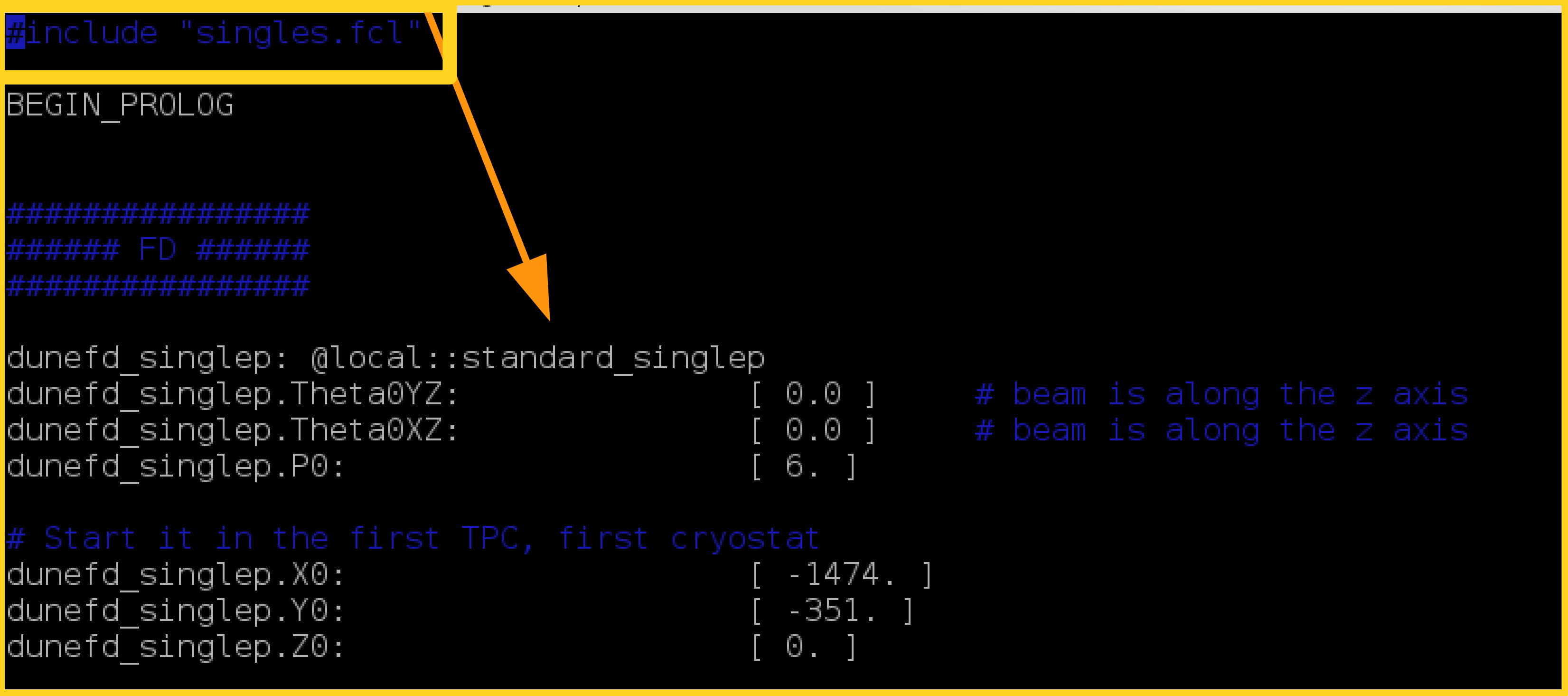
```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:       { module_type: "RandomNumberSaver" }
 }


 #define the producer and filter modules for this path, order matters,
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ rns, generator,largeant,opdigi,daq ]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

dunefd_singlep.
What is this and
where is it defined?

singles_dune.fcl

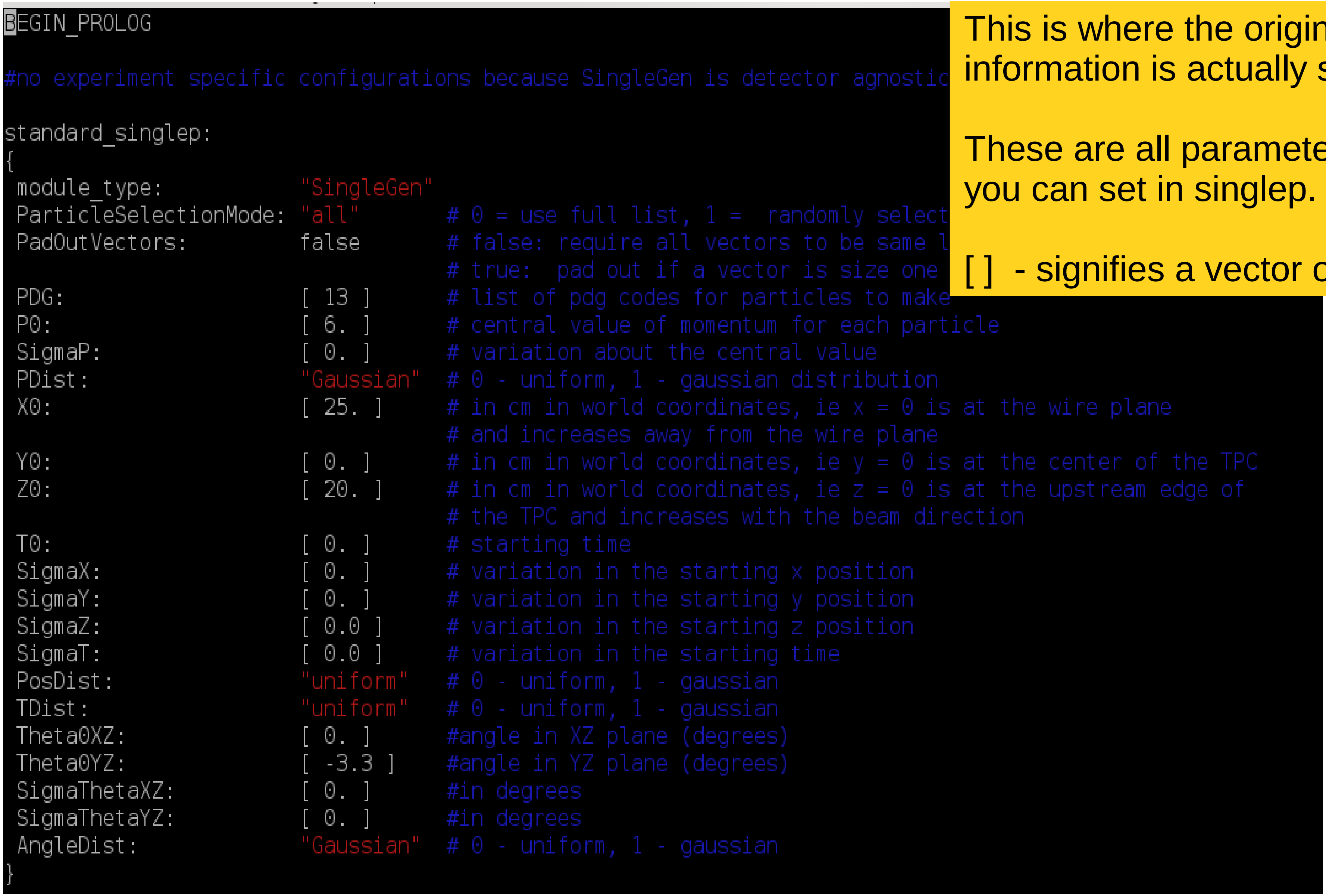
```
#include "services_dune.fcl"
#include "singles_dune.fcl"
#include "largeantmodules_dune.fcl"
#include "photpropservices_dune.fcl"
#include "opticaldetectormodules_dune.fcl"
#include "detsimmodules_dune.fcl"
#include "tools_dune.fcl"

process_name: SinglesGen

services:
{
  # Load the
  TFileServic
  TimeTracker
  RandomNumbe
  FileCatalogM
  @table::prot
}
```

This doesn't tell us what we need .



```
#include "singles.fcl"

BEGIN_PROLOG


################
###### FD ######
################

dunefd_singlep: @local::standard_singlep
dunefd_singlep.Theta0YZ:                    [ 0.0 ]     # beam is along the z axis
dunefd_singlep.Theta0XZ:                    [ 0.0 ]     # beam is along the z axis
dunefd_singlep.P0:                          [ 6. ]

# Start it in the first TPC, first cryostat
dunefd_singlep.X0:                          [ -1474. ]
dunefd_singlep.Y0:                          [ -351. ]
dunefd_singlep.Z0:                          [ 0. ]
```

singles.fcl



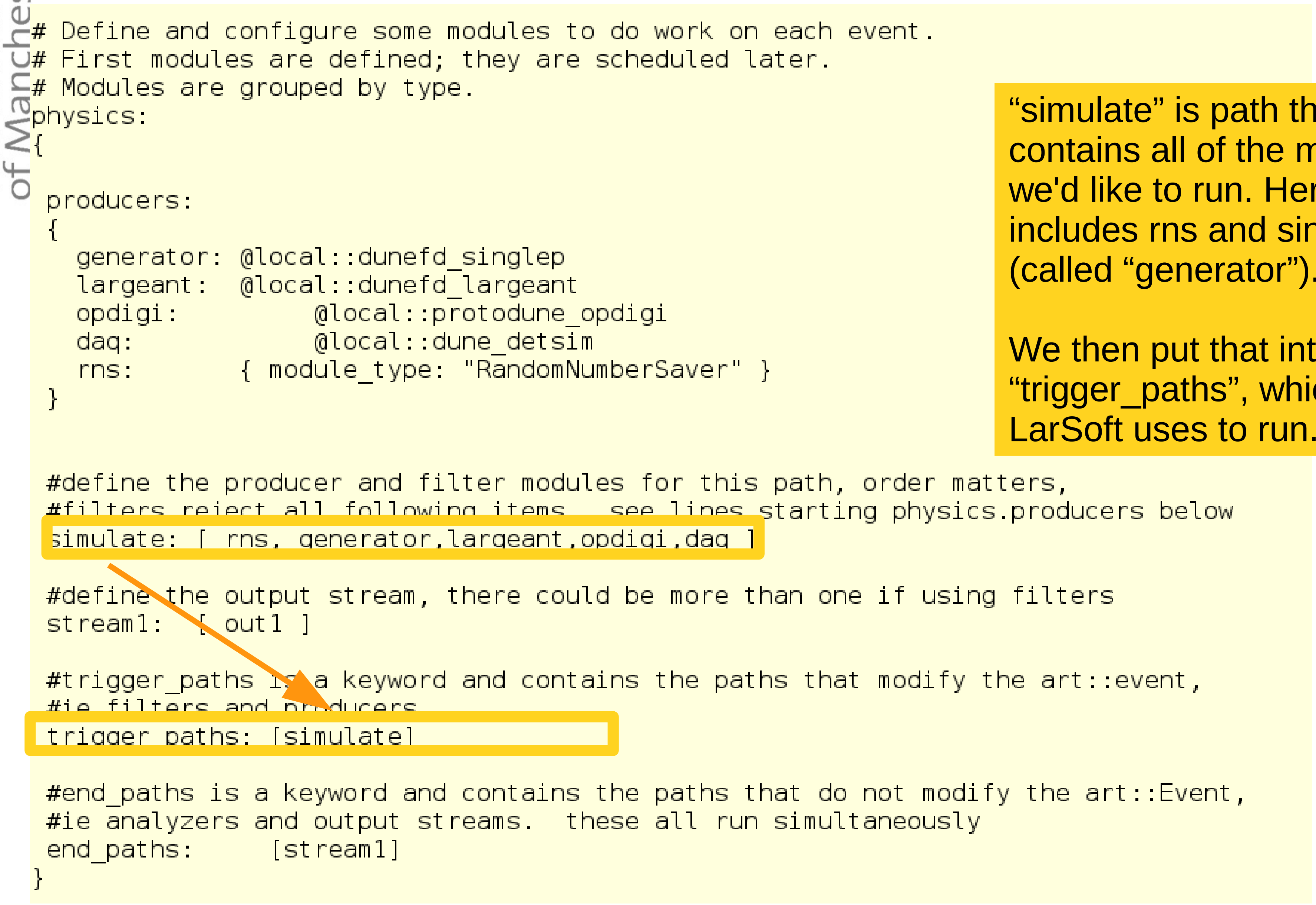
```
BEGIN_PROLOG

#no experiment specific configurations because SingleGen is detector agnostic

standard_singlep:
{
 module_type:           "SingleGen"
 ParticleSelectionMode: "all"       # 0 = use full list, 1 =  randomly select
 PadOutVectors:         false       # false: require all vectors to be same l
                                    # true:  pad out if a vector is size one
 PDG:                   [ 13 ]      # list of pdg codes for particles to make
 P0:                    [ 6. ]      # central value of momentum for each particle
 SigmaP:                [ 0. ]      # variation about the central value
 PDist:                 "Gaussian"  # 0 - uniform, 1 - gaussian distribution
 X0:                    [ 25. ]     # in cm in world coordinates, ie x = 0 is at the wire plane
                                    # and increases away from the wire plane
 Y0:                    [ 0. ]      # in cm in world coordinates, ie y = 0 is at the center of the TPC
 Z0:                    [ 20. ]     # in cm in world coordinates, ie z = 0 is at the upstream edge of
                                    # the TPC and increases with the beam direction
 T0:                    [ 0. ]      # starting time
 SigmaX:                [ 0. ]      # variation in the starting x position
 SigmaY:                [ 0. ]      # variation in the starting y position
 SigmaZ:                [ 0.0 ]     # variation in the starting z position
 SigmaT:                [ 0.0 ]     # variation in the starting time
 PosDist:               "uniform"   # 0 - uniform, 1 - gaussian
 TDist:                 "uniform"   # 0 - uniform, 1 - gaussian
 Theta0XZ:              [ 0. ]      #angle in XZ plane (degrees)
 Theta0YZ:              [ -3.3 ]    #angle in YZ plane (degrees)
 SigmaThetaXZ:          [ 0. ]      #in degrees
 SigmaThetaYZ:          [ 0. ]      #in degrees
 AngleDist:             "Gaussian"  # 0 - uniform, 1 - gaussian
}
```

This is where the original information is actually stored.

These are all parameters that you can set in singlep.

[ ] - signifies a vector of values

ProtoDUNE_gensingle.fcl

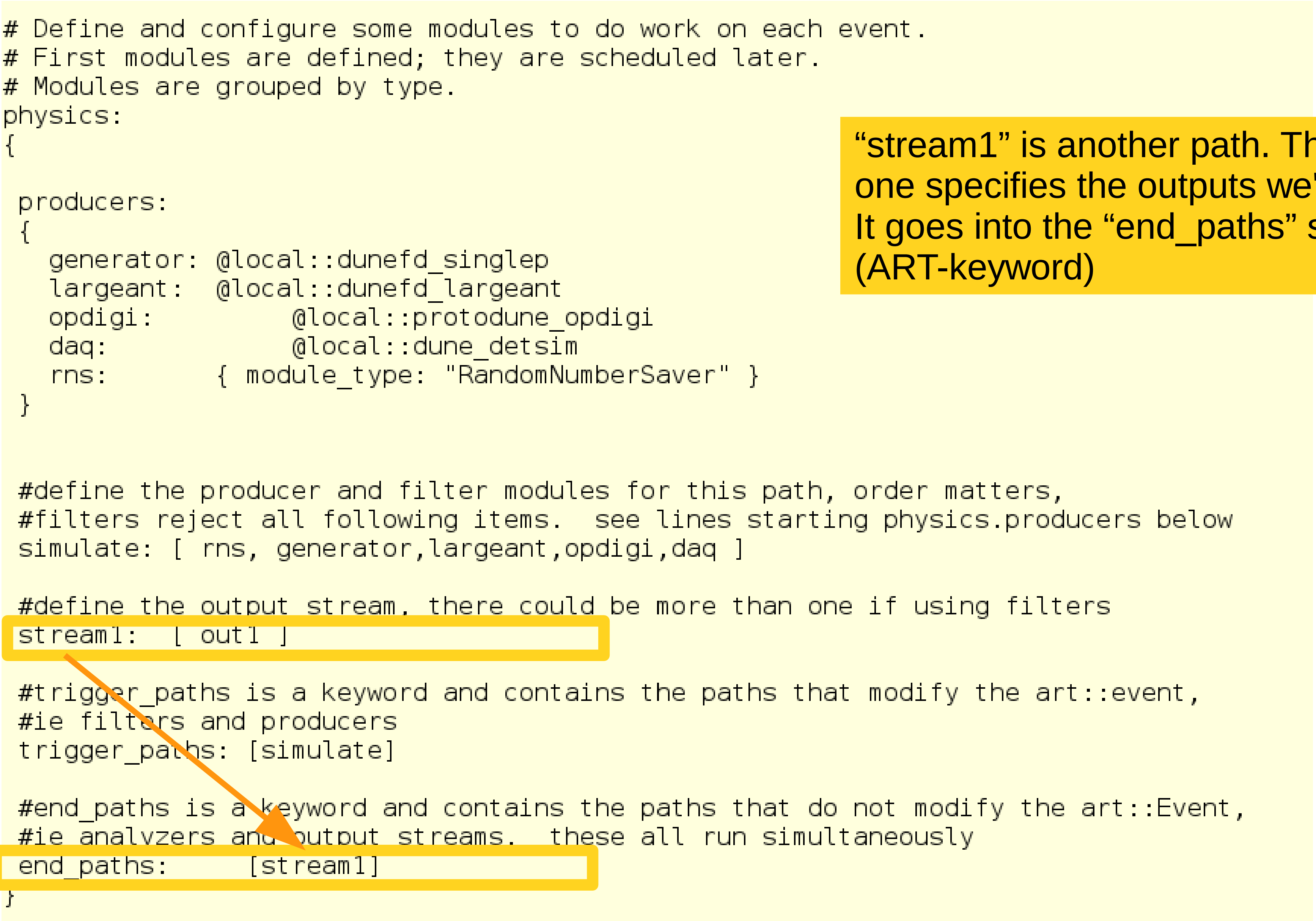


```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:       { module_type: "RandomNumberSaver" }
 }


 #define the producer and filter modules for this path, order matters,
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ rns, generator,largeant,opdigi,daq ]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

“simulate” is path that contains all of the modules we'd like to run. Here this includes rns and singlep (called “generator”).

We then put that into “trigger_paths”, which LarSoft uses to run.

ProtoDUNE_gensingle.fcl

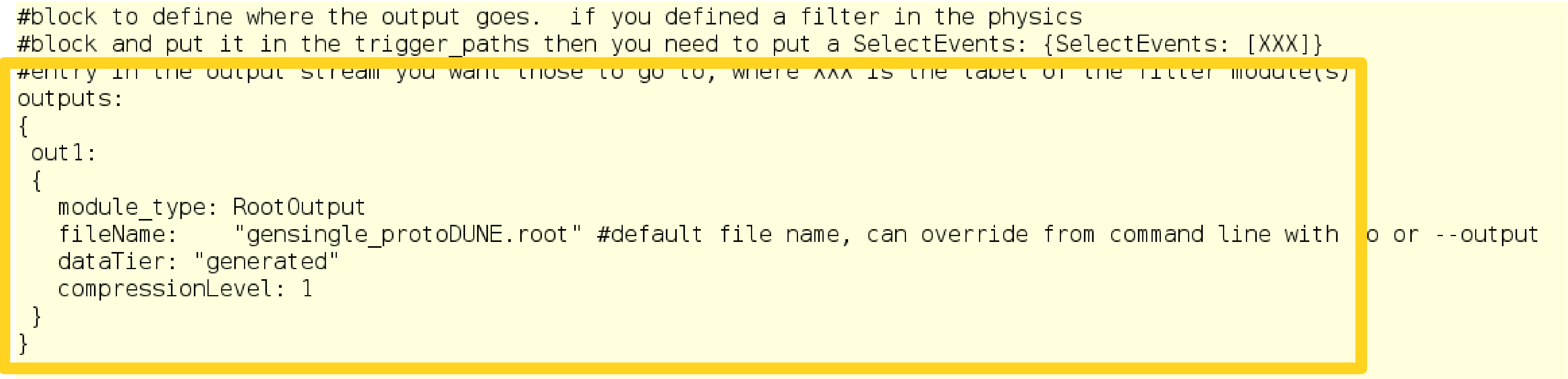


```
# Define and configure some modules to do work on each event.
# First modules are defined; they are scheduled later.
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:       { module_type: "RandomNumberSaver" }
 }


 #define the producer and filter modules for this path, order matters,
 #filters reject all following items.  see lines starting physics.producers below
 simulate: [ rns, generator,largeant,opdigi,daq ]

 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

“stream1” is another path. This one specifies the outputs we'd like. It goes into the “end_paths” section. (ART-keyword)

ProtoDUNE_gensingle.fcl

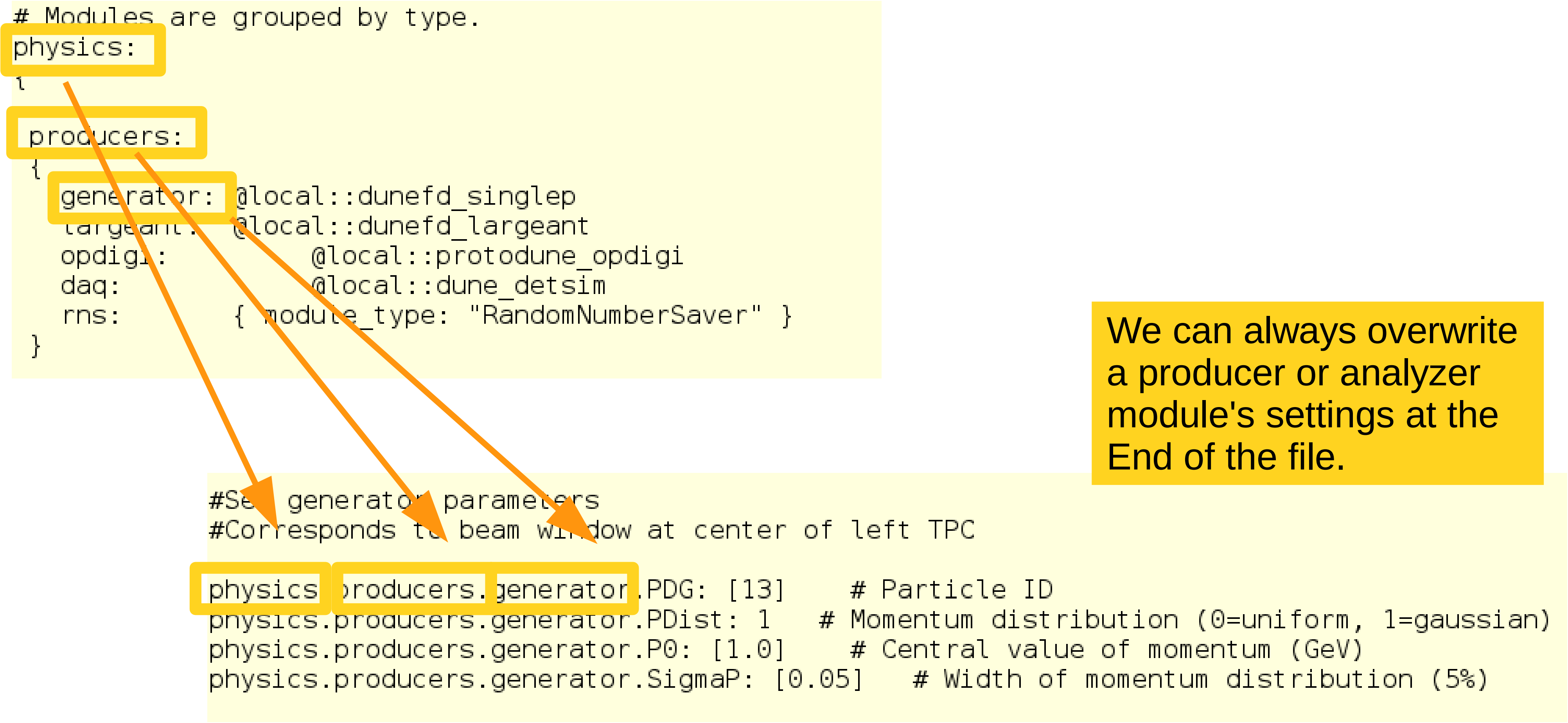
What the job produces is defined
in the “outputs” section.
This is an ART keyword.



```
#block to define where the output goes.  if you defined a filter in the physics
#block and put it in the trigger_paths then you need to put a SelectEvents: {SelectEvents: [XXX]}
#entry in the output stream you want those to go to, where XXX is the label of the filter module(s)
outputs:
{
 out1:
 {
   module_type: RootOutput
   fileName:    "gensingle_protoDUNE.root" #default file name, can override from command line with -o or --output
   dataTier: "generated"
   compressionLevel: 1
 }
}
```

In this particular case, we are
Requesting the output as
a .root file called:
“gensingle_protoDUNE.root”

ProtoDUNE_gensingle.fcl



```
# Modules are grouped by type.
physics:
{

 producers:
 {
   generator: @local::dunefd_singlep
   largeant:  @local::dunefd_largeant
   opdigi:         @local::protodune_opdigi
   daq:            @local::dune_detsim
   rns:       { module_type: "RandomNumberSaver" }
 }
```

```
#Set generator parameters
#Corresponds to beam window at center of left TPC

physics.producers.generator.PDG: [13]    # Particle ID
physics.producers.generator.PDist: 1   # Momentum distribution (0=uniform, 1=gaussian)
physics.producers.generator.P0: [1.0]    # Central value of momentum (GeV)
physics.producers.generator.SigmaP: [0.05]   # Width of momentum distribution (5%)
```

We can always overwrite a producer or analyzer module's settings at the End of the file.

4. Let's get some code and compile it.

- `cd $MRB_SOURCEDIR`
- `mrb g -t v07_11_00 dunetpc` //git clone repository
  - //optional: cd dunetpc, git branch -a (lists all the existing branches)
- `cd $MRB_BUILDDIR`
- `mrbsetenv` //check whether all dependencies are ok
- `mrb i -j4` // compile and install
- `mrbslp` //set up the products in your localProducts directory

This is the standard and Proper way to do it.

BUT HOLD OFF!
We will be doing something different to save time!

We now have our own compiled Version of dunetpc, that exists In localProducts.

We could make changes to it, Recompile, and see what happens.

4.1 Hacky alternative

- In case the mrb g from Fermilab is very slow, try:

-
  ```
  git clone /home/andrzej/larsoft/srcs/dunetpc/.git

  mrb uc
  cd dunetpc
  git checkout tags/v07_11_00
  cd $MRB_BUILDDIR
  mrbsetenv
  mrb i -j4
  ```

This clones a dunetpc repository I pre-downloaded. Normally you would never work like this, but it works...

Bonus task:
Checkout larsim and larana from
The same directory and checkout:
feature/andrzej_wkshop

Closing remarks

- If you have time Familiarize yourself with what is in the dunetpc directory. Take a look at other available .fcl files.
- Now that you know how to start a LarSoft job, we will look into how simulation works in LarSoft.
- You can leave dunetpc compiling in the background.

Glossary/Backups

4. Let's locate the .fcl files we used earlier.

- `gen_protoDune_pion_2GeV_mono.fcl`
- `protoDUNE_g4.fcl`
- `protoDUNE_detsim.fcl`
- `evd_protoDUNE_noped.fcl`

- They should be in more than one place in your working directory. Where?
- Which one of them gets used when you launch a job? (this is important to know!)

MRB (2)
- tips from the battle field

- Do **not** run in the build directory – an mrb z (make clean) will delete everything (ouch).
- Do **not** edit the .fcl or .gdml files in the build (futile) or localProducts... (will get overwritten at compilation - ouch) directories.
- **Do** edit the .fcl files in srcs/ and copy them over to localProducts by “mrb i” or “make install”.
- **Know** which files you are using: $FHICL_FILE_PATH, $FW_SEARCH_PATH and others define this.

5. Let's break something!

- Often we will need more than one repository, as our changes will have interplays between different pieces of code. This can sometimes cause trouble.
- `cd $MRB_SOURCEDIR`
- `mrb g larpandora` //git clone repository, containing simulation parts
  - //optional: cd dunetpc, git branch -a (lists all the existing branches)
- `cd $MRB_BUILDDIR`
- `mrbsetenv` //check whether all dependencies are ok
- `mrb i -j4` // compile and install
- `mrbslp` //set up the products in your localProducts directory

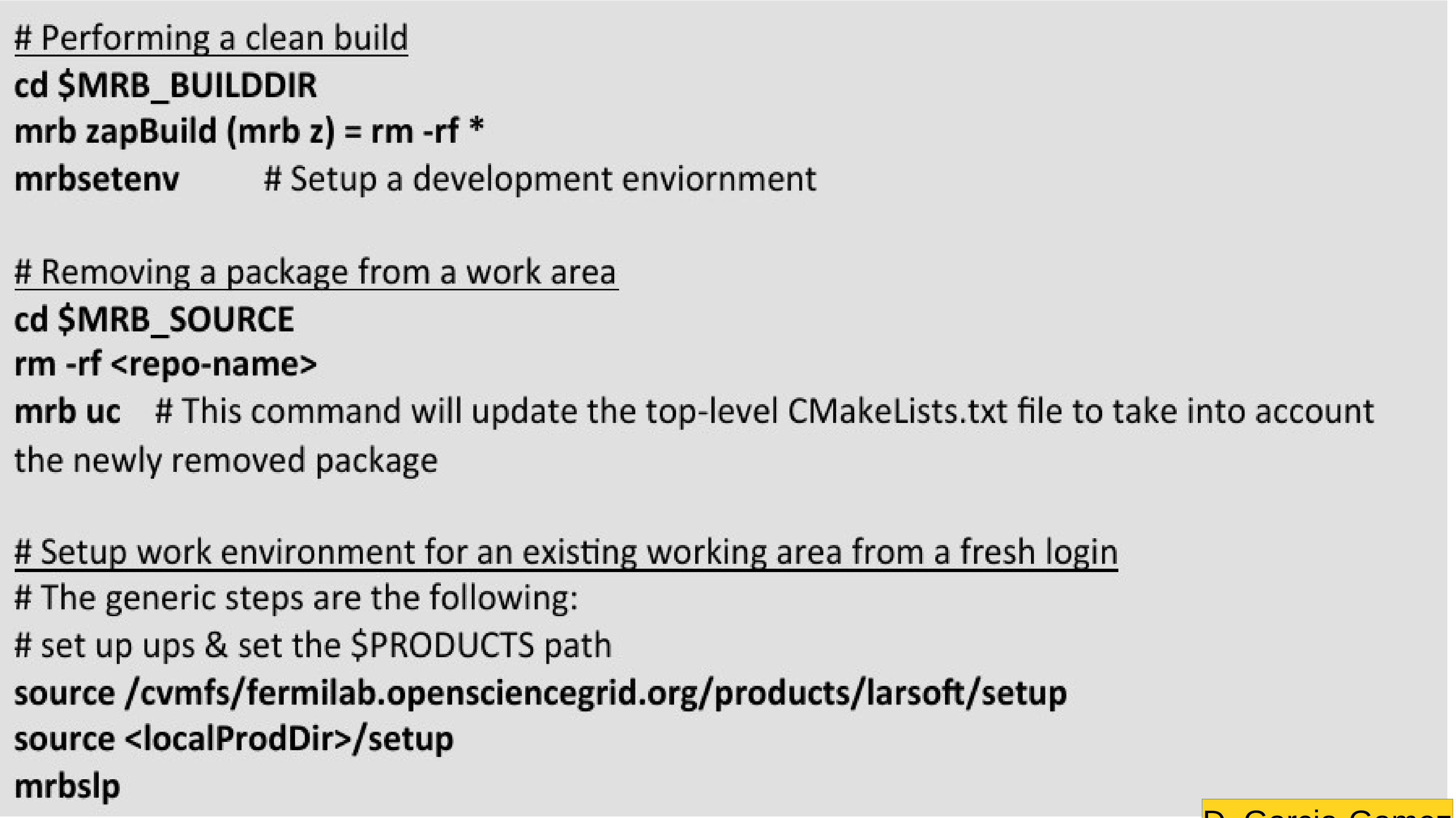
This will fail!
Not as badly
as it used to.

5.1 What went wrong?

- `mrb g larpandora` // checks out the newest version of the repository.
- `mrb g -t v06_85_00 dunetpc` //we checked out a specific (older) version of the dune repository.
- **These are not compatible. :(**
- We should have done:
  - `mrb g -t LARSOFT_SUITE_v06_85_00 larpandora`
    // note the clunky tag name. Different repositories have different numbering schemes and this is how they are aligned.
- We can either remove package (previous page) and reinstall, or:
- `cd $MRB_SOURCEDIR/larpandora`
- `git checkout tags/LARSOFT_SUITE_v06_85_00` //we have the code in the repository, we just need to bring it to the "front".
- `cd $MRB_BUILDDIR`
- `mrb z` //clean up everything.
- `mrbsetenv` //check whether all dependencies are ok
- `mrb i -j4` // compile and install
- `mrbslp` //set up the products in your localProducts directory

This should
Hopefully work now!

Some standard fixes to test



```
# Performing a clean build
cd $MRB_BUILDDIR
mrb zapBuild (mrb z) = rm -rf *
mrbsetenv        # Setup a development enviornment

# Removing a package from a work area
cd $MRB_SOURCE
rm -rf <repo-name>
mrb uc    # This command will update the top-level CMakeLists.txt file to take into account
the newly removed package

# Setup work environment for an existing working area from a fresh login
# The generic steps are the following:
# set up ups & set the $PRODUCTS path
source /cvmfs/fermilab.opensciencegrid.org/products/larsoft/setup
source <localProdDir>/setup
mrbslp
```

D. Garcia-Gamez

MRB
- basic trouble shooting

- `mrb newDev`
- `source localProducts.../setup`
- `cd $MRB_SOURCEDIR`
- `mrb g -t <right version> dunetpc` //git clone repository
- `cd $MRB_BUILDDIR`
- `mrbsetenv` //check whether all dependencies are ok
- `mrb i -j4` // compile and install
- `mrbslp` //set up your localProducts directory

This is where trouble happens.

MRB
- basic trouble shooting (2)

- `mrbsetenv` //check whether all dependencies are ok
- `mrb i -j4` // compile and install
- `mrbslp` //set up your localProducts directory

This is where trouble happens.

- More often than not, you are either missing a product version that is a dependency – check whether it is available:
  - ups list -aK+ <productname>
  - echo $PRODUCTS
  - ls <product directories>   – make sure it is there.
- Or you have set up a version of a dependency that clashes with one that you need for one of your other packages:
  - ups active <productname>
  - ls srcs/<repository>/ups/product_deps - do they clash?
  - Try unsetup <productname>