

# SKFlatAnalyzer User Guide

Jae Sung Kim  
Seoul National University  
[jae.sung.kim@cern.ch](mailto:jae.sung.kim@cern.ch)

January 24, 2019

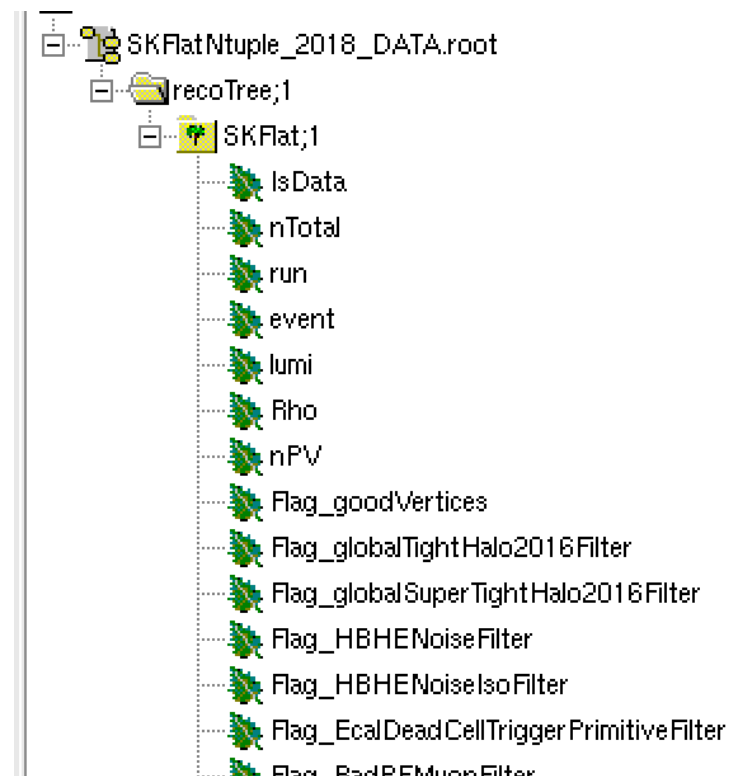
# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	SKFlat . . . . .	2
1.2	SKFlatAnalyzer . . . . .	2
<b>2</b>	<b>Directories</b>	<b>4</b>
2.1	DataFormats/ . . . . .	4
2.2	Analyzers/ . . . . .	4
2.3	include/ . . . . .	4
2.4	src/ . . . . .	4
2.5	data/\$SKFlatV . . . . .	4
2.6	python/ . . . . .	4
2.7	script/ . . . . .	4
2.8	lib/ . . . . .	4
<b>3</b>	<b>Structure</b>	<b>5</b>
3.1	Analyzer inheritance . . . . .	5
3.2	Physics object inheritance . . . . .	6
<b>4</b>	<b>Analyzer class and submission command</b>	<b>7</b>
4.1	SKFlatNtuple . . . . .	7
4.2	AnalyzerCore . . . . .	7
4.3	MyAnalyzer . . . . .	8
4.4	Job macro . . . . .	8
4.5	SKFlat.py . . . . .	9
<b>5</b>	<b>Macro run order</b>	<b>10</b>
5.1	Example of a macro with comments inline . . . . .	10
<b>6</b>	<b>Migration from CATAnalyzer</b>	<b>11</b>
<b>7</b>	<b>Rules for developers</b>	<b>12</b>
7.1	File/Function/Variable names are important . . . . .	12
7.2	Equality operator between float or double . . . . .	12
7.3	std::map is good, but be careful . . . . .	12
7.4	When using random variables.. . . . .	13

# 1 Introduction

## 1.1 SKFlat

- A flat ntuple
- Use MiniAOD as an input
- Github link : <https://github.com/CMSSNU/SKFlatMaker>



## 1.2 SKFlatAnalyzer

- ROOT6 based analyzer
- SNU (tamsa1), KISTI and KNU batch are supported by same submission commands (2019.01.22)
- Use SKFlat as an input
- Run over each event, and do the analysis!!
- Construct physics objects using branch elements:

```
Muon mu;  
double rc = muon_roch_sf->at(i);  
double rc_err = muon_roch_sf_up->at(i);  
mu.SetMiniAODPt(muon_pt->at(i));  
mu.SetPtEtaPhiM(muon_pt->at(i)*rc, muon_eta->at(i), muon_phi  
->at(i), muon_mass->at(i));
```

- GitHub link : <https://github.com/CMSSNU/SKFlatAnalyzer>

## **2 Directories**

### **2.1 DataFormats/**

Physics objects

### **2.2 Analyzers/**

Physics objects

### **2.3 include/**

Header files

### **2.4 src/**

Source files (define class, functions, ...)

### **2.5 data/\$SKFlatV**

Various data files including .root, .txt, ...

E.g., fake rates, scale factors,

Defined as an environment variable, \$DATA\_DIR

### **2.6 python/**

Python scripts for job submission

### **2.7 script/**

Any useful scripts

### **2.8 lib/**

Compiled shared-libraries moved here

### 3 Structure

#### 3.1 Analyzer inheritance

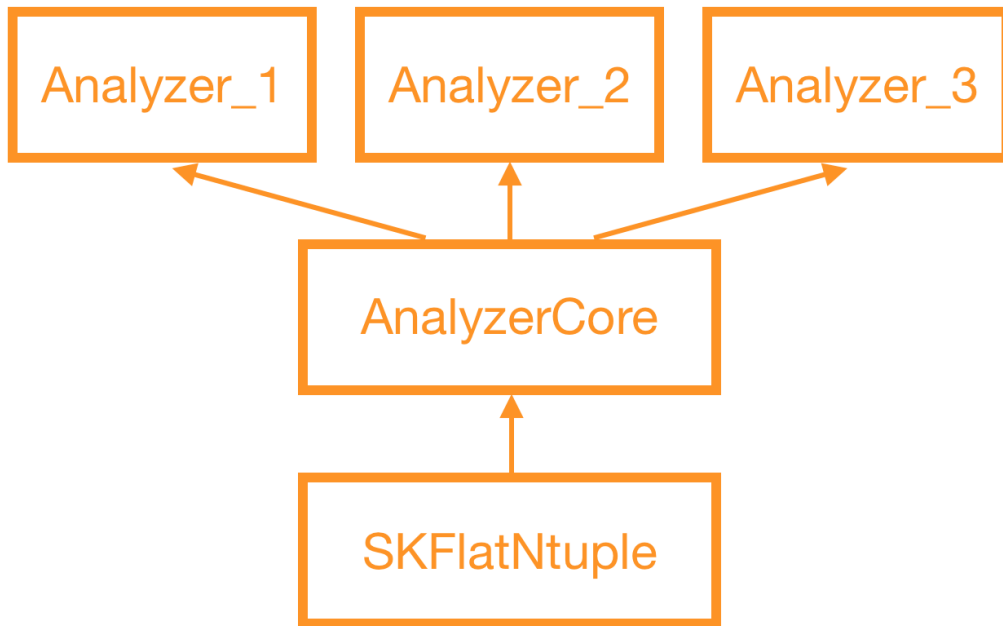


Figure 1: Diagram of analyzer inheritance.

### 3.2 Physics object inheritance

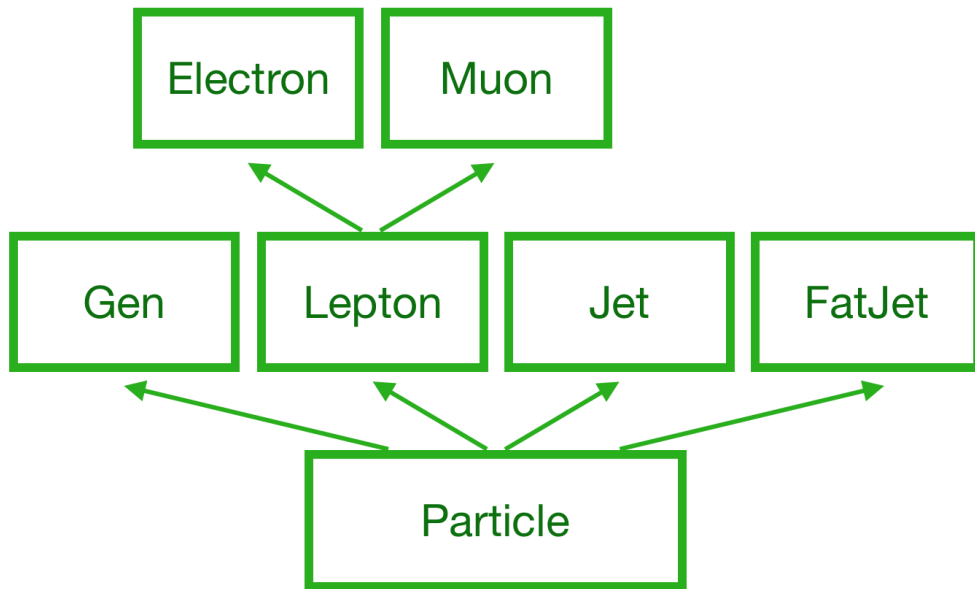


Figure 2: Diagram of physics object inheritance.

## 4 Analyzer class and submission command

*Every analyzer inherits AnalyzerCore  
AnalyzerCore inherits SKFlatNtuple*

### 4.1 SKFlatNtuple

- Almost same as the output from TTree::MakeClass()
- SKFlatNtuple::Loop() loops over each event

### 4.2 AnalyzerCore

- Inherits SKFlatNtuple
- Includes header files of physics objects classes
- Physics analysis functions

```
std::vector<Muon> AnalyzerCore::GetAllMuons(); // return all
muons
std::vector<Muon> AnalyzerCore::GetMuons(TString id, double
ptmin, double fetamax); // return muons passing ID
selection
std::vector<Muon> AnalyzerCore::SelectMuons(std::vector<Muon
> muons, TString id, double ptmin, double fetamax); //
Select muons passing id out of pre-collected muon
collections
```

- Histogram related functions

```
FillHist(TString histname, double value, double weight, int
n_bin, double x_min, double x_max); // histogram is
saved in the default directory of the output root file
JSFillHist(TString suffix, TString histname, double value,
double weight, int n_bin, double x_min, double x_max);
// histogram is saved in the directory named "suffix" of
the output root file
```

- (Example)

```
vector<Electron> electrons = GetElectrons(param.
Electron_Tight_ID, 10., 2.5);
for(unsigned int i=0; i<electrons.size(); i++){

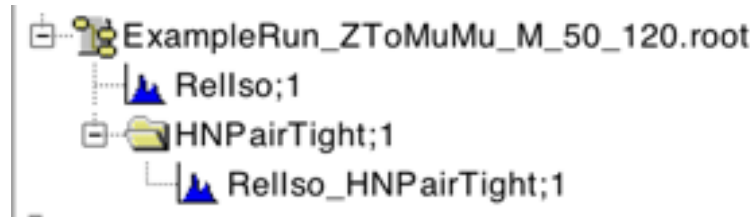
    Electron el = electrons.at(i);
```



```

    FillHist("RelIso", el.RelIso(), 1, 100, 0., 1.);
    JSFillHist(param.Electron_Tight_ID, "RelIso_"+param.
Electron_Tight_ID, el.RelIso(), 1, 100, 0., 1.);
}

```



- Even if two histograms are in different directories, if their names are the same, we have warning message : “Warning in <TFile::Append>: Replacing existing TH1: RelIso (Potential memory leak).”
- So I recommend you to add directory name as a prefix/suffix of the histogram name :  
Instead of “RelIso” alone, use “RelIso\_” + <Directory Name>

### 4.3 MyAnalyzer

- Inherits AnalyzeCore
- Run by the job macro

### 4.4 Job macro

- Macro will be created *automatically* by SKFlat.py command
- MyAnalyzer object is declared
- Input sample information ([DATA] DataStream / [MC] Sample name, input files, xsec, sumW) is set
- Output file path is set
- SKFlatNtuple::Init() is run : Initializing branch element variables
- AnalyzerCore::initializeAnalyzer() is run
  - This function is virtual, and can be redefined in ExampleRun

- Anything you want to do before the event loop can be done here
- Userflag is supported by python/SKFlat.py, by the option  
“--userflags flag1,flag2,flag3”
- The existence of a flag can be checked by using  
AnalyzerCore::HasFlag(TString flag)
- SKFlatNtuple::Loop() is run : loop over events
- AnalyzerCore::WriteHist() is run : write histograms in the output

## 4.5 SKFlat.py

Script for batch job submission

## 5 Macro run order

### 5.1 Example of a macro with comments inline

```
R_LOAD_LIBRARY(libPhysics.so)
R_LOAD_LIBRARY(libTree.so)
R_LOAD_LIBRARY(libHist.so)
R_LOAD_LIBRARY(./lib/libDataFormats.so)
R_LOAD_LIBRARY(./lib/libAnalyzers.so)

void run(){

    //==== Declaring an analyzer class immediately runs followings
    //in orders;
    //==== 1) Constructor of SKFlatNtuple is called
    //==== 2) Constructor of AnalyzerCore is called
    //==== 3) Constructor of ExampleRun is called
    ExampleRun m;

    //==== SKFlat ntuple directory structure..
    m.SetTreeName("recoTree/SKFlat");

    //==== DATA or MC?
    m.IsDATA = true;
    //==== If DATA, PD name
    m.DataStream = "SingleMuon";
    //==== DATA year
    m.DataYear = 2016;
    //==== Files to be ran with this macro
    m.AddFile("SKFlatNtuple_2016_DATA_100.root");
    //==== output rootfile path
    m.SetOutfilePath("hists.root");
    //==== SKFlatNtuple::Init(), which does SetBranchAddress()
    m.Init();
    //==== AnalyzerCore::initializeAnalyzerTools Read histograms
    //or initialize MCCorrection helpers or data-driven estimators
    m.initializeAnalyzerTools();
    //==== Any initialization just before running event loop. This
    //is only ran once within a macro. For example, you should run
    //AnalyzerCore::HasFlag() here. More example can be found HERE
    m.initializeAnalyzer();
    //==== Finally, run event loops
    m.Loop();

    //==== All events are ran. Now write histograms to the output
    //rootfile
    m.WriteHist();
}
```

## 6 Migration from CATAnalyzer

Direct copy from CATAnalyzer codes to SKFlatAnalyzer won't work, but here are some tips.

- `FillHist(histname, variable, weight, x_min, x_max, n_bin)`  
→ `FillHist(histname, variable, weight, n_bin, x_min, x_max)`  
: follow the order of arguments of TH1 in ROOT

## 7 Rules for developers

Some rules you should follow, if you want to make a pull request to the master branch.

### 7.1 File/Function/Variable names are important

Please, spend at least 10 minutes before you create a new file/function/variable. Good naming makes programming efficient.

### 7.2 Equality operator between float or double

Guess what you would get from “root -l -b -q test.C” with below.

```
float GetFatJetSF(float tau21cut){  
    if(tau21cut == 0.45){  
        return 0.45;  
    }  
    if(tau21cut == 0.6){  
        return 0.6;  
    }  
    else{  
        return 1.;  
    }  
}  
  
void test(){  
    cout << "Value : " << GetFatJetSF(0.45) << endl;  
}
```

Result is **Value : 1.** It works properly if you change **float GetFatJetSF(float tau21cut)** to **float GetFatJetSF(double tau21cut)**. However, it is NOT recommended to apply equality operator between floats. If you really need it, you can do  $|A - B| < e$  with a very small  $e$  (e.g., 0.001). Similarly, never do **if(value == -999.){ ... }**.

### 7.3 std::map is good, but be careful

We use a lot of `std::map` in the analyzer; rootfile for MCCorrection are saved as “`std::map<TString, TH1D> histmap`”, and histogram can be accessed by “`histmap[key]`”. But if you store so many histograms into the map, it spends

so much time to obtain “histmap[mykey]”, because it checks “mykey==key” for each keys. If you have saved thousands of fake-rate histograms into a map and run a fake estimation, it will take years... If you are applying muon scale factors, “map\_hist\_Muon[YOUR\_ID]” is ran for each event and each muons. If you wrote too many IDs in ID/Muon/histmap.txt, you will waste your time looping over unnecessary keys. To save your time, you can add a “#” at the beginning of each lines in “ID/Muon/histmap.txt” (i.e., deactivating it) :

```
ID SF NUM_MediumID_DEN_genTracks RunAveraged_SF_ID.root NUM_MediumID_DEN_genTracks_eta_pt
                                ↓
#ID SF NUM_MediumID_DEN_genTracks RunAveraged_SF_ID.root NUM_MediumID_DEN_genTracks_eta_pt
```

Then histogram for Medium ID will not be saved in the histmap.

## 7.4 When using random variables..

Some functions use random variables (e.g., smearing from a PDF). If you use default random seed, your results can be changed everytime you run the analyzer. Easiest way to avoid this issue is using a combination of RunNumber and EventNumber as a seed. E.g., seed = RunNumber  $\times$  1000000000 + EventNumber.