Outline

- Motivation
- What we had before
- What have now
- One example
- Planned enhancements
- Outlook

Improving FCCAnalyses

FCC Week 2022, Paris

May 31, 2022
C. Helsens
KIT

Motivation

- Analysis framework are in constant evolution and always need to follow state of the art solutions and suggestions from user community to improve/facilitate the usage
- In the process of re-writing the batch interface, we realised that a lot of improvements could be obtained
- Decided to proceed with changing the processing logic and this triggered a lot of new ideas for developments that are explained in this talk
- User feedback on the new developments and the one planned is still very valuable
    - Post your questions / request on the FCC Forum under FCCAnalyses category https://fccsw-forum.web.cern.ch/ or using github issues, pull requests https://github.com/HEP-FCC/FCCAnalyses/

What is FCCAnalyses

FCCAnalyses is a common analysis framework for FCC related analyses

- It is based on Root DataFrames
- It is composed of python wrappers for ease of use
  - connection with database of common samples of events
  - utilities for batch processing
  - Full analysis cycle: pre-selection, final selection, histograms, final TTrees and plotting
- Can read EDM4Hep format
- Operations on the events are done
  - Directly using inline definitions: Define(“pt_100”,”pt[pt>100]”)
  - Defining inline C++: .Define(“xx”, [&x] {return x*x;})
  - Writing your own C++ functions and add them to the FCCAnalyses dictionnary of analysers of provide them through ROOT.gInterpreter.Declare(“””....some code…”””)

What we had before

Each analysis is hosted in a single directory, for example `examples/FCCee/higgs/mH-recoil/mumu/` and contains the same kind of files, please use the same naming convention for all analysis.

1. `analysis.py` : This class that is used to define the list of analysers and filters to run on as well as the output variables.
2. `preSel.py` : This configuration file is used to define how to run the `analysis.py` . It contains the list of samples, the number of CPUs, the fraction of the original sample to process and the base directory for the yaml files (that contains the informations about the samples). This will run the `analysis.py` with a common code `config/runDataFrame.py` (this last file is common to all analyses and should not be touched).

- The analysis.py file contained a lot of analysis specific code to run over single file, write outputs, import various modules etc…
- Configuration was done in the preSel.py user script
- This was bringing a lot of code duplication, thus error prone -> simplifications needed!!

What we had before

Not showing the details for other running modes

Each analysis is hosted in a single directory, for example `examples/FCCee/higgs/mH-recoil/mumu/` and contains the same kind of files, please use the same naming convention for all analysis.

1. `analysis.py` : This class that is used to define the list of analysers and filters to run on as well as the output variables.
2. `preSel.py` : This configuration file is used to define how to run the `analysis.py` . It contains the list of samples, the number of CPUs, the fraction of the original sample to process and the base directory for the yaml files (that contains the informations about the samples). This will run the `analysis.py` with a common code `config/runDataFrame.py` (this last file is common to all analyses and should not be touched).

```
python examples/FCCee/higgs/mH-recoil/mumu/preSel.py

python examples/FCCee/higgs/mH-recoil/mumu/finalSel.py

python config/doPlots.py examples/FCCee/higgs/mH-recoil/mumu/plots.py
```

What we have now - 1

Analyses in the FCCAnalyses framework usually follow standardized workflow, which consists of multiple files inside a single directory. Individual files denote steps in the analysis and have the following meaning:

1. `analysis.py` or `analysis_stage<num>` : In this file(s) the class of type `RDFanalysis` is used to define the list of analysers and filters to run on (`analysers` function) as well as the output variables (`output` function). It also contains the configuration parameters `processList` , `prodTag` , `outputDir` , `inputDir` , `nCPUS` and `runBatch` . User can define multiple stages of `analysis.py` . The first stage will most likely run on centrally produced EDM4hep events, thus the usage of `prodTag` . When running a second analysis stage, user points to the directory where the samples are located using `inputDir` .

- The analysis.py and preSel.py files have been merged, and code duplication removed
- Configuration is now done in analysis.py that should follow some simple nomenclature
- To run over the samples defined inside analysis_stage<num>.py

```
fccanalysis run examples/FCCee/higgs/mH-recoil/mumu/analysis_stage1.py
```

What we have now - 1

Analyses in the FCCAnalyses framework usually follow standardized workflow, which consists of multiple files inside a single directory. Individual files denote steps in the analysis and have the following meaning:

1. `analysis.py` or `analysis_stage<num>` : In this file(s) the class of type `RDFanalysis` is used to define the list of analysers and filters to run on (`analysers` function) as well as the output variables (`output` function). It also contains the configuration parameters `processList` , `prodTag` , `outputDir` , `inputDir` , `nCPUS` and `runBatch` . User can define multiple stages of `analysis.py` . The first stage will most likely run on centrally produced EDM4hep events, thus the usage of `prodTag` . When running a second analysis stage, user points to the directory where the samples are located using `inputDir` .

- The analysis.py and preSel.py files have been merged, and code duplication removed
- Configuration is now done in analysis.py that should follow some simple nomenclature
- To run over files without using the processList

```
fccanalysis run examples/FCCee/higgs/mH-recoil/mumu/analysis_stage1.py \
      --output <myoutput.root> \
      --files-list <file.root or file1.root file2.root or file*.root>
```

Inside analysis_stage1.py (EDM4Hep) - 1

- Configuration part of the analysis_stage1.py

```
#Mandatory: List of processes
processList = {
    'p8_ee_ZZ_ecm240':{},#Run the full statistics in one output file named <outputDir>/p8_ee_ZZ_ecm240.root
    'p8_ee_WW_ecm240':{'fraction':0.5, 'chunks':2}, #Run 50% of the statistics in two files named <outputDir>/p8_ee_WW_ecm240/chunk<N>.root
    'p8_ee_ZH_ecm240':{'fraction':0.2, 'output':'p8_ee_ZH_ecm240_out'} #Run 20% of the statistics in one file named <outputDir>/p8_ee_ZH_ecm240_out.root
}

#Mandatory: Production tag when running over EDM4Hep centrally produced events, this points to the yaml files for getting sample statistics
prodTag     = "FCCee/spring2021/IDEA/"

#Optional: output directory, default is local dir
outputDir   = "ZH_mumu_recoil/stage1"

#Optional: ncpus, default is 4
nCPUS       = 8

```

Inside analysis_stage1.py (EDM4Hep) - 2

```python
#Mandatory: RDFanalysis class where the use defines the operations on the TTree
class RDFanalysis():

    #__________________________________________________________
    #Mandatory: analysers funtion to define the analysers to process, please make sure you return the last dataframe, in this example it is df2
    def analysers(df):
        df2 = (
            df
            # define an alias for muon index collection
            .Alias("Muon0", "Muon#0.index")
            # define the muon collection
            .Define("muons",  "ReconstructedParticle::get(Muon0, ReconstructedParticles)")
            #select muons on pT
            .Define("selected_muons", "ReconstructedParticle::sel_pt(10.)(muons)")
            # create branch with muon transverse momentum
            .Define("selected_muons_pt", "ReconstructedParticle::get_pt(selected_muons)")
            # create branch with muon rapidity
            .Define("selected_muons_y",  "ReconstructedParticle::get_y(selected_muons)")
            # create branch with muon total momentum
            .Define("selected_muons_p",     "ReconstructedParticle::get_p(selected_muons)")
            # create branch with muon energy
            .Define("selected_muons_e",     "ReconstructedParticle::get_e(selected_muons)")
            # find zed candidates from  di-muon resonances
            .Define("zed_leptonic",         "ReconstructedParticle::resonanceBuilder(91)(selected_muons)")
            # create branch with zed mass
            .Define("zed_leptonic_m",       "ReconstructedParticle::get_mass(zed_leptonic)")
            # create branch with zed transverse momenta
            .Define("zed_leptonic_pt",      "ReconstructedParticle::get_pt(zed_leptonic)")
            # calculate recoil of zed_leptonic
            .Define("zed_leptonic_recoil",  "ReconstructedParticle::recoilBuilder(240)(zed_leptonic)")
            # create branch with recoil mass
            .Define("zed_leptonic_recoil_m","ReconstructedParticle::get_mass(zed_leptonic_recoil)")
            # create branch with leptonic charge
            .Define("zed_leptonic_charge","ReconstructedParticle::get_charge(zed_leptonic)")
            # Filter at least one candidate
            .Filter("zed_leptonic_recoil_m.size()>0")
        )
        return df2
```

```python
    #__________________________________________________________
    #Mandatory: output function, please make sure you return the branchlist as a python list
    def output():
        branchList = [
            "selected_muons_pt",
            "selected_muons_y",
            "selected_muons_p",
            "selected_muons_e",
            "zed_leptonic_pt",
            "zed_leptonic_m",
            "zed_leptonic_charge",
            "zed_leptonic_recoil_m"
        ]
        return branchList
```

Inside analysis_stage2.py (custom files) - 1

```
processList = {
    'p8_ee_ZZ_ecm240':{},#Run over the full statistics from stage1 input file <inputDir>/p8_ee_ZZ_ecm240.root. Keep the same output name as input
    'p8_ee_WW_ecm240':{}, #Run over the statistics from stage1 input files <inputDir>/p8_ee_WW_ecm240_out/*.root. Keep the same output name as input
    'p8_ee_ZH_ecm240_out':{'output':'MySample_p8_ee_ZH_ecm240'} #Run over the full statistics from stage1 input file <inputDir>/p8_ee_ZH_ecm240_out.root. Change
}

#Mandatory: input directory when not running over centrally produced edm4hep events.
#It can still be edm4hep files produced standalone or files from a first analysis step (this is the case in this example it runs over the files produced from an
inputDir    = "ZH_mumu_recoil/stage1"

#Optional: output directory, default is local dir
outputDir   = "ZH_mumu_recoil/stage2"

#Optional: ncpus, default is 4
nCPUS       = 2

#Optional running on HTCondor, default is False
runBatch    = False

#USER DEFINED CODE
import ROOT
ROOT.gInterpreter.Declare("""
bool myFilter(ROOT::VecOps::RVec<float> mass) {
    for (size_t i = 0; i < mass.size(); ++i) {
        if (mass.at(i)>80. && mass.at(i)<100.)
            return true;
    }
    return false;
}
""")
#END USER DEFINED CODE
```

Inside analysis_stage2.py (custom files) - 2

```python
#Mandatory: RDFanalysis class where the use defines the operations on the TTree
class RDFanalysis():

    #_____________________________________________________________
    #Mandatory: analysers funtion to define the analysers to process, please make sure you return the last dataframe, in this example it is df2
    def analysers(df):
        df2 = (df
               #Filter to have exactly one Z candidate
               .Filter("zed_leptonic_m.size() == 1")
               #Define Z candidate mass
               .Define("Zcand_m","zed_leptonic_m[0]")
               #Define Z candidate recoil mass
               .Define("Zcand_recoil_m","zed_leptonic_recoil_m[0]")
               #Define Z candidate pt
               .Define("Zcand_pt","zed_leptonic_pt[0]")
               #Define Z candidate charge
               .Define("Zcand_q","zed_leptonic_charge[0]")
               #Define new var rdf entry (example)
               .Define("entry", "rdfentry_")
               #Define a weight based on entry (inline example of possible operations)
               .Define("weight", "return 1./(entry+1)")
               #Define a variable based on a custom filter
               .Define("MyFilter", "myFilter(zed_leptonic_m)")
               )
        return df2

    #_____________________________________________________________
    #Mandatory: output function, please make sure you return the branchlist as a python list.
    def output():
        branchList = [
            "Zcand_m", "Zcand_pt", "Zcand_q","MyFilter","Zcand_recoil_m",
            "entry","weight"
        ]
        return branchList

```

What we have now - 2

2. `analysis_final.py` : This analysis file contains the final selections and it runs over the locally produced n-tuples from the various stages of `analysis.py`. It contains a link to the `procDict.json` such that the samples can be properly normalised by getting centrally produced cross sections. (this might be removed later to include everything in the yaml, closer to the sample). It also contains the list of processes (matching the standard names), the number of CPUs, the cut list, and the variables (that will be both written in a `TTree` and in the form of `TH1` properly normalised to an integrated luminosity of $1pb^{-1}$.

- The analysis_final.py replace finalSel.py
- Configuration is now done in analysis_final.py that should follow some simple nomenclature
- To run over the samples defined inside analysis_final.py

```
fccanalysis final examples/FCCee/higgs/mH-recoil/mumu/analysis_final.py
```

What we have now - 3

3. `analysis_plots.py` : This analysis file is used to select the final selections from running `analysis_final.py` to plot. It usually contains information about how to merge processes, write some extra text, normalise to a given integrated luminosity etc... For the moment it is possible to only plot one signal at the time, but several backgrounds.

- The analysis_plots.py replace plots.py
- Configuration is now done in analysis_plots.py that should follow some simple nomenclature
- To run over the samples defined inside analysis_plots.py

```
fccanalysis plots examples/FCCee/higgs/mH-recoil/mumu/analysis_plots.py
```

What we have now - summary

- Developments done in the python wrapper allows a more intuitive and understandable way of running analyses
    - 1 executable: fccanalysis
    - 3 running modes: run, final, plots
- Batch support running at CERN on HTCondor, should be easy to run elsewhere

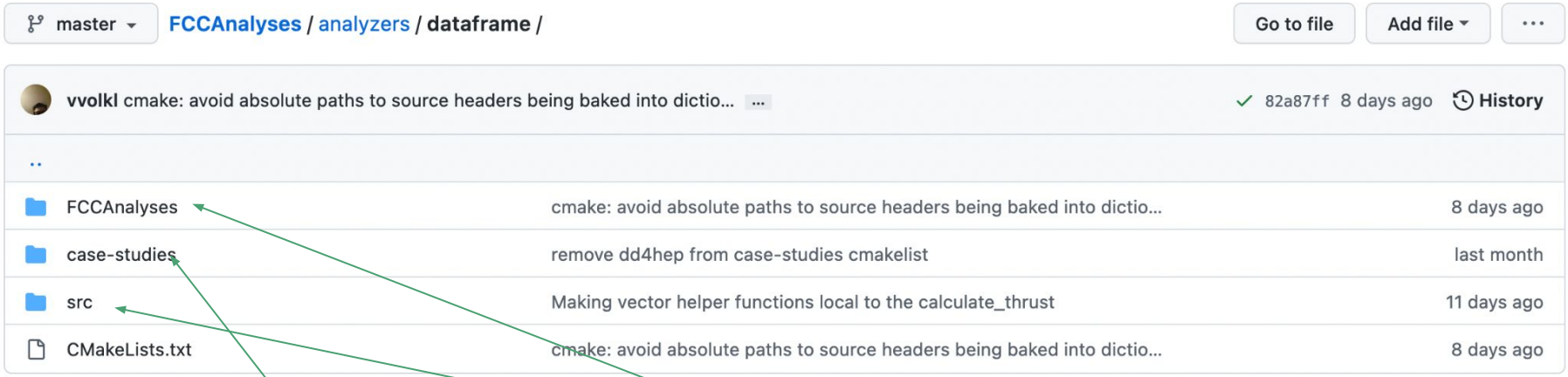
```
fccanalysis run examples/FCCee/higgs/mH-recoil/mumu/analysis_stage1.py
```

```
fccanalysis final examples/FCCee/higgs/mH-recoil/mumu/analysis_final.py
```

```
fccanalysis plots examples/FCCee/higgs/mH-recoil/mumu/analysis_plots.py
```

Other stages as well

FCCAnalyses organisation



Common dictionnary of analysers, living in namespaces FCCAnalyses::<analysersname>

Example to add case study code to simplify the workflow (still under tests/evaluation)

FCCAnalyses organisation

List of python configuration files that can be used to fully reproduce the inputs needed by analyses

Planned developments

On the FCCAnalyses side

- Add functionalities for batch job validation and re-submission
- RDataFrame allows to connect to spark and dask, need to support this in addition of standard batch submission
- Further re-organisation of the analysers and code optimisations
- Continue to add test and benchmarks
- Continue to write extensive documentation and take user feedback into account
- Continue to work on the vertexing with ACTS, custom LCFiplus custom implementation, and Franco’s fitter
- Consider producing a central “derivation” to allow users to run over a “simplified” format

Outlook

- FCCAnalyses is the baseline framework for FCC physics studies
  - It has already been used for publication quality results
  - Need user feedback to improve, please don’t be shy!

- Other planned developments on the RDF side include:
  - built-in support for handling systematic variations, with a clever "only re-run what's strictly needed" approach
  - seamless switch between TTree and RNTuple inputs
  - deep learning model inference as part of the multi-thread RDF event loop with SOFIE