FCCAnalyses ?

Cheat sheet

Juraj Smieško, CERN

About the framework

FCCAnalyses is a framework which builds on top of the new powerful event processing abstraction developed by ROOT, RDataFrame. It adds other necessary components needed for the analysis framework. Most notably, the management of the input samples and a standard library of analyzer functions.

In order to use FCCAnalyses you need to write one or more analysis scripts, where you design your analysis in the form of a graph consisting of analyzer functions which define variables you are interested in.

Key4hep

Key4hep stack comes in two flavors "Release" and "Nightly". For the analysis it is recommend to use "Release" as it is retained indefinitely, "Nightly" releases are deleted after few months.

FCCAnalyses is available in the Key4hep stack by default. To see all available sub-commands you can run:
`fccanalysis -h`

To see what are the available Key4hep stack versions:
`source /cvmfs/sw.hsf.org/key4hep/setup.sh -r`

Setup specific version of the stack:
`source /cvmfs/sw.hsf.org/key4hep/setup.sh -r 2024-04-12`

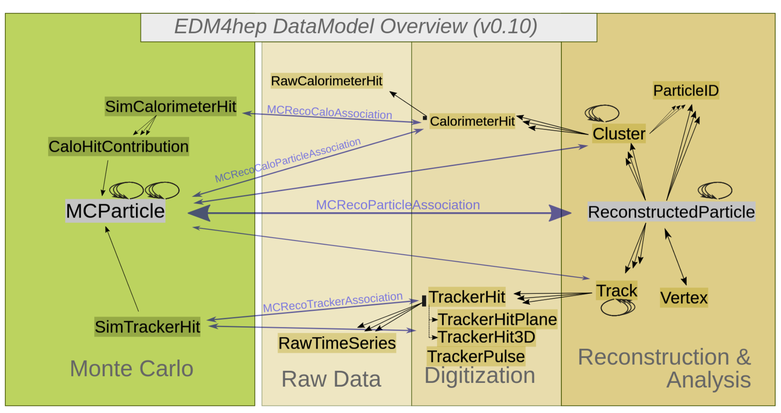
Pin your analysis to a specific version of the Key4hep stack:
`fccanalysis pin`

See what is the stack version you pin your analysis to:
`fccanalysis pin -s`

To clear your stack pin do:
`fccanalysis pin -c`

EDM4hep

In order to be able to properly communicate between various components of the Key4hep stack one needs a common description format of the event. EDM4hep datamodel defines optimized set of collections to describe almost any data in the event. The FCCAnalyses framework expects input in the EDM4hep format saved in ROOT file(s).



To list collections contained within a particular file:
`podio-dump file.edm4hep.root`

Dumping all data contained in a particular event:
`podio-dump -d -e event_number file.edm4hep.root`

Input samples

FCCAnalyses can ingest local samples as well as centrally produced pre-generated samples.
In order to see which version of the Key4hep stack was used to produce Full Sim file one can do:
`podio-dump -c runs -d file_edm4hep.root`

To specify locally produced sample to run on use:
`fccanalysis run -files-list file1.edm4hep.root`

All centrally produced pre-generated physics samples are listed at:
`https://fcc-physics-events.cern.ch`

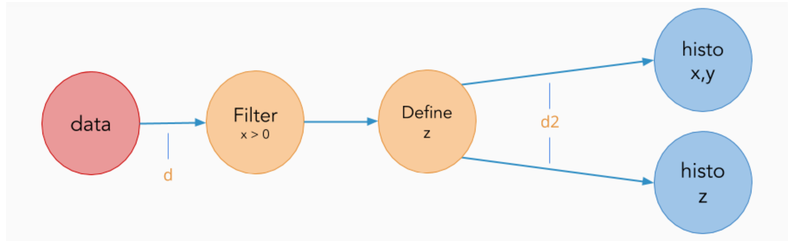
What are the available campaigns?
Delphes (Fast Sim) samples are available in `spring2021` and `winter2023` samples.

Note: EDM4hep 1.0 coming soon — new campaign will be generated and will include also Full Sim samples.

Which Key4hep stack was used for the production of the particular centrally produced sample?
The stack used is shown at top of the page which lists the sample information.

ROOT RDataFrame

In ROOT RDataFrame you are designing your analysis in a form of a graph. This computational graph is not executed right away as you are defining it, rather it is executed only when the result is actually needed.




```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z","x*x + y*y");
// make histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D({"hxy","hxy",16,-1,1,64,-1,1},"x","y");
```

In order to see more information from the FCCAnalyses framework which also includes ROOT RDataFrame information you can use -v or -vv:
`fccanalysis -v run analysis_script.py`

Most verbose output you can obtain includes output of the JIT compiled code:
`fccanalysis -vvv run analysis_script.py`

When you are designing your analyzer function you can use few logging macros to output some information:

```
rdfInfo « "Info message";
rdfDebug « "Debug info";
rdfVerbose « "Verbose information";
```

Running the analysis

What are the different modes FCCAnalyses can run in?

- **Stages**
  One can split the analysis into several preparatory stages *stage1, stage2, ...*, then have final selection in the *final* stage and finally generate plots in the *plots* stage.
- **Histmaker**
  This mode combines all work usually done in the *stages* and *final* into one script, after which one can generate plots.
- **NTupleizer**
  Enables generation of flat NTules with the ability to access detector geometry (specialized form of the standalone mode).
- **Standalone Python/C++**
  One can benefit from the use of the standard library of analyzer functions, but needs to manage the samples manually.

FCCAnalyses supports running on your local machine, but also at the CERN's HTCondor. To run on HTCondor you can set the appropriate attribute in your analysis script:
`runBatch = True`

Building

Note: Before unnecessarily building FCCAnalyses, please try to use the version of the FCCAnalyses distributed in the Key4hep stack.

To build FCCAnalyses one can use built-in sub-command:
`fccanalysis build -j n_threads`

To rebuild whole FCCAnalyses from scratch, run:
`fccanalysis build -c`

In case the compilation fails and `fccanalysis` command is not available to you, try recovering with:
`hash -r`

If custom version of the FCCAnalyses is required one can use standard CMake build procedure:

```
git clone git@github.com:HEP-FCC/FCCAnalyses.git
cd FCCAnalyses
source setup.sh
mkdir build install
cd build
cmake -DCMAKE_INSTALL_PREFIX=install ..
make install
cd ..
fccanalysis run analysis_script.py
```

Analyzer functions

In order to be able to define new variables in the dataframe FCCAnalyses comes with a standard set of analyzer functions. However, there is still many analyzer functions missing. There are several options, how to add yours.

Simple analyzers can be defined right in the dataframe `.Define()` statement:

```
.Define("first_electron_pt",
        "all_electrons_pt[0]")
```

More complex ones can be defined either in the analysis script itsef:

```
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;
  }
""")
```

or in the separate C++ header file:

```
ROOT.gInterpreter.Declare(
  '#include "my_header.hxx"'
)
```

Libraries integrated into FCCAnalyses, which can be used in your analyzer(s):

ROOT	ONNX	DD4hep
ACTS	FastJet	Delphes

Contributing

Any contribution to the framework is warmly welcome. In any stage of the development of your analyzer function you can contact us. Here are few recommendations when designing your analyzer function(s). As it will be used also by other fellow analyzers we would like to ensure certain level of quality.

- With your analyzer function design also few tests for it, so its correctness can be guaranteed over time.
- Write few lines of documentation in the form of a comment above your function.
- Format your function with the help of `clang-format`. This helps with readability of the function and keeps style consistent across all analyzer functions.
- Try to design your function in a way that it is composable with other analyzer functions.
- If your analyzer function grows above ~ 30 lines, try to slit it up into smaller functions.
- Use logging macros to inform users about what is happening inside your analyzer function.

Documentation

Central hub for the FCCAnalyses documentation materials is:
`https://hep-fcc.github.io/FCCAnalyses/`

Several FCCAnalyses related tutorials can be found at:
`https://hep-fcc.github.io/fcc-tutorials/`

The reference documentation itself is hosted at:
`https://hep-fcc.github.io/FCCAnalyses/doc/latest/`

Finally, the manual pages can be invoked with:
`man fccanalysis` or `man fccanalysis-subcommand` .

Contact

FCCAnalyses section in FCCSW forum:
`https://fccsw-forum.web.cern.ch/c/fccanalysis/`

FCCAnalyses Github repository:
`https://github.com/HEP-FCC/FCCAnalyses/`

FCC-PED SW Analysis mailing list:
`FCC-PED-SoftwareAndComputing-Analysis@cern.ch`