Experience with FCC software

Scott Snyder

Brookhaven National Laboratory, Upton, NY, USA

June, 2024
2024 FCC Week

Introduction

From Brieuc:

> *The idea is to give an overview of the FCC software, highlight what you found nice and less nice, what and how things could be improved, etc.*

Largely what I've tried to do here.
Focus more on the experience of using/developing the software rather than on what it actually does.
From my perspective, of course — your mileage may vary!

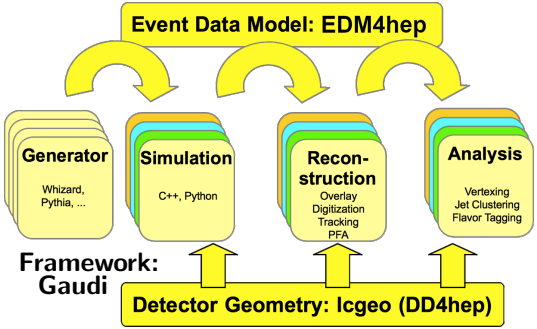
Key4hep

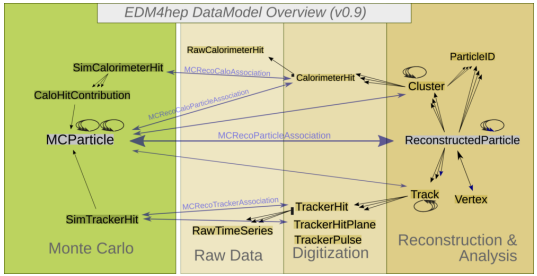
Collection of software supporting HEP experiments, (mostly) targeting studies of future colliders: FCC, ILC, CEPC, etc.

Relies on Gaudi (framework), dd4hep (detector description), EDM4hep/podio (event data model / event persistency).

Provides common components and data definitions that can be used across experiments — allows sharing effort.

Not really an integrated system like CMSSW/Athena. More like a Linux distribution, aggregating independent packages with consistent dependencies and a common toolchain.





EDM4hep / Podio

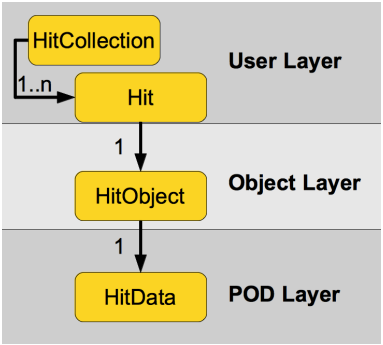
Common 'language' allowing different software components to interoperate.

Allows sharing code across experiments.

C++ code generated from data description language:

```
edm4hep::Cluster:
  Description: "Calorimeter Hit Cluster"
  Author: "F.Gaede, DESY"
  Members:
    - int32_t               type            //flagword ...
    - float                 energy          //energy ...
    - float                 energyError     //error ...
    - edm4hep::Vector3f     position        //position ...
    - std::array<float,6>   positionError   //covariance ...
    ...
```

Layered design favors composition over inheritance.



Default backend is ROOT: effectively a flat tuple. Files can be read without EDM library.

Basic setup

On EL9 (+EL7) machine with cvmfs, set up latest release with:

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

Set up a specific release with:

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh -r 2024-04-12
```

Set up latest nightly with

```
source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
```

Releases are built using `spack`.
Find logs and CI results from nightlies (good luck finding this in the documentation!):

```
https://gitlab.cern.ch/key4hep/k4-deploy/-/pipelines
```

Validation results:

```
https://key4hep-validation.web.cern.ch/
```

Example jobs

Setup (tested on EL9; EL7 shows crashes in Cling):

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh  -r 2024-04-12
```

Run Whizard:

```
git clone -b winter2023 https://github.com/HEP-FCC/FCC-config
# Disable Pythia vertex smearing to avoid crash in ddsim
sed -e 's/MSTP(151)=1;//' FCC-config/FCCee/Generator/Whizard/v3.0.3/\
wzp6_ee_eeH_HZZ_ecm240.sin > heezz.sin
whizard -e 'n_events=100' -e '$sample="heezz"' heezz.sin
```

Run fast simulation/reco:

```
DelphesSTDHEP_EDM4HEP  $DELPHES_DIR/cards/delphes_card_CLD.tcl \
 FCC-config/FCCee/Delphes/edm4hep_IDEA.tcl  heezz-delphes.root heezz.stdhep
```

Example jobs

Full simulation (shown for CLD):

```
git clone https://github.com/key4hep/CLDConfig
# HEAD not compatible with latest release.
(cd CLDConfig; git checkout 56e68fb4)
ddsim --compactFile $LCGEO/FCCee/CLD/compact/CLD_o2_v05/CLD_o2_v05.xml \
 --steeringFile CLDConfig/CLDConfig/cld_steer.py --numberOfEvents -1   \
 --inputFiles heezz.stdhep --outputFile=heezz.sim.edm4hep.root
```

Reconstruction (for CLD):

```
ln -sf CLDConfig/CLDConfig/PandoraSettingsCLD .
# Have to specify number of events even if processing the entire file.
k4run --inputFiles=heezz.sim.edm4hep.root --outputBasename=heezz.rec \
 --num-events=100 CLDConfig/CLDConfig/CLDReconstruction.py
```

Analysis

Not really the main concern here — but a few comments nontheless.

`fccanalysis` script available to ease making derived tuples from edm4hep files.

```
class RDFanalysis:
  def analysers (df):
    return (
      df
      .Define('pts',
              'ReconstructedParticle::get_pt (PandoraPFOs)')
    )
  def output():
    return ['pts']
```

Can be run with

```
fccanalysis run ana.py --output=ana.root \
     --files-list=heezz.rec_edm4hep.root
```

Histograms are meant to be made in a separate pass. Makes sense for making final plots for talks, etc. But it's convenient to produce rough plots at the same time as the tuple for debugging. It's possible to do this with `fccanalysis` but not obvious.

Running the same script on Delphes output fails immediately. Even though both use the same EDM4hep types, the container names are different — even before considerations of actual differences between fast/full sim.

Available tools are rather low-level and not comprehensive.

Debugging

Can have `k4run` start a debugger by adding `--gdb` to the command line:

```
k4run --inputFiles=heezz.sim.edm4hep.root --outputBasename=heezz.rec \
 --num-events=100 --gdb CLDConfig/CLDConfig/CLDReconstruction.py
```

Debugger will start before most dynamic libraries are loaded, so will probably have to continue until the library you want is available:

```
(gdb) catch load DDMarlinPandora
Catchpoint 1 (load)
(gdb) c
Continuing.
...
Catchpoint 1
  Inferior loaded .../libDDMarlinPandora.so.0.12.1
(gdb) break DDCaloDigi::processEvent
Breakpoint 2 at 0x7f340e7b1aa0
(gdb) c
Continuing.
...
Breakpoint 2, 0x00007f340e7b1aa0 in DDCaloDigi::processEvent
  (EVENT::LCEvent*)    ()
```

But you'll find:

- Releases have no debug symbols available.
- Nightlies have debug symbols, but no un-optimized builds are available.
- In any case, source is not in cvmfs.

So you'll probably want to rebuild packages yourself.

Rebuilding locally

First, find the proper repository. Some are under `https://github.com/key4hep`, some under `https://github.com/iLCSoft`, some under `https://github.com/AIDASoft`, some elsewhere. In case of doubt, clone `https://github.com/key4hep/key4hep-spack` and look under `packages`.

```
git clone https://github.com/iLCSoft/DDMarlinPandora
```

Can find the version used by looking in the release. Note that the package you're interested in may actually be taken from a release earlier than you've set up. Check your environment in case of doubt.

```
$ ls /cvmfs/sw.hsf.org/key4hep/releases/2024-03-10/\
   x86_64-almalinux9-gcc11.3.1-opt/ddmarlinpandora
0.12.01-p742e5/
$ (cd DDMarlinPandora; git checkout v00-12-01)
```

Rebuilding locally 2

Build package:

```
mkdir -p build/DDMarlinPandora inst
cd build/DDMarlinPandora
cmake -DCMAKE_BUILD_TYPE=Debug -DCMAKE_INSTALL_PREFIX=../../inst \
  ../../DDMarlinPandora
make -j6 install
cd ../..
```

Update environment:

```
export LD_LIBRARY_PATH=`pwd`/inst/lib:`pwd`/inst/lib64:$LD_LIBRARY_PATH
export PYTHONPATH=`pwd`/inst/python:$PYTHONPATH
export ROOT_INCLUDE_PATH=`pwd`/inst/include:$ROOT_INCLUDE_PATH
export CMAKE_PREFIX_PATH=`pwd`/inst:$CMAKE_PREFIX_PATH
export PATH=`pwd`/inst/bin:$PATH
```

If your library is listed in `$MARLIN_DLL`, *remove* the old entry and add the new one.

Some packages may require additional settings.

Rebuilding locally 3

Can build multiple packages as well.
If you've set `$CMAKE_PREFIX_PATH` properly, then they should be able to find each other.
But need to build by hand in the proper dependency order.

Should be possible in principle to use `spack` to manage builds. Some (complicated) instructions in the key4hep documentation (but I haven't tried it):
`https://key4hep.github.io/key4hep-doc/index.html`

Some packages may have additional quirks. For example, `ConformalTracking` overrides settings from `$CMAKE_BUILD_TYPE` to force compiling with optimization. Need to edit the cmake file if you want a non-optimized debug build.
Some packages require additional `cmake` definitions — look in `key4hep-spack`

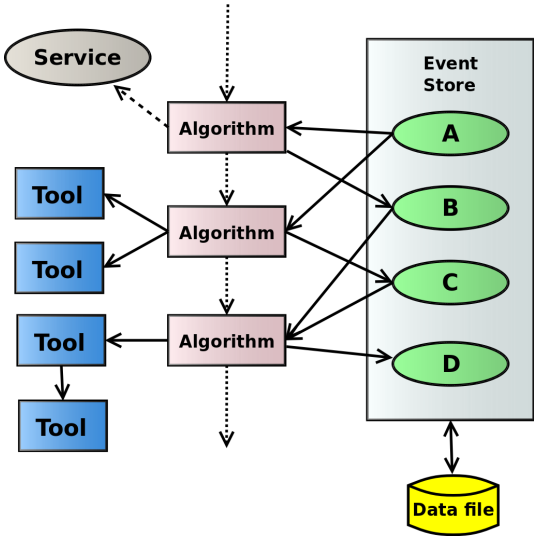
Gaudi Algorithms

Gaudi Algorithms read and write data from the event store.

Ideally have no event/conditions data state and do not modify their inputs.

Can access event data using the `DataHandle` class from k4FWCore — but that has to be a non-`const` member of the Algorithm class.

Alternative is to use the Gaudi functional interface.



Gaudi Functional algorithms

```
#include "Gaudi/Functional/Transformer.h"
#include "GaudiAlg/Transformer.h"
#include "k4FWCore/BaseClass.h"
#include "edm4hep/ClusterCollection.h"
#include "podio/UserDataCollection.h"

class TestAlg
  : public Gaudi::Functional::Transformer<podio::UserDataCollection<double> (const edm4hep::ClusterCollection&), BaseClass_t>
{
public:
  TestAlg (const std::string& name, ISvcLocator* svcloc)
    : Transformer (name, svcloc, KeyValue("input", "PandoraClusters"), KeyValue("energies", "energies2")) {}

  podio::UserDataCollection<double>
  operator() (const edm4hep::ClusterCollection& clusts) const override {
    podio::UserDataCollection<double> energies;
    for (const auto& cluster : clusts) {
      energies.push_back (cluster.getEnergy() * m_scale);
    }
    return energies;
  }
private:
  Gaudi::Property<float> m_scale { this, "scale", 1.1, "Scale factor" };
};
DECLARE_COMPONENT(TestAlg)
```

Nice for small algorithms — may get awkward for more complicated ones.

Marlin and LCIO

Although key4hep uses Gaudi/EDM4hep, much of the actual code was written for a different framework + EDM:
Marlin / LCIO.

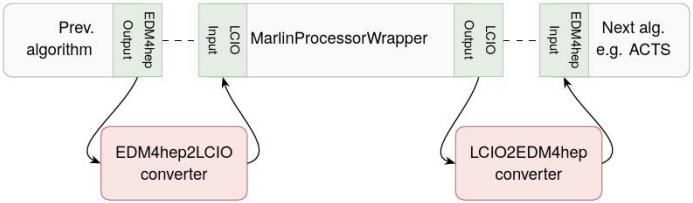
Marlin Algorithms (Processors) are wrapped with a generic Gaudi Algorithm that also converts data between EDM4hep and LCIO.

Configuration is more awkward:

```
MyStatusmonitor = MarlinProcessorWrapper("MyStatusmonitor")
MyStatusmonitor.OutputLevel = WARNING
MyStatusmonitor.ProcessorType = "Statusmonitor"
MyStatusmonitor.Parameters = { "HowOften": ["100"] }
```

versus

```
MyStatusmonitor = Statusmonitor("MyStatusmonitor",
  OutputLevel = WARNING, HowOften=100)
```



Multiple repositories

key4hep code is in many disparate repositories, built with `spack`. Leads to some issues:

- Can sometimes be difficult to even find the proper repository to look at.
- PRs / issues spread over many places.
- Not much support for coherently building multiple packages.

`spack` might help, but doesn't really seem like the proper tool for development. FCCSW documentation references top-level make file, but this appears to be obsolete.

- Incompatible interface changes are hard.
- Pull requests often languish for a long time, especially in older packages.
- CI across multiple packages?

Consider writing additional metadata in built packages, including repo/tag.

Consider copying source code to released packages.

Provide top-level cmake code for building multiple packages.

Some sort of librarian role who can make changes across all packages in the stack?

Summary

- FCC software works well for its intended purpose — result of a huge amount of good work.
  - Evaluate detector concepts without having to reinvent SW infrastructure.
  - Common EDM across experiments instrumental in reducing overall effort.
  - But numerous obstacles for newcomers.
  - And long history has led to the technical debt of wrappers and conversions.
- `spack` used to automate release builds.
  - But not much assistance provided for development: debugging and local builds.
  - Can get away with it as long as people working on the SW are mostly motivated experts.
  - More attention to this should help to reduce the barriers for new people to contribute.
  - Build releases with debuggability in mind.
  - Having the project spread over many repositories increases friction.
  - Should try to increase responsiveness for pull requests.
  - Remove reliance on artifacts not part of the key4hep release.

Some open tasks

(Largely lifted from Brieuc's Annecy talk)

- Complete full simulation/reconstruction of IDEA/Allegro:
- Particle flow performance of CLD with/without ARC detector.
- Migrate Marlin/LCIO tools to Gaudi/EDM4hep.
- Technical maintenance of existing packages (k4...).
- Tau reconstruction.
- Central implementation of detector performance evaluation tools.
- Full-simulation physics analysis.

Contact SW group if you want to contribute!

Documentation/tutorials

`https://hep-fcc.github.io/FCCSW`
`https://hep-fcc.github.io/fcc-tutorials/main/index.html`
`https://key4hep.github.io/key4hep-doc/index.html`