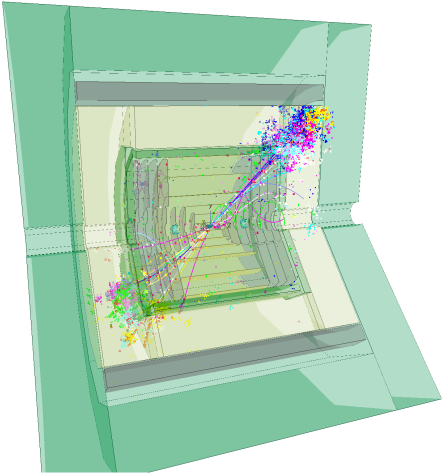
Key4hep: Status and Plans

André Sailer

CERN-EP-SFT

CEPC Workshop
November 10, 2021

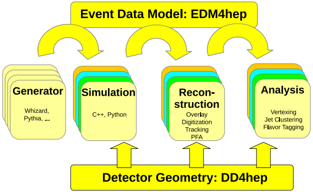
Table of Contents



Turnkey Software Stack

Create a software stack that connects and extends individual packages towards a complete data processing framework for detector studies with fast or full simulation, reconstruction, and for analysis

- Major ingredients: Event Data Model (EDM), Geometry Information, Processing Framework
- Sharing common components reduces overhead for all users
- Should be easy to use for librarians, developers, users
  - **easy to deploy, extend, set up**
- Full of functionality: plenty of examples for simulation and reconstruction of detectors
- Preserve and adapt existing functionality into the stack, e.g., from iLCSoft, FCCSW, CEPCSW



Framework: Gaudi

- Data processing frameworks are the skeleton on which HEP applications are built
- Gaudi was chosen as the framework, based on considerations for
  - portability to various computing resources, architectures and accelerators
  - support for task-oriented concurrency
  - adoption and developer community size; is used by LHCb, ATLAS
- Contribute developments were we see a need

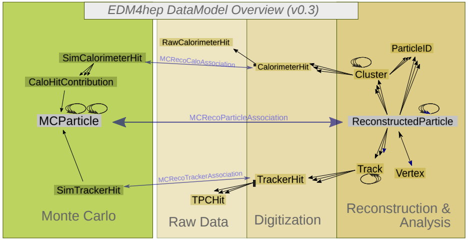
k4FWCore

- Basic IO functionality: podio data service

The Key4hep EDM: EDM4hep

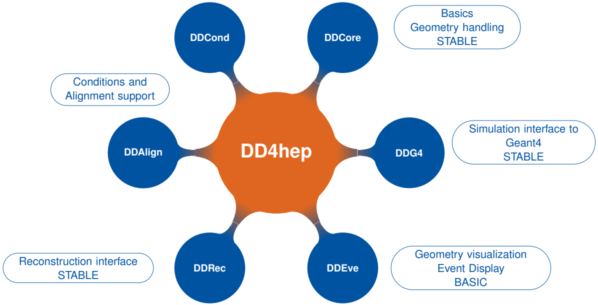
For a high degree of interoperability, EDM4hep provides a common event data model

- Using podio to manage the EDM (described by `yaml`) and easily change the persistency layer (ROOT, SIO, ...)
- EDM4hep data model based on LCIO and FCC-edm
- http://github.com/key4hep/edm4hep
- Recent developments for podio or EDM4hep
  - EDM4hep: additional types, associations
  - podio: event, run, collection metadata; UserDataCollection, Subset Collections
- A number of out standing issues still need to be resolved
  - "Wrapper" for using different hit types transparently
  - multi-threading
  - schema evolution



Geometry Information: DD4hep

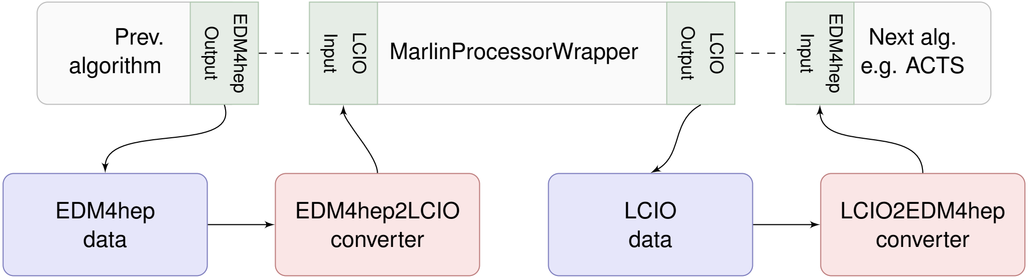
- **Complete Detector Description**
  - Providing geometry, materials, visualization, readout, alignment, calibration. . .
- **Single source of information → consistent description**
  - Use in simulation, reconstruction, analysis
- **Supports full experiment life cycle**
  - Detector concept development, detector optimization, construction, operation
  - Facile transition from one stage to the next



- **DD4hep already in use by ILC, CLIC, FCC, and many more**

Integrating iLCSoft Tools

- To adapt the full reconstruction suite from iLCSoft, which uses a different framework (Marlin) and event data model (LCIO) a wrapper was created. **key4hep/k4MarlinWrapper**
  - Tracking, Particle Flow Clustering, Flavour Tagging, Particle ID
- No change of user code required to run existing Marlin tools in Key4hep/Gaudi.
- Integrate battle proven tools into new reconstruction workflows
- The wrapper: converts LCIO to EDM4hep Data, calls the marlin processor, converts LCIO to EDM4hep (using **k4LCIOReader**)



MarlinWrapper Configuration

Example for configuring one of the Marlin processors for Gaudi

```
VXDBarrelDigitiser = MarlinProcessorWrapper("VXDBarrelDigitiser")
VXDBarrelDigitiser.OutputLevel = WARNING
VXDBarrelDigitiser.ProcessorType = "DDPlanarDigiProcessor"
VXDBarrelDigitiser.Parameters = {
                                 "IsStrip": ["false"],
                                 "ResolutionU": ["0.003", "0.003", "0.003", "0.003", "0.003", "0.003"],
                                 "ResolutionV": ["0.003", "0.003", "0.003", "0.003", "0.003", "0.003"],
                                 "SimTrackHitCollectionName": ["VertexBarrelCollection"],
                                 "SimTrkHitRelCollection": ["VXDTrackerHitRelations"],
                                 "SubDetectorName": ["Vertex"],
                                 "TrackerHitCollectionName": ["VXDTrackerHits"],
                                 }
```

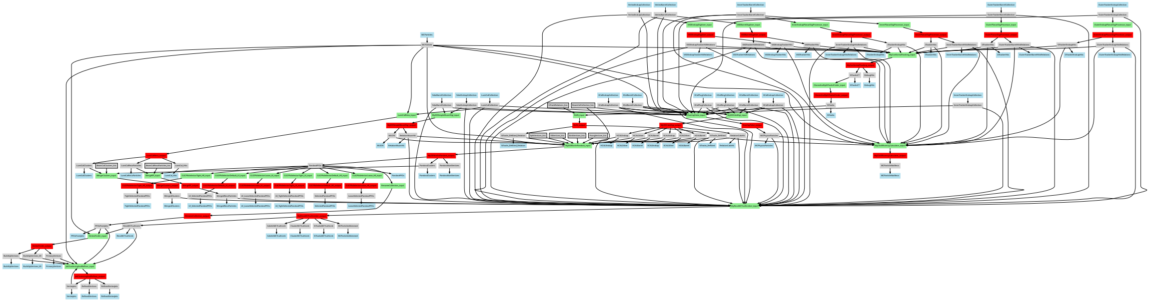
```
# EDM4hep to LCIO
edmConvTool = EDM4hep2LcioTool("VXDBarrelEDM4hep2lcio")
edmConvTool.Parameters = [
  "VertexBarrelCollection", "VertexBarrelCollection",
  ]
edmConvTool.OutputLevel = DEBUG
VXDBarrelDigitiser.EDM4hep2LcioTool = edmConvTool
```

```
# LCIO to EDM4hep
VXDBarrelDigitiserLCIOConv =
   Lcio2EDM4hepTool("VXDBarrelDigitiserLCIOConv")
VXDBarrelDigitiserLCIOConv.Parameters = [
  "VXDTrackerHits", "VXDTrackerHits",
  "VXDTrackerHitRelations", "VXDTrackerHitRelations"
  ]
VXDBarrelDigitiserLCIOConv.OutputLevel = DEBUG
VXDBarrelDigitiser.Lcio2EDM4hepTool =
    VXDBarrelDigitiserLCIOConv
```

CLIC Reconstruction

- Implemented full CLIC reconstruction chain, fixing subtle issues
- Validation pending

EDM4hep collections
Marlin processor inputs
LCIO collections
Marlin processor output

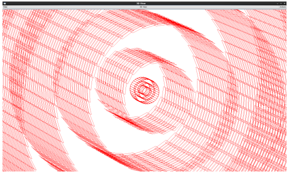


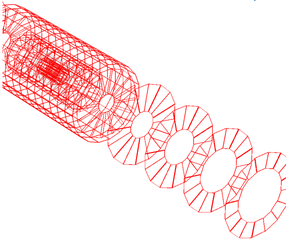
k4Clue

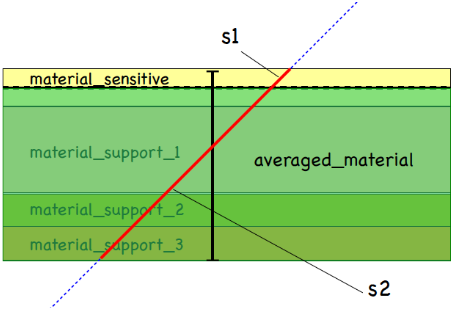
- Investigating use of the GPU friendly algorithm **CLUE** (CLUstering of Energy) as part of particle flow reconstruction
- CLUE Gaudi algorithm created: **k4Clue** and run as part of the CLIC reconstruction chain
- Validation and use of the clusters pending

k4ActsTracking

- Started work towards integration of the ACTS tracking toolkit with Key4hep:
  - Planning to create thin Gaudi Algorithm(s) converting necessary information for ACTS and tracks back to EDM4hep
  - See also Talk by Paul Gessinger later today
- Try to use information provided by `dd4hep::rec::Surface` class to ACTS
  - Surfaces can be added after the fact to the geometry instantiation
- Explicit addition of XML tags for ACTS will also be possible, but maybe requires dedicated drivers







Geant4 Simulations

- Currently have two somewhat independent simulation approaches: `ddsim` standalone and `k4SimGeant` framework integration.
- Both with useful functionality: sensitive detectors, MCHistory, particle guns
- How the simulation is configured shouldn't matter, but different approach to plugin instantiation currently prohibits sharing of the functionality

Ideally allow use of `DDG4` plugins with `k4SimGeant` and vice versa

```
from DDSim.DD4hepSimulation import DD4hepSimulation
from g4units import mm, GeV, MeV, m, deg
SIM = DD4hepSimulation()
SIM.compactFile = "CLIC_o3_v14.xml"
SIM.inputFile = "electrons.HEPEvt"
SIM.part.minimalKineticEnergy = 1.0*MeV
SIM.filter.filters = {'edep0':
    {'parameter': {'Cut': 0.0},
     'name': 'EnergyDepositMinimumCut/Cut0'},
  'edep1kev':
    {'parameter': {'Cut': 0.001},
     ' name': 'EnergyDepositMinimumCut'}}
SIM.filter.calo = "edep0"
SIM.filter.tracker = "edep1kev"
```

```
from Configurables import (SimG4Alg, SimG4SaveTrackerHits,
   SimG4PrimariesFromEdmTool,GeoSvc, SimG4Svc)
geoservice = GeoSvc("GeoSvc",
  detectors=['file:Detector/DetFCChhBaseline1/compact/FCChh_I
  'file:Detector/DetFCChhTrackerTkLayout/compact/Tracker.xml
geantservice = SimG4Svc("SimG4Svc",
  detector='SimG4DD4hepDetector',
  physicslist="SimG4FtfpBert", actions="SimG4FullSimActions"
savetrackertool = SimG4SaveTrackerHits("SimG4SaveTrackerHits'
  readoutNames = ["TrackerBarrelReadout",
    "TrackerEndcapReadout"])
savetrackertool.positionedTrackHits.Path = "positionedHits"
savetrackertool.trackHits.Path = "hits"
particle_converter = SimG4PrimariesFromEdmTool("EdmConverter'
particle_converter.genParticles.Path = "allGenParticles"
geantsim = SimG4Alg("SimG4Alg",
  outputs=["SimG4SaveTrackerHits/SimG4SaveTrackerHits"],
  eventProvider=particle_converter)
```

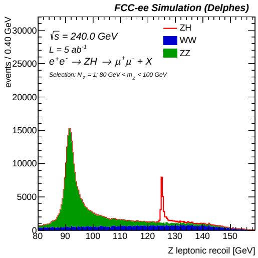
Using Delphes Fast Simulation

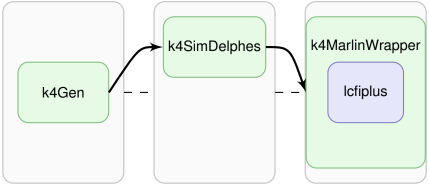
- **key4hep/k4SimDelphes** uses Delphes for fast simulation and creates output files in EDM4hep

```
from Configurables import k4SimDelphesAlg
delphesalg = k4SimDelphesAlg()
delphesalg.DelphesCard = "delphes_card_IDEA.tcl"
delphesalg.DelphesOutputSettings = "edm4hep_output_config.tcl"
delphesalg.GenParticles.Path = "GenParticles"
```

- Part of a coherent approach to generation and simulation in Key4hep
- No difference between output of Delphes and reconstruction after full simulation
- Plug and play higher level reconstruction, analysis for different ways of creating necessary samples

courtesy of C. Helsens





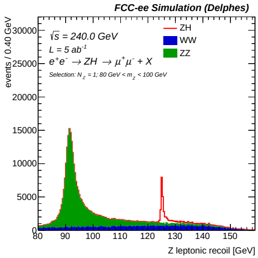
Using Delphes Fast Simulation

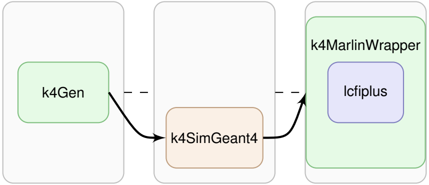
- **key4hep/k4SimDelphes** uses Delphes for fast simulation and creates output files in EDM4hep

```
from Configurables import k4SimDelphesAlg
delphesalg = k4SimDelphesAlg()
delphesalg.DelphesCard = "delphes_card_IDEA.tcl"
delphesalg.DelphesOutputSettings = "edm4hep_output_config.tcl"
delphesalg.GenParticles.Path = "GenParticles"
```

- Part of a coherent approach to generation and simulation in Key4hep
- No difference between output of Delphes and reconstruction after full simulation
- Plug and play higher level reconstruction, analysis for different ways of creating necessary samples

courtesy of C. Helsens





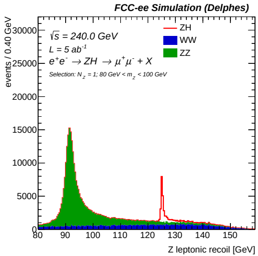
Using Delphes Fast Simulation

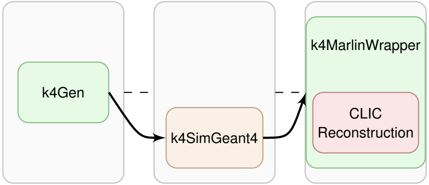
- **key4hep/k4SimDelphes** uses Delphes for fast simulation and creates output files in EDM4hep

```
from Configurables import k4SimDelphesAlg
delphesalg = k4SimDelphesAlg()
delphesalg.DelphesCard = "delphes_card_IDEA.tcl"
delphesalg.DelphesOutputSettings = "edm4hep_output_config.tcl"
delphesalg.GenParticles.Path = "GenParticles"
```

- Part of a coherent approach to generation and simulation in Key4hep
- No difference between output of Delphes and reconstruction after full simulation
- Plug and play higher level reconstruction, analysis for different ways of creating necessary samples

courtesy of C. Helsens





FCCSW Reorganization

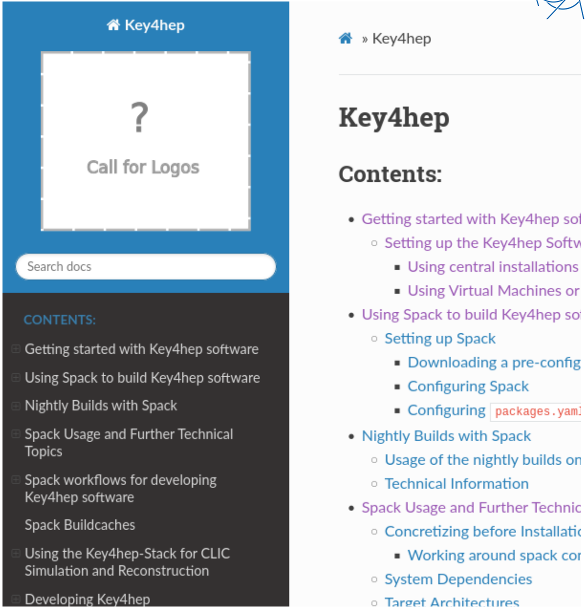
- FCCSW was based on Gaudi and PODIO (fcc-edm) from the start
  - Less effort to transition, switch to EDM4hep already finished
- hep-fcc/fccsw mono-repository was split up to allow use and re-use of common components in Key4hep
  - k4FWCore (Framework Core): PodioData services and event overlay
  - k4Gen: generator interfaces and particle guns
  - k4SimDelphes: for delphes fast simulation with EDM4hep output
  - k4RecCalorimeter: for calorimter reconstruction

Testing and Validation

- Basic tests are part of the GitHub infrastructure
- Physics validation for simulation and reconstruction is still missing, especially with automation

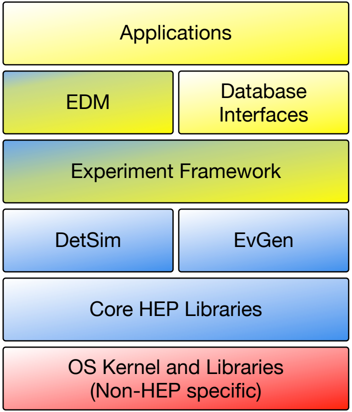
Documentation

- Main documentation page key4hep.github.io based on GitHub pages https://github.com/key4hep/key4hep-doc
- Test the examples in the documentation via `notedown`
- Doxygen, e.g., EDM4hep https://edm4hep.web.cern.ch/
- CLIC simulation and reconstruction example
  - Would be nice to add the CEPC workflows as well
- Restructuring of documentation in the works
  - Separate *User*, *Developer*, *Librarian* content



Packaging: Spack

- Need to build a large number of packages to run our applications
- Adopted **spack** as the package manager
- Go beyond sharing of *build results* to sharing of *build recipes*
  - Separate repository for Key4hep specific recipes
- Can build any and all pieces of the stack with minimum effort
  - `spack install key4hep-stack`
  - `spack dev-build conformaltracking@master`
- Used for nightly builds and releases of the stack
  - `source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh`



Conclusions

- The Key4hep software stack has made significant progress
- CEPC, CLIC, FCC, ILC components are being used from one stack
- Additional functionality (ACTS, CLUE) is being integrated
- Simulation functionality needs more convergence
- (Automatic) validation will become a higher priority

Thanks to Valentin Volkl, Placido Fernandez, Erica Brondolin for material.

Thank you for your attention