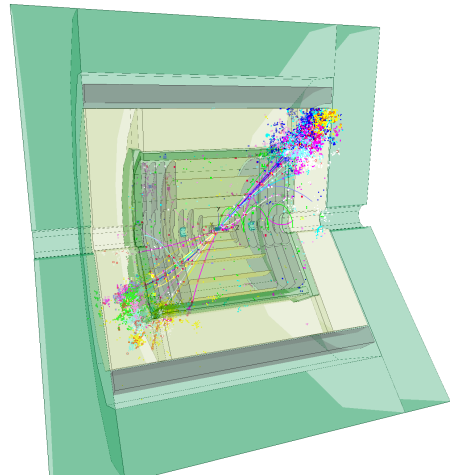
Key4hep: Turnkey Software Stack

André Sailer (CERN) for the Key4hep Team

Software and Computing Round Table
May 4, 2021

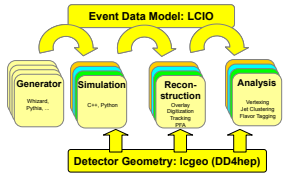
Table of Contents



The Vision

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
- Members of FCC, ILC, CEPC, SCT, CLIC agreed to work on a common software stack
- Preserve and adapt existing functionality into the stack, e.g., from iLCSoft or FCCSW



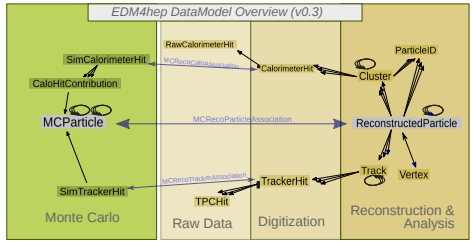
iLCSoft components, but general scheme of ingredients applies

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, . . .)
- http://github.com/key4hep/edm4hep
- EDM4hep data model based on LCIO and FCC-edm
- Key4hep triggered many developments for podio:
  - SIO backend
  - new templating
  - metadata
  - multi-threading investigations
  - schema evolution

EDM4hep to be used at any stage of Key4hep

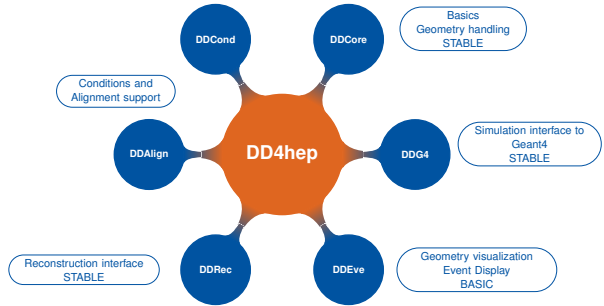


Data Processing Framework: Gaudi

- 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

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

Example for configuring one of the Marlin processors for Gaudi

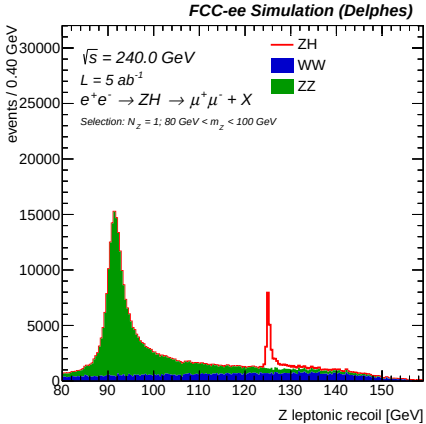
```
VXDBarrelDigitiser = MarlinProcessorWrapper("VXDBarrelDigitiser")
VXDBarrelDigitiser.OutputLevel = WARNING
VXDBarrelDigitiser.ProcessorType = "DDPlanarDigiProcessor"
VXDBarrelDigitiser.Parameters = [
                                "IsStrip", "false", END_TAG,
                                "ResolutionU",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

Packaging: Spack

- Adopted spack as the package manager
- Go beyond sharing of *build results* to sharing of *build recipes*
- Large community supporting recipes for many of the packages in the stack
- Eases work load librarians and developers
- Can build any and all pieces of the stack with minimum effort
- Separate repository for Key4hep specific recipes
- Used for nightly builds and releases of the stack

Fast Simulation with Delphes: k4SimDelphes

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

Full Simulation

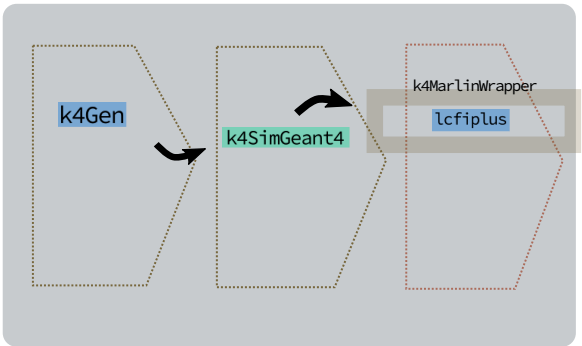
- Full (Geant4) Simulation to be integrated into the framework
- Also available the DD4hep `ddsim` python executable for standalone simulation, produces EDM4hep output

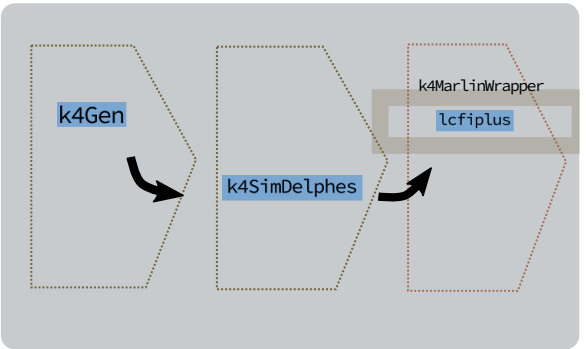
```
ddsim --compactFile mydetector.xml --outputFile simulatedEvents_edm4hep.root ...
```

- Hopefully two approaches for simulation configuration compatible, but more work needed on this front

Fast Simulation and High Level Reconstruction

- We can feed the EDM4hep output from Delphes to the High Level reconstruction tools
- Or run the full simulation and reconstruction instead of Delphes
- Once the framework integration of Delphes is complete with the flip of a switch





Full Simulation/Reconstruction Example

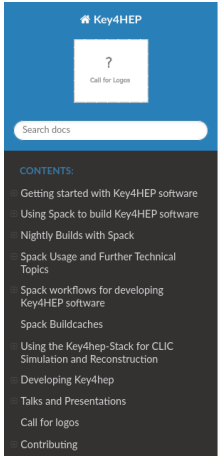
https://key4hep.github.io/key4hep-doc/examples/clic.html

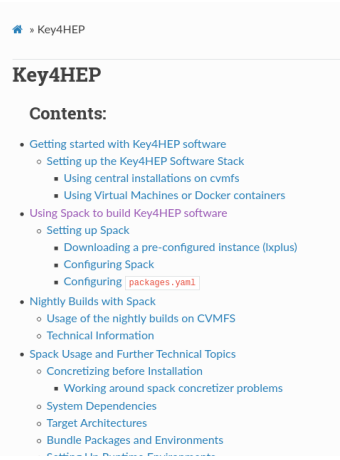
```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
git clone https://github.com/iLCSoft/CLICPerformance
ddsim --compactFile $LCGEO/CLIC/compact/CLIC_o3_v14/CLIC_o3_v14.xml \
      --outputFile ttbar.slcio \
      --steeringFile clic_steer.py \
      --inputFiles ../Tests/yyxyev_000.stdhep \
      --numberOfEvents 3
Marlin clicReconstruction.xml \
       --InitDD4hep.DD4hepXMLFile=$LCGEO/CLIC/compact/CLIC_o3_v14/CLIC_o3_v14.xml \
       --global.LCIOInputFiles=ttbar.slcio \
       --global.MaxRecordNumber=3
```

- A few more commands to convert steering file and run this via Gaudi
  `k4Run clicReconstruction.py`

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/





Self Documenting Programs

- Writing Documentation is hard, keeping it up-to-date even more
- Ideally have one place were code and documentation lives together so easy to remember to update
- iLCSoft: `Marlin -x` prints config file for all known *processors* including documentation (see below)
- DD4hep: `ddsim --dumpSteeringFile` creates config file with all options and explanations; `ddsim --help`
- `k4Run config.py --help` prints available parameters

```
<processor name="VXDBarrelDigitiser" type="DDPlanarDigiProcessor">
  <parameter name="SubDetectorName" type="string">Vertex </parameter>
  <!--PlanarDigiProcessor creates TrackerHits from SimTrackerHits, smearing them according to the input pa
  <!--whether hits are 1D strip hits-->
  <parameter name="IsStrip" type="bool">false </parameter>
  <!--resolution in direction of u-->
  <parameter name="ResolutionU" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003 </parameter>
  <!--resolution in direction of v-->
  <parameter name="ResolutionV" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003  </parameter>
  <!--Name of the Input SimTrackerHit collection-->
  <parameter name="SimTrackHitCollectionName" type="string" lcioInType="SimTrackerHit">VertexBarrelCollec
  <!-- ... -->
</processor>
```

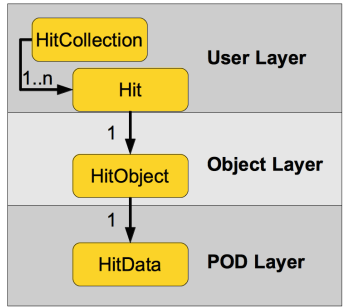
Summary

- The Key4hep Turnkey software stack aims to provide a complete framework for detector optimisation studies for future experiments
- Incorporate existing and new solutions
- Provide examples that can be adapted for other use cases.
- Interested parties are welcome to participate
  - Tuesday, 9:00 AM CET, weekly alternating EDM4hep/Key4hep https://indico.cern.ch/category/11461/
  - Contributors from China, Germany, Italy, CERN / ILC, CLIC, FCC, CEPC
  - If there is demand, we can find an occasional more US friendly slot

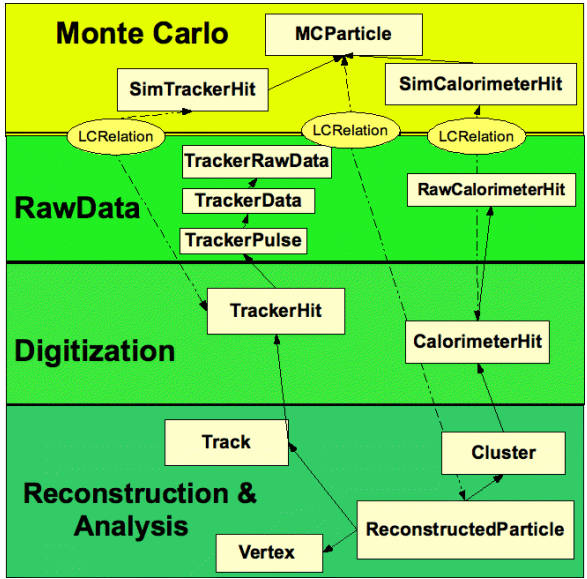
Backup Slides

podio Core Features

- Three layers:
  - user layer (API): collections of EDM object handles, *HitCollection*
  - object layer: transient objects (*HitObject*)
  - POD layer: persistent information
- Clear ownership: objects owned by *EventStore* are persisted, other objects ref-counted
- Python as first class citizen
- Different I/O implementations, but currently only ROOT



LCIO


Monte Carlo
MCParticle
SimTrackerHit
SimCalorimeterHit
LCRelation
LCRelation
LCRelation
TrackerRawData
TrackerData
TrackerPulse
RawCalorimeterHit
RawData
TrackerHit
CalorimeterHit
Digitization
Track
Cluster
Reconstruction &
Analysis
ReconstructedParticle
Vertex

Marlin & Gaudi

Apart from some naming conventions, very similar ideas in the two frameworks*

	Marlin	Gaudi
language	`c++`	`c++`
working unit	Processor	Algorithm
configuration language	XML	Python
set up function	`init`	`initialize`
working function	`processEvent`	`execute`
wrap up function	`end`	`finalize`
Transient data format	`LCIO`	anything

- To start using Gaudi: use a generic wrapper around the processors.
- https://github.com/key4hep/k4MarlinWrapper
- Read LCIO files and pass the `LCIO::Event` to our processors

*Of course subtle differences emerge

Wrapper Configuration

- Translate the `XML` to `python`, using a stand alone python script
- Pass arbitrary number, types, and names of parameters to the processor

Marlin/XML

```
<processor name="VXDBarrelDigitiser" type="DDPlanarDigiProcessor">
  <parameter name="SubDetectorName" type="string">Vertex </parameter>
  <parameter name="IsStrip" type="bool">false </parameter>
  <parameter name="ResolutionU" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003 </parameter>
  <parameter name="ResolutionV" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003  </parameter>
  <parameter name="SimTrackHitCollectionName" type="string" lcioInType="SimTrackerHit">VertexBarrelColl
  <parameter name="SimTrkHitRelCollection" type="string" lcioOutType="LCRelation">VXDTrackerHitRelation
  <parameter name="TrackerHitCollectionName" type="string" lcioOutType="TrackerHitPlane">VXDTrackerHits
  <parameter name="Verbosity" type="string">WARNING </parameter>
</processor>
```

Wrapper Configuration

- Translate the `XML` to `python`, using a stand alone python script
- Pass arbitrary number, types, and names of parameters to the processor

Gaudi/Python

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

Configuration: Control flow

- XML `execute` section translated to a python list

```xml
<execute>
  <processor name="MyAIDAProcessor"/>
  <processor name="EventNumber" />
  <processor name="InitDD4hep"/>
  <processor name="Config" />
  <!-- ... -->
</execute>
```

```python
algList = []
algList.append(lcioReader)
algList.append(MyAIDAProcessor)
algList.append(EventNumber)
algList.append(InitDD4hep)
algList.append(OverlayFalse)
algList.append(VXDBarrelDigitiser)
#...
```