Software Prototype based on SNiPER

Wenhao Huang[1], Xingtao Huang[1], Qiyun Li[1], Zuhao Li[2], Zheng Quan[2], Zhicheng Tang[2], Meng Wang[1], Ming Xu[2],Jumin Yuan[1]

[1] Shandong University
[2] IHEP

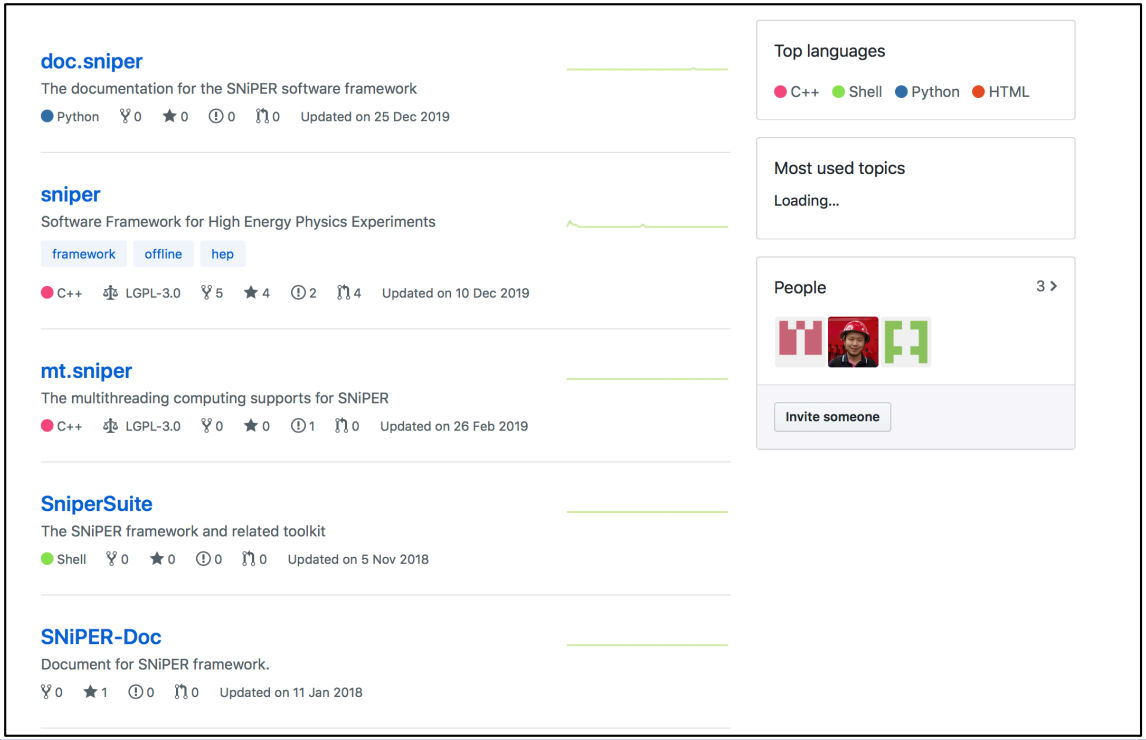
Mar. 17, 2020

Outline

- **Framework**
  - ⇨ Key Features of SNiPER
  - ⇨ Prototype architecture
- **Detector Simulation**
  - ⇨ DD4hep to describe Detector Geometry
- **Event Data Model**
  - ⇨ PODIO to define Event Data Model
- **Software Management**
  - ⇨ Computing environment and Tools
- **Summary**

SNiPER

- Introduction to SNiPER has been given at 8th HERD Workshop
  - ⇨ https://indico.ihep.ac.cn/event/10745/session/10/contribution/44
- More details can be found at the web
  - ⇨ https://github.com/SNiPER-Framework



Key Features of SNiPER (I)

- Modern Programming Language
  - ⇨Mainly developed with C++
  - ⇨Configured with python scripts, clear, readable and powerful

```
 1 #!/usr/bin/env python
 2
 3 import Sniper
 4
 5 task = Sniper.Task("task")
 6 task.setLogLevel(2)
 7
 8 import HelloWorld
 9 alg = task.createAlg("HelloAlg/dalg")
10 alg.property("someString").set("some value")
11
12 task.setEvtMax(5)
13 task.show()
14 task.run()
~
```

"import" for Dynamically loading package libs

Set new values to parameters

*Also possible to support python programming

Key Features of SNiPER (II)

- Highly modular
  - ⇨ Each module is functionally independent
  - ⇨ Main functions for data processing have been implemented in kernel modules
    - Data processing controlling, Event Data Management, Logging, user interfaces (Algorithm, Service, Tools)…
- Standard interfaces between different modules
  - ⇨ The interfaces have been very stable
    - Extensive used by the JUNO Experiment for M.C. Data production
    - Started raw data analysis by the LHAASO Experiment
  - ⇨ we only focus on event data model, detector simulation, calibration , reconstruction and analysis tools

Key Features of SNiPER (III)

- Separation between data and algorithm
  - ⇨ Less coupling between algorithms
  - ⇨ Development of new algorithms at the same time
- Data Store for event data management
  - ⇨ The data objects in the Data Store is open, currently implemented with
    - Data Object based on ROOT or defined by PODIO
  - ⇨ Provided handles to get/put event data from/to Data Store
- Support multithreading(not covered today)
  - ⇨ Underlying the intel TBB is deployed
  - ⇨ Multi-tasks naturally maps with multi-threads

Key Features of SNiPER (IV)

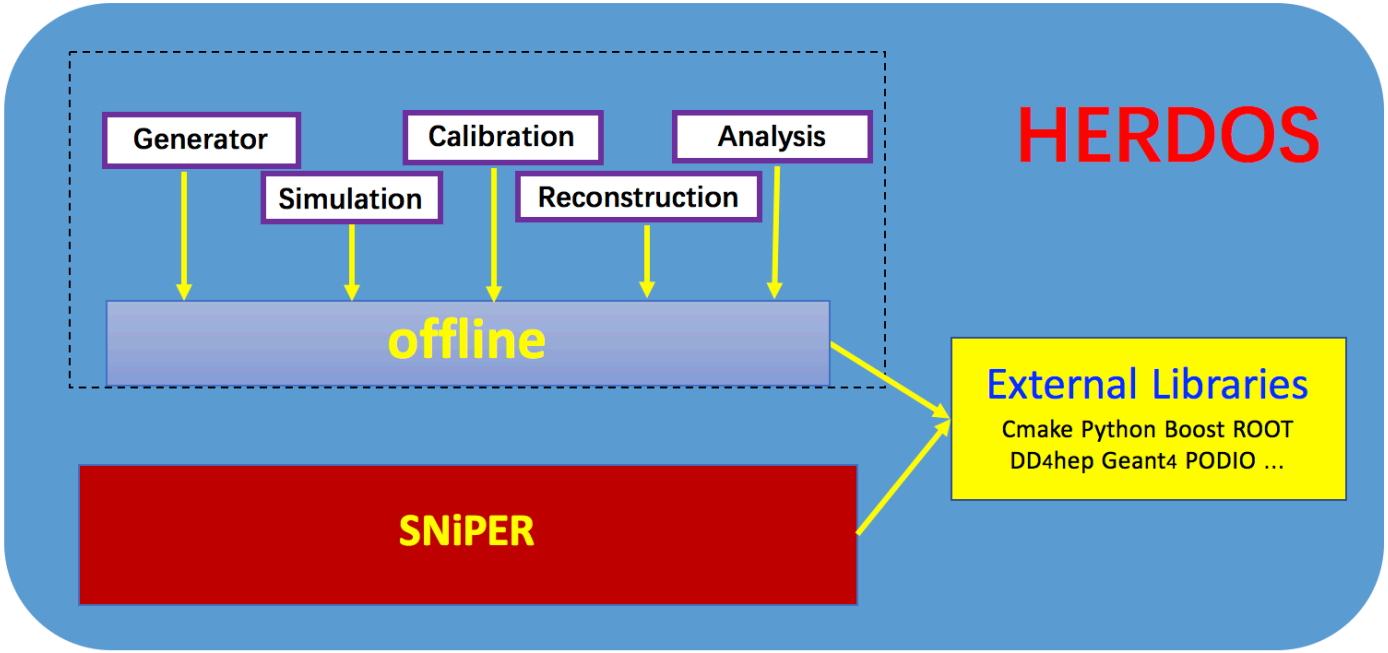
- It is lightweight, fast, flexible and easy to learn
  - ⇨See the examples
- It is being used by JUNO, LHAASO, STCF, nEXO
- There is a good team to maintain and optimize
  - ⇨SDU and IHEP

Progress since Xian Workshop

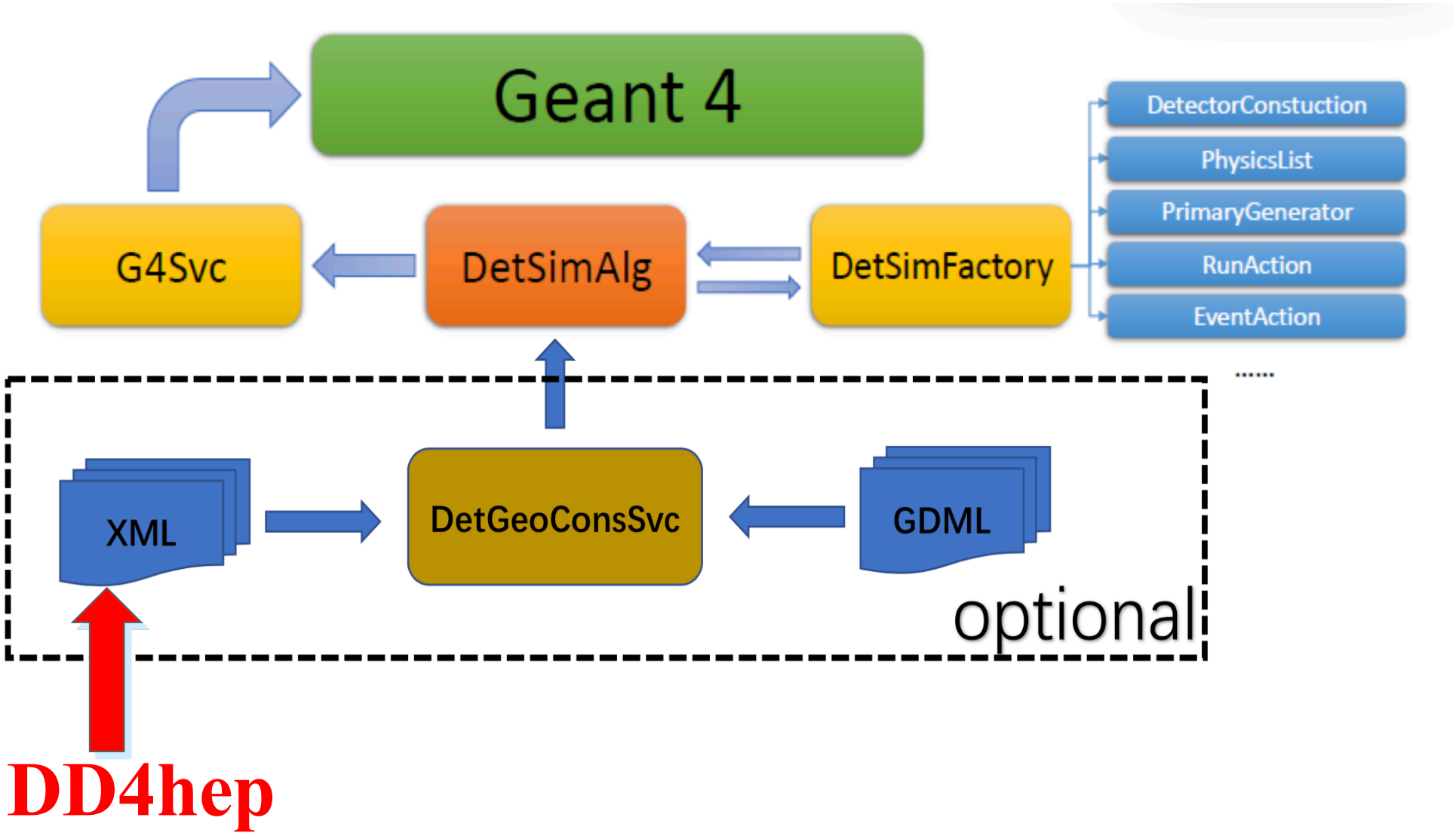
- Several Discussions inside the group since Xian workshop
  - ⇨ Chinese New Year Holiday
  - ⇨ Coronavirus
- Not too much progress
- Only setup a HERDOS (HERD Offline Software) prototype for testing
  - ⇨ Setup the environment for the HERD
  - ⇨ Extension to SNiPER with DD4hep and PODIO
  - ⇨ Use DD4hep to describe sub-detectors
  - ⇨ Use PODIO to define Data Model
  - ⇨ Add some examples to test DD4hep and PODIO

HERDOS Architecture

- **External Libraries**
  - ⇨ Frequently used third-party software and tools
- **SNiPER Framework**
  - ⇨ Data Processing Management, Event data Management, Common Services…
- **Offline**
  - ⇨ Specific to the HERD Experiment, including Extension to SNiPER , Generator, Simulation, Calibration , Reconstruction and Analysis



Overview of Detector Simulation System



Main Components of Detector Simulation

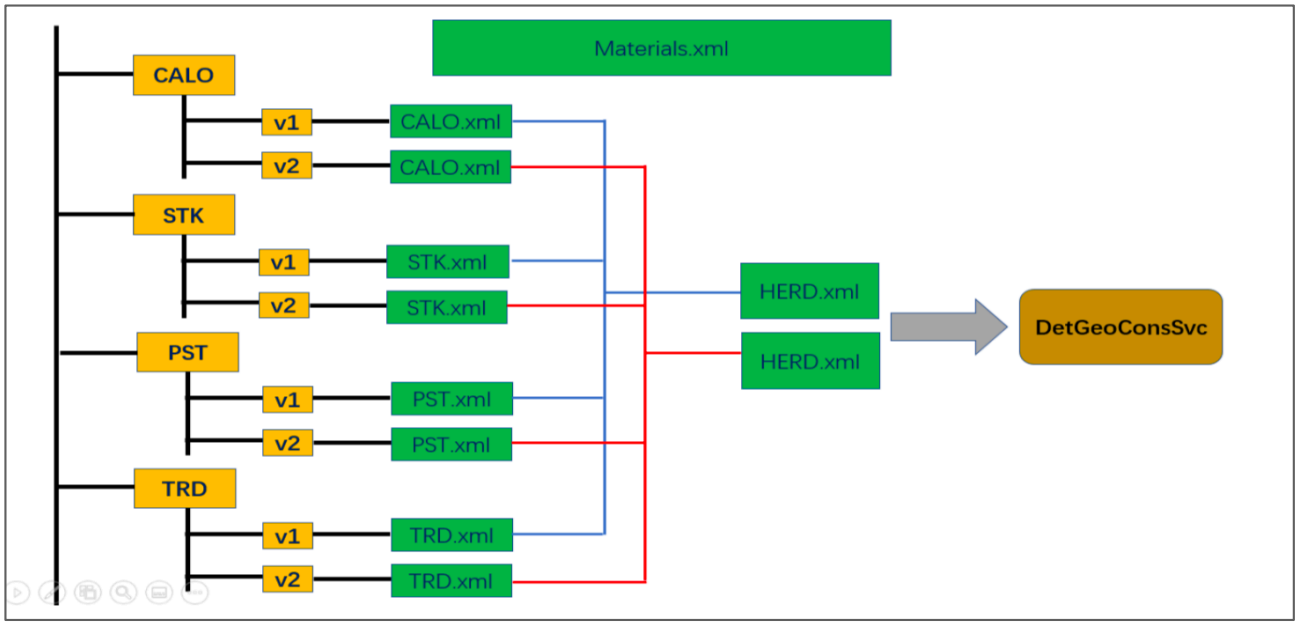
- DetSimAlg: the Algorithm for all detector simulation
- DetGeoConsSvc: the service to deliver xml/gdml geometry to Geant4
- G4Svc :A the service to launch Geant4
- DetSimFactory : the service to set up all the Geant4 actions

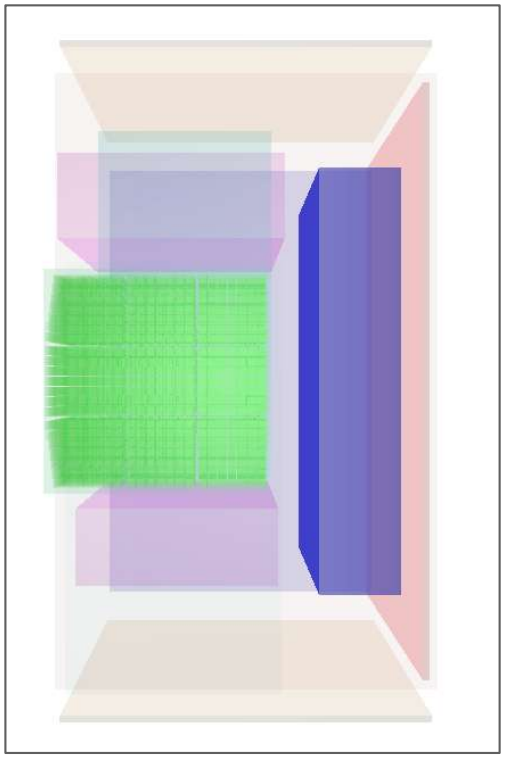
- G4VUserDetectorConstruction
- G4VUserPhysicsList
- G4VUserPrimaryGeneratorAction

- G4UserRunAction
- G4UserEventAction
- G4UserStackingAction
- G4UserTrackingAction
- G4UserStepingAction

Detector Description with DD4hep

- Each sub-detector is independent with others, different version in different path
- Flexible to build a full detector with different combinations of sub-detectors
- Common files for materials and elements





From zhicheng

```
[huangxt@lxslc605 compact]$ ls
calo_config.xml  elements.xml  materials.xml   psd_det.xml  tk_config.xml
calo_det.xml     master.xml    psd_config.xml  simple.xml   tk_det.xml
```

Event Data Model defined with PODIO

- Event Data Model is very importance for offline data proceeding
- PODIO is developed with future-oriented design
  - Automatic memory-management
  - Support concurrency access
  - Define event data in the yaml file
  - Allow to have 1-1, 1-N or N-M relation-ship
  - Currently support writing data into ROOT files, and HDF5 and other formats will supported
- PODIO has been integrated with SNiPER
  - Add PodioDataSvc to wrap PODIO inside SNiPER
  - Provide Handles to access EventStore of PODIO
  - Provide PODIO Input and Output System for reading and writing

Part Yaml file (offline/DataModel/EventDataModel/datalayout.yaml)

```
69 #------------- SimTrackerHit
70   HERDOS::SimTrackerHit:
71     Description: "Simulated tracker hit"
72     Author : "SDU"
73     Members:
74       - int cellID      //ID of the sensor that created this hit
75       - float EDep                        //energy deposited in the hit [Ge
76       - float pathLength                  //path length of the particle in
77       - float time                        //proper time of the hit in the l
78       - HERDOS::Vector3d position        //the hit position in [mm].
79       - HERDOS::Vector3f momentum        //the 3-momentum of the particle a
80     OneToOneRelations:
81       - HERDOS::MCParticle MCParticle //MCParticle that caused the hit.
82
```

Part C++ codes (offline/DataModel/EventDataModel/EventDataModel/SimTrackerHit.h)

```
31 class SimTrackerHit {
32
33 public:
34   /// default constructor
35   SimTrackerHit();
36   SimTrackerHit(int cellID,float EDep,float pathLength,
37                 float time,HERDOS::Vector3d position,HERDOS::Vector3f momentum);
38
39 public:
40
41   const int& getCellID() const;
42   const float& getEDep() const;
43
44 private:
45   SimTrackerHitObj* m_obj;
46
47 };
48
```

Example: Produce and Write Event Data

- Package : offline/Examples/EDMtests
  - ⇨ Produce new Event Data

```
29 class EDMProducer : public AlgBase
30 {
31     public :
32
33         EDMProducer(const std::string& name);
34         virtual ~EDMProducer();
35
36         virtual bool initialize();
37         virtual bool execute();
38         virtual bool finalize();
39
40     private :
41
42         int                                 m_iEvt;
43         std::string                         m_string;
44
45        DataHandle<HERDOS::MCParticleCollection>  *mcParticleCol;
46        DataHandle<HERDOS::SimTrackerHitCollection>  *simTrackerHitCol;
47        DataHandle<HERDOS::SimCalorimeterHitCollection>  *simCalorimeterHitCol;
48 };
```

```
36 }
37
38 bool EDMProducer::initialize()
39 {
40     // default Write mode
41     mcParticleCol          = new DataHandle<HERDOS::MCParticleCollection>
42                              ("mcParticleCol", this->getParent());
43     simTrackerHitCol       = new DataHandle<HERDOS::SimTrackerHitCollection>
44                              ("simTrackerHitCol", this->getParent());
45     simCalorimeterHitCol   = new DataHandle<HERDOS::SimCalorimeterHitCollection>
46                              ("simCalorimeterHitCol", this->getParent());
47
48     LogInfo << " initialized successfully" << std::endl;
49     return true;
50 }
51
52 bool EDMProducer::execute()
53 {
54     ++m_iEvt;
55     // MCParticle
56     auto mcps = mcParticleCol->getCollection();
57     auto p = mcps->create();
58     auto pprime = mcps->create();
59     p.setPdgID(m_iEvt+10000);
60     pprime.setPdgID(m_iEvt+1000);
61     p.addDaughter(pprime);
62
63     //SimTrackerHit
64     auto sths = simTrackerHitCol->getCollection();
65     auto sth = sths->create();
66     sth.setPathLength(m_iEvt+0.5);
67
68     //SimCalorimeterHit
69     auto schs = simCalorimeterHitCol->getCollection();
70     auto sch = schs->create();
71     sch.setEnergy(m_iEvt*0.4);
```

Python job configuration file: write.py

⇨ Write the produced Event Data into the Root File, output.root

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import Sniper

task = Sniper.Task("task")
task.setLogLevel(2)

import EDMtests
alg = task.createAlg("EDMProducer/dalg")

import PodioDataSvc
dsvc = task.createSvc("PodioDataSvc")

import PodioSvc
svc = task.createSvc("PodioOutputSvc/OutputSvc")
svc.property("OutputFile").set("output.root")
svc.property("OutputCollections").set( { "mcParticleCol", "simTrackerHitCol"})

task.setEvtMax(5)
task.show()
task.run()
~
```

```
[huangxt@lxslc606 share]$ python write.py
```

Example: Read and Analyze Event Data

- Package : offline/Examples/EDMtests
  - ⇨ Read Event Data from Root Files and do analysis
  - ⇨ Python job configuration file: read.py

```
#!/usr/bin/env python

import Sniper

task = Sniper.Task("task")
task.setLogLevel(2)

import EDMtests
alg = task.createAlg("EDMConsumer/dalg")

import PodioSvc
svc = task.createSvc("PodioInputSvc/InputSvc")
svc.property("InputFile").set("output.root")

import PodioDataSvc
pSvc = task.createSvc("PodioDataSvc")


task.setEvtMax(-1)
task.show()
task.run()
```

Example: Read and Analyze Event Data

- Algorithm: EDMConsumer
  - ⇨Read Event Data from Root Files and do analysis

```
14 class EDMConsumer : public AlgBase
15 {
16     public :
17
18         EDMConsumer(const std::string& name);
19         virtual ~EDMConsumer();
20
21         virtual bool initialize();
22         virtual bool execute();
23         virtual bool finalize();
24
25     private :
26
27         int      m_iEvt;
28         DataHandle<HERDOS::MCParticleCollection>  *mcParticleCol;
29
30
31 };
```

```
25 bool EDMConsumer::initialize()
26 {
27     mcParticleCol = new DataHandle<HERDOS::MCParticleCollection>("mcParticleCol",
28                                                                  this->getParent(), Read);
29
30
31     LogInfo << " initialized successfully" << std::endl;
32     return true;
33 }
34
35 bool EDMConsumer::execute()
36 {
37     ++m_iEvt;
38     auto mcps = mcParticleCol->get();
39     if(mcps->isValid()){
40         for(auto p : (*mcps)){
41             std::cout << "Particle "<< p.getObjectID().index<<" ID : "
42                       << p.getPdgID()<<std::endl;
43         }
44     }
45     else {
46        throw std::runtime_error("Collection 'mcps' should be present");
47     }
48
49     LogDebug << "Processing event " << m_iEvt << std::endl;
```

More Examples are provided

```
[huangxt@lxslc606 Examples]$ ls
DemoSim  EDMtests  FullSim  Geometry  HelloHist  HelloTree  HelloWorld  SimpleDetector
```

- Package :/afs/ihep.ac.cn/soft/HERD/HERDOS/offline/Examples
  - ⇨ HelloWorld: a simple algorithm example
  - ⇨ HelloHist: Define and Write Histogram into root file
  - ⇨ HelloTree: Define and write Ttree into root file
  - ⇨ DemoSim: detector description with DD4hep (modify Geant4/B2a Example)
  - ⇨ EDMtests: Write/Read event data to/from Root files

Software Environments and Management

- **Programming language: C++ and Python**
  - ⇨ C++ : main part implementation
  - ⇨ Python : job configuration interface
- **Packages management tool: CMake**
  - ⇨ Help developers to compile packages easily
  - ⇨ Help users to setup the environment for running the application easily
- **Operation System: Scientific Linux**
  - ⇨ Scientific Linux 6/CentOS 7 or higher
  - ⇨ G++ > 6.5.0 (C++14)
- **Version Control System: GitLab: http://code.ihep.ac.cn/herdos/offline**
  - ⇨ Keep the history of code evolution
  - ⇨ Synchronization and sharing between developers
  - ⇨ Tag and release
- **Installation**
  - ⇨ A shell script is provided to Automatically install the whole offline software

Summary

- Setup SNiPER-based software prototype for testing
- Develop Detector Simulation system
  - ⇨ Describe detector geometry with DD4hep
  - ⇨ Simulate detector response with Geant4
  - ⇨ Event looping with SNiPER
- Define Event Data with PODIO
  - ⇨ Define Event Data Model with yaml file
  - ⇨ Event Data Management with PodioDataSvc
  - ⇨ Provide Handles to get/put data from/to EventStore of PODIO
  - ⇨ PodioOutputSvc/PodioinputSvc for writing and reading data from root files
- Most popular tools/compiler have been used
  - ⇨ Cmake, Gitlab, C++14,TBB

More collaboration on software !

Thanks for your attention !