Get Started with Analysis in OSCAR

Version 0.0.1

STCF Physics Group

July 16, 2025

Contents

List of Code Listings

1 Introduction

This document is an example of a software manual for OSCAR framework.

2 Getting started

author: Hang Zhou

2.1 Prerequisites

It is recommended to install and run OSCAR on a server. To apply for a USTC server account, please contact hepglmh@ustc.edu.cn. SSH protocol is typically used to connect to the server.

The GitLab is used to manage the development and release of the OSCAR. You can find all the codes at the GitLab repository

```
https://git.ustc.edu.cn/oscar1/offline
```

Access to the repository requires an account. If you have a USTC email address with the suffix ustc.edu.cn, you can easily apply for an account according to the website instructions. Users from other institutions, please contact qipenghu@ustc.edu.cn to apply for a GitLab account. The OSCAR repository is not public, so you need to contact the administrators(qbb180@ustc.edu.cn) to join the group.

2.2 Building the project

Log in to the server node and build your working directory.

```
$ ssh username@stcf01.ustc.edu.cn
or
$ ssh username@stcf02.ustc.edu.cn

$ mkdir your_workarea
$ cd your_workarea
```

If you have joined the OSCAR group in GitLab, the following command can be used to clone the repository.

```
$ git clone https://git.ustc.edu.cn/oscar1/offline.git
```

Otherwise, you may be able to retrieve the code from some private directories, but this is not recommended.

Set the third party software environment and build the project.

```
$ source /software/STCF/OSCAR/ExternalLibs-gcc85/bashrc.sh
$ cd your_workarea/offline
$ ./build.sh
```

2.3 Setting OSCAR environments

You can set up a pre-installed OSCAR environment by the command:

```
$ source /software/STCF/OSCAR/release_version/setup.sh
```

Where "release_version" should be replaced by a released OSCAR version number, for example:

```
$ source /software/STCF/OSCAR/2.5.0/setup.sh
```

It is highly recommend to use your own OSCAR environment, because if you change or add a package, you have to recompile the OSCAR. In that condition, following commands are used to set the environments for running jobs.

```
$ source /software/STCF/OSCAR/ExternalLibs-gcc85/bashrc.sh
$ source your_workarea/offline/install/setup.sh
```

Note!

If you want to recompile OSCAR, the second line of the above command is unnecessary and should not be used.

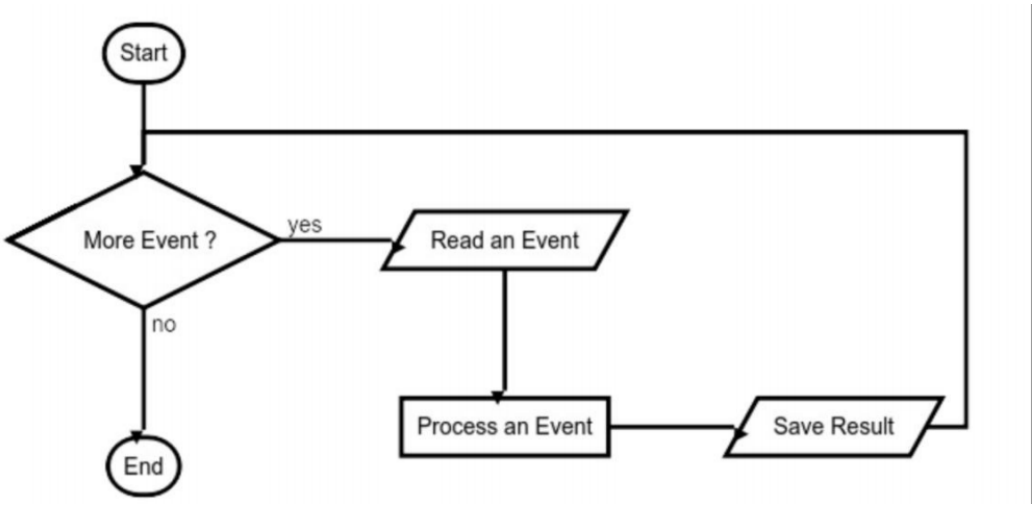

Figure 1: Data processing procedure

2.4 Main components of OSCAR

The typical procedures are as illustrated in Figure 1:
1. Read in the event data,
2. Process the event via launching some subroutines,
3. Write out the result.

These subroutines are implemented with Algorithms, Services and Tools.

Algorithm is a specific calculation subroutine, which applies to each event and is invoked by the framework during the event loop.

Service is another type of subroutine, which usually provides a specific functionality for example, users can write the data into root file via RootWriter service, or access the database via the DatabaseSvc, or initialize to the detector geometry via the DDXMLSvc. It can be invoked by User's code (such as a algorithm) or the framework.

The tool is similar with algorithm and service, it is also a dynamically loadable element, but it belongs to an specific algorithm and helps the algorithm to organize code more flexibly.

Task is a lightweight application manager, it is a controller for the event looping and the entrance of a job. As illustrated in Figure 2, Task consists of algorithms, service, and sub-tasks, it controls the execution of all algorithms belonging to it according the order of the algorithms when they are created

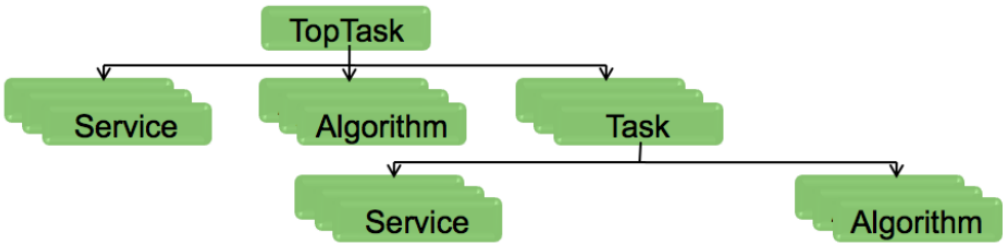

Figure 2: Task architecture

or added.

3 Simulation and Reconstruction

author: Hang Zhou

After setting up the OSCAR environment, you can run jobs using Python scripts:

```
$ python job_script.py
```

User interface via python steering code is under development.The following sections will introduce some usage examples.Usable scripts can be found in the directory

```
OSCAR_area/offline/Examples/RunCheck
```

3.1 Simulation

3.1.1 Generators introduction

Several generators are integrated into OSCAR,including Babayaga, BBBrem, DIAG36, KKMC and Phokhara. Additionally, the StcfEvtGen package is used to control the decay of short-lived particles. To use these generators, you need to import the corresponding packages in the script file and set the parameters. For example:

```
import Babayaga #import the package
babaalg = task.createAlg("Babayaga") #create the Babayaga algorithm
# 1: e+e-->e+e-;2:e+e_->mu+mu-;3:e+e-->gamma gamma;4:e+e--->pi+pi-
babaalg.property("Channel").set(2)
babaalg.property("Ebeam").set(2) # Ecm = 2*Ebeam [GeV]
babaalg.property("MinThetaAngle").set(20) # [degree]
babaalg.property("MaxThetaAngle").set(160) # [degree]
```

This is an example demonstrating how to use the Babayaga generator.Particles can also be generated using G4ParticleSource. In this case, there is no need to import additional generator packages, but a macro file specifying the distribution of particle parameters is required.

```
import G4Svc
g4svc = task.createSvc("G4Svc")
g4svc.property("RunMac").set("run_test.mac")
#macro file specifying particle parameters

import FullSim
genTool = simalg.createTool("GeneratorMgr")
genTool.property("Translation").set([0,0,0])
#generator translation [mm]
#genTool.property("ParticleSource").set("Generator")
#Generator or ParticleGun
genTool.property("ParticleSource").set("ParticleGun")
#Generator or ParticleGun
genTool.property("Boost").set(False)
#only work when ParticleSource == Generator
```

3.1.2 Examples of event generation

Below is a complete example of a simulation script:

OSCAR_area/offline/Examples/RunCheck/simPiPiMuMu.py

This script generates the Psi(2s) -> pi+pi-mu+mu- event and simulates the detector response.

```
import KKMC
detalg0 = task.createAlg("KKMC")
detalg0.property("CMSEnergy").set(3.686)
detalg0.property("GenerateResonance").set(True)
detalg0.property("GenerateContinuum").set(False)
#detalg0.property("GenerateDownQuark").set(False)
#detalg0.property("GenerateUpQuark").set(False)
#detalg0.property("GenerateStrangeQuark").set(False)
#detalg0.property("GenerateCharmQuark").set(True)
#detalg0.property("GenerateBottomQuark").set(False)
detalg0.property("GenerateMuonPair").set(False)
detalg0.property("GenerateTauPair").set(False)
detalg0.property("GenerateRho").set(False)
detalg0.property("GenerateOmega").set(False)
detalg0.property("GeneratePhi").set(False)
detalg0.property("GenerateJPsi").set(False)
detalg0.property("GeneratePsiPrime").set(True)
detalg0.property("GeneratePsi3770").set(False)
detalg0.property("GeneratePsi4030").set(False)
detalg0.property("GeneratePsi4160").set(False)
detalg0.property("GeneratePsi4415").set(False)
detalg0.property("GeneratePsi4260").set(False)
detalg0.property("ParticleDecayThroughEvtGen").set(True)
detalg0.property("RadiationCorrection").set(True)


import StcfEvtGen
detalg = task.createAlg("EvtDecay")
detalg.property("DecayDecDir").set("pipijpsi.dec")
detalg.property("userDecayTableName").set("pipijpsi.dec")
```

Using KKMC and StcfEvtGen, StcfEvtGen controls the decay of particles through processes specified in the decay card.

```
detalg.property("DecayDecDir").set("pipijpsi.dec")
```

If no path is specified, the default decay table will be used.

```
OSCAR_area/offline/Generator/StcfEvtGen/share/DECAY.DEC
```

You can use your own decay card.

```
detalg.property("userDecayTableName").set("pipijpsi.dec")
```

```
#
Decay psi(2S)
1.0000    J/psi    pi+  pi-    JPIPI;
Enddecay
#
Decay J/psi
1.0000    mu+   mu-    PHOTOS VLL;
Enddecay

End
```

The following code specifies the output file and the list of data collections to be saved.

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

import PodioSvc
svc = task.createSvc("PodioOutputSvc/OutputSvc")
svc.property("OutputFile").set("simPiPiMuMu_0.root")
reduced_coll_list = ["EventHeaderCol","MCParticleCol","ITKPointCol",
"ITKHitCol","MDCHitCol","DTOFBarHitCol","DTOFPDHitCol",
"RICHHitCol","ECALHit3Col","ECALTPointCol",
"MUDPointCol","MUDHitCol","BkgSEEHitCol"]
svc.property("OutputCollections").set(reduced_coll_list)
```

And services record detector information and provide geometry for simulation.

```
import MdcGeomSvc
Mdcgeom1 = task.createSvc("MdcGeomSvc")
Mdcgeom1.property("geomDataFilePath").set \
(topdir+"/CommonSvc/MDCSvc/MdcGeomSvc/dat/mdcGeomData.dat")
Mdcgeom1.property("geomDataSource").set(0)

import GeometrySvc
myxmlsvc = task.createSvc("GeometrySvc")
myxmlsvc.property("GeoCompactFileName").set \
(topdir+"/Geometry/FullGeometry/compact/STCF.xml")
```

```
import G4Svc
g4svc = task.createSvc("G4Svc")

import DetSimAlg
simalg = task.createAlg("DetSimAlg/DetSimAlg")
simalg.property("DetFactory").set("FullFactory")

import FullSim
genTool = simalg.createTool("GeneratorMgr")
genTool.property("Translation").set([0,0,0])
#generator translation [mm]
genTool.property("ParticleSource").set("Generator")
#Generator or ParticleGun
genTool.property("Boost").set(False)
#only work when ParticleSource == Generator

geoTool = simalg.createTool("GeoAnaMgr")

mdcsim = simalg.createTool("MDCAnaMgr")
mdcsim.property("InducedQSim").set(True)

itksim = simalg.createTool("ITKAnaMgr")
itksim.property("UsePAI").set(True)

factory = task.createSvc("FullSimFactory/FullFactory")
factory.property("OpticalSimOn").set(True)
factory.property("PAISimOn").set(True)
factory.property("ITKPAISimOn").set(True)
factory.property("useRandom").set(False)
#If this is set to true, time will be used as the random seed;
#otherwise, the seed set in the next line will be used.
factory.property("RandomSeed").set(304334425)
factory.property("AnaMgrList").set \
(["GeoAnaMgr","GeneratorMgr","ECALAnaMgr","DTOFAnaMgr", \
"RICHAnaMgr","MUDAnaMgr","MCTruthWriter"])
```

The above code configures the simulation settings.

Run the simulation script:

```
$ python simPiPiMuMu.py
```

3.2 Reconstruction

In the current workflow, digitization and reconstruction run within the same script, using simulation data files as input. However, please note that digitization is not part of the reconstruction. The reconstruction script

```
OSCAR_area/offline/Examples/RunCheck/rec_Full_forAnalysis.py
```

is mainly divided into two parts:
background mixing and digitization
sub-detector reconstruction and reconstruction information assembly.

You only need to change the input and output file paths.

```
AlgTask.addInputFile("simSingleMu.root", "Sig") #signal hits
#The following lines are used to add the background input
#AlgTask.addInputFile(BKGDir+"pipiee/pipiee_100_total.root", "Bkg_Phys")
#for i in range(len(BKGNameList)):
#  AlgTask.addInputFile(BKGFileList[i], BKGNameList[i])

outsvc = AlgTask.createSvc("PodioOutputSvc/OutputSvc")
outsvc.property("OutputFile").set("Rec_SingleMuTest0.root")
#outsvc.property("TransferCollections").set(["MDCHitCol"])
#outsvc.property("TransferAllCollections").set(True)

import RootWriter
osvc = AlgTask.createSvc("RootWriter/hSvc")
osvc.property("Output").set({"Fkey" : "Info3.root"})
```

The two root files "simSingleMu.root" and "Rec_SingleMuTest0.root" are OSCAR standard data files, the input file is the output file of the full simulation. "Info3.root" is used to store user-defined data. How to use RootWriter will be introduced in section 4.1.7.

3.3 Job submission

See section 5.

4 Analysis

Two examples of analysis package can be found in:

```
OSCAR_area/offline/Analysis/RhoPiAnalysis
OSCAR_area/offline/Analysis/AnalysisTest
```

4.1 How to write an analysis package

4.1.1 Head files

author: Hang Zhou
To write an analysis software package, the following header files are necessary.

1. Framework (Sniper) functional modules

```
#include "SniperKernel/AlgBase.h"
#include "SniperKernel/SniperLog.h"
#include "SniperKernel/AlgFactory.h"
#include "SniperKernel/SniperPtr.h"
#include "SniperKernel/Incident.h"
```

2. DataModel

```cpp
#include "DataModel/MCParticle.h"
#include "DataModel/MCParticleCollection.h"
#include "DataModel/TrackerHypoTrackCollection.h"
#include "DataModel/DTOFBarHitCollection.h"
#include "DataModel/DTOFHypoRecCollection.h"
#include "DataModel/DTOFHypoTrackCollection.h"
#include "DataModel/DTOFPDHitCollection.h"
#include "DataModel/DTOFpeHitCollection.h"
#include "DataModel/DTOFPidCollection.h"
#include "DataModel/DTOFDigiCollection.h"
#include "DataModel/CanTrackCollection.h"
#include "DataModel/RecExtTrackCollection.h"
#include "DataModel/MCParticleCollection.h"
#include "DataModel/TrackerRecTrackCollection.h"
#include "DataModel/dEdXRecTrackCollection.h"
#include "DataModel/ReconstructedParticleCollection.h"
#include "DataModel/DTOFPidCollection.h"
#include "DataModel/RecRICHLikelihoodCollection.h"
#include "DataModel/RecECALShowerCollection.h"
#include "DataModel/MUDTrackCollection.h"
#include "DataModel/MUDClusterCollection.h"
```

3. Packages for analysis, such as PID, vertex fitting, and kinematic fitting

```cpp
#include "GlobalPID/GlobalPIDSvc.h"
#include "VertexFit/KalmanKinematicFit.h"
#include "VertexFit/KinematicFit.h"
#include "VertexFit/HTrackParameter.h"
#include "VertexFit/WTrackParameter.h"
#include "VertexFit/VertexFit.h"
#include "VertexFit/BField.h"
#include "RootWriter/RootWriter.h"
```

4. Standard C++ library headers or other external library headers that are needed

```
#include <stdio.h>
#include <map>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>
#include <string.h>
#include <signal.h>
#include "TVector3.h"
#include "TMatrixD.h"
#include "TInterpreter.h"
```

4.1.2 Charged track selection

author: Hang Zhou

Reconstructed information is combined to form a "ReconstructedParticle". A ReconstructedParticle contains the reconstructed information of a charged particle as well as the PID information corresponding to that particle.A ReconstructedParticle can also be created from calorimeter reconstruction results that cannot be associated with a charged particle.To get charged track information:

```
auto m_recParticle = getROColl(ReconstructedParticleCollection,
"ReconstructedParticleCol");
if (!m_recParticle)
    return;
int nRecParSize = m_recParticle->size();
for (int i = 0; i < nRecParSize; i++)
{
    ReconstructedParticle aRecPar = m_recParticle->at(i);
    if (aRecPar.getTrack().isAvailable()){
        TrackerRecTrack aRecTrack = aRecPar.getTrack();
    }
}
```

Track information can also be directly obtained through the collection of charged particle tracks.

```
auto m_RecTracksCol = getROColl(TrackerRecTrackCollection,
"TrackerRecTrackCol");
if (!m_RecTracksCol)
    return;
int nRecParSize = m_RecTracksCol->size();
for (int i = 0; i < nRecParSize; i++)
{
    TrackerRecTrack aRecTrack = m_recParticle->at(i).getTrack();
}
```

Each TrackerRecTrack stores the fitting information for five particle hypotheses.

```
for (int j = 0; j < 5; j++)
{
  TrackerHypoTrack aTrackHypo = aRecTrack.getTrackHypos(j);
}
```

Index j, ranging from 1 to 5, represents the five particle hypotheses: *e*, μ, π, *K*, *p*, respectively.Track parameters are obtained in the following way:

```
Vector3d PosFirst = aTrackHypo.getPosFirst();
Vector3d MomFirst = aTrackHypo.getMomFirst();
//momentum and position at first measurement plane

Vector3d PosPOCA = aTrackHypo.getPosPOCA();
Vector3d MomPOCA = aTrackHypo.getMomPOCA();
//momentum and position at POCA

double helixP0 = aTrackHypo.getD0();
double helixP1= aTrackHypo.getPhi0();
double helixP2 = aTrackHypo.getKap();
double helixP3 = aTrackHypo.getZ0();
double helixP4 = aTrackHypo.getTanlamda();
//five helix parameters at POCA
```

The definitions of the five helix parameters are shown in Figure 3, with the pivot point being the origin (0, 0, 0) by default.

The PID information corresponding to the charged track can also be obtained

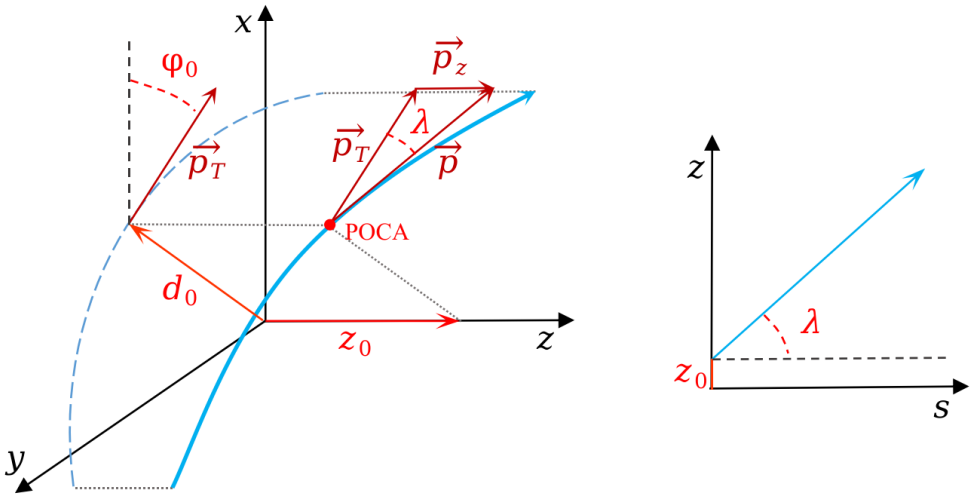

Figure 3: Demonstration of helix parameters

form the ReconstructedParticle.

```
if (aRecPar.getMudTrack().isAvailable())
{
 cout << "MUD info:" << aRecPar.getMudTrack().getIsMuon()
 << "BDT value:" << aRecPar.getMudTrack().getBDTExp();
}
if (aRecPar.getDEdX().isAvailable())
{
 cout << "de/dx chi2:" << aRecPar.getDEdX().getChiE();
 //The following functions can be used,
 //a smaller chi value for a given particle hypothesis suggests
 //a higher likelihood that the particle is of that type.
 /// Access the chi value (Electron hypothesis)
 // const double& getChiE() const;

  /// Access the chi value (Muon hypothesis)
 // const double& getChiMU() const;

  /// Access the chi value (Pion hypothesis)
  //const double& getChiPI() const;

  /// Access the chi value (Kaon hypothesis)
  //const double& getChiK() const;

  /// Access the chi value (Proton hypothesis)
 // const double& getChiP() const;
}
if (aRecPar.getRICHPid().isAvailable())
{
 cout << "RICH likelihood:" << aRecPar.getRICHPid().getLikelihood_e();
 // more functions,
 //see offline/install/include/DataModel/RecRICHLikelihood.h

}
if (aRecPar.getDTOFPid().isAvailable())
{
    for (int j = 0; j < 5; j++)
        cout << "  DTOF likelihood:" << aRecPar.getDTOFPid().getDTOFHypoTrackL
    // for (int j = 0; j < aRecPar.getDTOFPid().DTOFHypoRecList_size(); j++)
    // {
    //     cout << "  DTOF likelihood:" << aRecPar.getDTOFPid().getDTOFHypoRecL
    // }
}
   // more functions, see offline/install/include/DataModel/DTOFPid.h
   // offline/install/include/DataModel/DTOFHypoRec.h
```

Get neutral track from ReconstructedParticle:

```
if (aRecPar.getShower().isAvailable())
{
    const RecECALShower aShower = aRecPar.getShower();
}
```

4.1.3 Neutral track selection

author: Bo Wang

```
// Get the ECAL Shower Collection:
// Used in the .h
RecECALShowerCollection* mShower;
// Use in the execute() in .cc
mShower = getRWColl(RecECALShowerCollection, "RecECALShowerCol");
// Get the information of each shower:
for(size_t i=0; i<mShower->size(); i++){  // Loop all showers in one event
  const RecECALShower aShower = m_shower->at(i);
  double Energy = aShower.getEnergy();
  double Time = aShower.getSeed().getTime();
  double RecX = aShower.getPosition().x;
  double RecY = aShower.getPosition().y;
  double RecZ = aShower.getPosition().z;
  double RecTheta = atan2(sqrt(RecX*RecX+RecY*RecY),RecZ);
  double RecPhi = atan2(RecY,RecX);
}
```

Listing 1: Example C++ code of Neutral track selection

4.1.4 Particle identification

author: Yuncong Zhai

Particle identification (PID), i.e., the ability to discriminate between different particle species produced during the collision, is one of the most important and commonly used analysis tools in high-energy physics experiments.

Traditional PID methods such as maximum likelihood estimation have been successfully applied in collider physics experiments. However, the process of constructing PID features and performing weighted processing using the measurement results of sub-detectors is complex, difficult, and inefficient, making it challenging to meet the requirements of PID.

Therefore, in the STCF experiment, we have developed a new particle identification software package based on data-driven machine learning methods. The GlobalPIDSvc software package includes pre-trained BDT (based on XGBoost) model and algorithm, and is an important part of the OSCAR software package's Analysis branch.To help analysts become familiar with the software package and its functionalities, the user manual is as follows:

1 Users need to add the directive to include the GlobalPID header file in the source file of the instance selection program.

```
#include "GlobalPID/GlobalPIDSvc.h"
```

2 Users need to check and retrieve the GlobalPIDSvc instance in the initialize() function of the instance selection program.

```
SniperPtr<GlobalPIDSvc> _globalpidsvc(getParent(), "GlobalPIDSvc");
if ( _globalpidsvc.valid() ) {
  LogInfo << "the GlobalPIDSvc instance is retrieved" << std::endl;
}
else{
  LogError << "Failed to get the GlobalPIDSvc instance!" << std::endl;
  return false;
}
m_pid = _globalpidsvc.data();
```

3 To obtain the information of a specific track which needs particle identification as well as the information of each subdetector.

```
m_pid->calculate(RecParticle);
```

4 Users can choose the PID mode, the currently supported PID modes are: All ($e/\mu/\pi/K/p$), $\pi/K/p$, π/K, $e/\pi/K$, μ/π.

```
m_pid->setmode (m_pid->onlyKaon()|m_pid->onlyPion()|m_pid->onlyProton());
m_pid->setmode (m_pid->onlyPionKaonProton());
```

5 Users can obtain the predicted probabilities of the trajectory under five particle hypotheses.

```
float m_prob_e = m_pid->prob(Electron);
float m_prob_mu = m_pid->prob(Muon);
float m_prob_pi = m_pid->prob(Pion);
float m_prob_k = m_pid->prob(Kaon);
float m_prob_p = m_pid->prob(Proton);
```

5 Configuration in Python running files.

```
import os
td = os.getenv("OFFLINETOP")
import GlobalPID
pidsvc = task.createSvc("GlobalPIDSvc")
pidsvc.property("SetModelPath").set(td+"/Analysis/GlobalPID/src/xgb.model
pidsvc.property("SetMethod").set("XGBoost")
```

4.1.5 Vertex fit

author: Mingyu Yu

The reconstruction of vertices by least-squares methods and Kalman methods is a mathematical procedure in which one uses the physical laws (especially the geometric constraints) governing a particle interaction or decay to improve the measurements of the process.There are two packages intergrated into OSCAR,including VertexFit and SecondVertexFit.

In VertexFit package, we suppose each input-track originated from a common vertex and the output is the common vertex with the information of renewed input parameters.To use this package, you can follow the steps below:

```
#include "VertexFit/VertexFit.h"  //include necessary header file

HepPoint3D vx(0., 0., 0.);
HepSymMatrix  Evx(3, 0);
double bx = 1E+6;
double by = 1E+6;
double bz = 1E+6;
Evx[0][0] = bx * bx;
Evx[1][1] = by * by;
Evx[2][2] = bz * bz;
VertexParameter vxpar;
vxpar.setVx(vx);
vxpar.setEvx(Evx);  //set a common vertex with huge error

//do vertex fit
VertexFit *vtxfit = VertexFit::instance();  //get instance of VertexFit
vtxfit->init();  //initial the pramaters of VertexFit
//add charged track in TrackerHypoTrack format(the third parameter) and
//the last parameter is a TrackerRecTrack format to get the charge
//of the track
vtxfit->AddTrack(0, m1,hptrk1,rectrk1);
vtxfit->AddTrack(1, m2,hptrk1,rectrk2);
vtxfit->AddVertex(0, vxpar, 0, 1);  //add common vertex
bool oksq = vtxfit->Fit(0);// whether the fitting is done successfully
if(!oksq) return true;
double chisq = vtxfit->chisq(0);
//get the chi-square value of vertex fit
vtxfit->Swim(0);  //renew information of input-track
WTrakParameter wtrk1 = vtxfit->wtrk(0);
//renewed track parameters in WTrakParameter format
WTrakParameter wtrk2 = vtxfit->wtrk(1);
HepVector vtxPos = vtxfit->Vx(0);  //common vertex
HepSymMatrix vtxPosErr = vtxfit->Evx(0);
//covariance matrix of common vertex
```

Listing 2: Example C++ code of VertexFit

In SecondVertexFit package, we handle the particle with the secondary vertex.

The input requires production point, decay point, the four-momentum at decay point and the output is decay length in flight with the information of renewed input parameters.To use this package, you can follow the steps below:

```
#include "VertexFit/SecondVertexFit.h"
#include "VertexFit/VertexFit.h"
//set the covariance using a random number which has the same magnitude
//level with average IP in each run on BESIII
srand((int)time(NULL));
double a = 1 + rand() % (10);
a = a / 1000;
double b = 1 + rand() % (10);
b = b / 1000;
double c = 1 + rand() % (10);
c = c / 10;
HepPoint3D ip(0., 0., 0.);
HepSymMatrix  ipEx(3, 0);
ipEx[0][0]=a;
ipEx[1][1]=b;
ipEx[2][2]=c;
VertexParameter bs;
bs.setVx(ip);
bs.setEvx(ipEx);//set IP information
HepPoint3D vx(0., 0., 0.);
HepSymMatrix  Evx(3, 0);
double bx = 1E+6;
double by = 1E+6;
double bz = 1E+6;
Evx[0][0] = bx * bx;
Evx[1][1] = by * by;
Evx[2][2] = bz * bz;
VertexParameter vxpar;
vxpar.setVx(vx);
vxpar.setEvx(Evx);//set a common vertex with huge error
```

Listing 3: Example C++ code of SecondVertexFit(part 1)

```
//do vertex fit
VertexFit *vtxfit = VertexFit::instance();
vtxfit->init();
//add charged track in TrackerHypoTrack format(the third parameter) and
//the last parameter is a TrackerRecTrack format to get the charge of the track
vtxfit->AddTrack(0, m1,hptrk1,rectrk1);
vtxfit->AddTrack(1, m2,hptrk2,rectrk2);
vtxfit->AddVertex(0, vxpar, 0, 1);//add common vertex
bool oksq1 = vtxfit->Fit(0);
if (!oksq1) return true;
double vtx_chisq= vtxfit->chisq(0);
vtxfit->Swim(0);
vtxfit->BuildVirtualParticle(0);
```

Listing 4: Example C++ code of SecondVertexFit (part 2)

```
//do second vertex fit
SecondVertexFit *svtxfit = SecondVertexFit::instance();
//get instance of SecondVertexFit
svtxfit->init();
svtxfit->setPrimaryVertex(bs);//set primary vertex
svtxfit->AddTrack(0, vtxfit->wVirtualTrack(0));
svtxfit->setVpar(vtxfit->vpar(0));//set second vertex
bool oksq2 = svtxfit->Fit();
if (!oksq2) return true;
double svtx_chisq = svtxfit->chisq();
WTrakParameter wtrk1 = vtxfit->wtrk(0);
WTrakParameter wtrk2 = vtxfit->wtrk(1);
HepLorentzVector p = svtxfit->p4par();
//the 4-momentum of the intermediate particles
HepVector vtx = svtxfit->vpar().Vx();//get second vertex
double ctau = svtxfit->ctau();
double len = svtxfit->decayLength();
double lenerr = svtxfit->decayLengthError();
```

Listing 5: Example C++ code of SecondVertexFit (part 3)

4.1.6 Kinematic fit

author: Mingyu Yu

In Kinematic Fitting, some of physics laws are used to constraint the decay process, then the resolutions of particle parameters are improved. Many sorts of constraints can used in Kinematic Fitting, such as four momentum constraints(4C), resonance constraints and so on.To use this package, you can follow the steps below:

```
//Take RhoPi events as a reference
#include "VertexFit/KinematicFit.h"
//#include "VertexFit/KalmanKinematicFit.h"

KinematicFit * kmfit = KinematicFit::instance();
// get instance of KinematicFit
// get instance of KalmanKinematicFit
//KalmanKinematicFit * kmfit = KalmanKinematicFit::instance();
kmfit->init(); // initial the pramaters of kinematic fitting
//add charged tracks in WTrackParameter format,which can get after vertexfit
kmfit->AddTrack(0,wpip);
kmfit->AddTrack(1,wpim);
//add neutral track in RecECALShower format(the third parameter)
kmfit->AddTrack(2, 0.0, g1Trk);
kmfit->AddTrack(3, 0.0, g2Trk);
kmfit->AddResonance(0,mass, 2, 3); // add resonance constraints
kmfit->AddFourMomentum(1, ecms); // add 4c constraints
bool oksq = kmfit->Fit();
if(!oksq) return true;
HepLorentzVector ppip = kmfit->pfit(0);
// get the 4-momentum after kinematic fitting
double chisq  = kmfit->chisq();
// get the chi-square value of kinemtic fitting
```

Listing 6: Example C++ code of KinematicFit

4.1.7 Writing variables to ROOT

author: Hang Zhou/Yuncong Zhai

The service "RootWriter" is used to save user-defined data.The output data is saved in a ROOT file. You have to define the data types that need to be saved, for example:

```
TTree* recInfo;
int EventID;
double JpsiMass;
double PsiMass;
double KinChi2;
```

Then call the RootWriter service and register the TTree and data branches:

```
SniperPtr<RootWriter> hs(getParent(), "hSvc");
if (hs.valid())
{
    {
        recInfo = hs->bookTree(*getParent(),
        "Fkey/AnalysisTest", "title");
        //“Fkey“ is the key corresponding to the root file
        //where this tree will be saved.
        recInfo->Branch("eventID", &EventID);
        recInfo->Branch("JpsiMass", &JpsiMass);
        recInfo->Branch("PsiMass", &PsiMass);
        recInfo->Branch("KineticFitChi2", &KinChi2);
    }
}
```

To use RootWriter, you need to include the relevant parts in your script:

```
import RootWriter
osvc = AlgTask.createSvc("RootWriter/hSvc")
osvc.property("Output").set({"Fkey" : "recPiPiMuMu_TrackingInfo_1.root"})
#The first parameter is the key corresponding to the root file,
#and the second parameter is the output file name.
```

4.1.8 Create a new analysis package

The analysis package is located at /offline/Analysis and includes three directories: headers, src, python, and a CMakeLists.txt file.In the current situation,

it is recommended to create a new analysis package by copying and modifying the existing package.If a new package is added, you need to add the compile directory in offline/Analysis/CMakeLists.txt.

```
set(dirlist RhoPiAnalysis VertexFit MultiStreamAnalysis GlobalPID
AnalysisTest SingleParticleAna)
#add your own package to the dirlist
foreach(dir ${dirlist})
  add_subdirectory(${dir})
endforeach()
```

4.2 Run analysis jobs

author: Hang Zhou, Bo Wang, Yuncong Zhai, Mingyu Yu

Example script of running analysis package:

OSCAR_area/offline/Examples/RunCheck/Ana.py

```
#!/usr/bin/env python
#******************General_begin
import os
topdir = os.getenv('OFFLINETOP')

import Sniper
AlgTask = Sniper.Task("AlgTask")
AlgTask.setLogLevel(0) #higher is less

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

import PodioSvc
Isvc = AlgTask.createSvc("PodioInputSvc/InputSvc")
Isvc.property("InputFile").set("Rec_SingleMuTest0.root")

outsvc = AlgTask.createSvc("PodioOutputSvc/OutputSvc")
outsvc.property("OutputFile").set("AnaTest.root")

import RootWriter
osvc = AlgTask.createSvc("RootWriter/hSvc")
osvc.property("Output").set({"Fkey" : "Info0.root"})

import TrackInfoSvc
trInfo = AlgTask.createSvc("TrackInfoSvc")

import DeDxRec
dedxrecsvc = AlgTask.createSvc("DeDxRecSvc")

#Global
import GlobalPID
GlobalPID = AlgTask.createSvc("GlobalPIDSvc")
GlobalPID.property("SetModelPath") \
.set(topdir+"/Analysis/GlobalPID/src/xgb.model")
GlobalPID.property("SetMethod").set("XGBoost")

import AnalysisTest
AnaTest = AlgTask.createAlg("AnalysisTest")
AnaTest.property("CheckGlobalPID").set(False)
AnaTest.property("CheckAssembler").set(True)

AlgTask.setEvtMax(100)
AlgTask.show()
AlgTask.run()
```

The script should include the necessary services and the algorithms required for the analysis.

5 Scripts for job submission

author: Hang Zhou

HTCondor is used to manage batch jobs on server. More details about HTCondor, see https://htcondor.readthedocs.io/en/latest/. Examples of running batch jobs can be found in the following directories:

```
OSCAR_area/offline/Examples/JobSubmission
OSCAR_area/offline/Performance/XXXXWriter/share
```

It is usually to generate the scripts needed to run jobs by adjusting some parameters in the template. A template,for instance:

```
OSCAR_area/offline/Examples/JobSubmission/condor/template/sim_template.py
```

In this template, some parameters that need to be set by the user are replaced with placeholders, such as the random number seed.

```
import StcfRndmGenSvc
sSvc=task.createSvc("StcfRndmGenSvc/hSvc")
sSvc.property("RndmSeed").set(RANDOM)
```

Then these placeholders will be replaced and scripts will be generated in bulk.See

```
OSCAR_area/offline/Examples/JobSubmission/condor/sub_job.sh
```

To submit jobs using HTCondor, a submit description file is needed.

```
Universe              = vanilla
Notification          = Never
GetEnv                = True
accounting_group      = long
#Default:short     time limit 8 hours
Executable            = SCRIPT
#SCRIPT: The job to be run
Arguments             = ID
Initialdir            = LOGDIR
Output                = NAME.out
#The program's output information.
Error                 = NAME.err
#Error message
Log                   = NAME.log
#Job running status information
Queue
```

Submit job to compute nodes:

```
$ condor_submit your_submit_description_file
```

Or control jobs using hepjob commands.Environment setup is required beforehand:

```
$ source /cvmfs/common.ihep.ac.cn/software/hepjob_lzu/setup_hepjob.sh ustc
```

Then you can submit jobs using the hep_sub command,like:

```
$ hep_sub "run.sh" -n 10 -o job.out -e job.err -wt long
```

To view more usage.

```
$ hep_sub --help
```

To batch generate scripts and submit jobs via shell scripts, refer to the files in the directory mentioned at the beginning of this section.