**Users Guide**

**LodeStar**

**(LHAASO Offline Data Processing Software Framework)**

Wenhao Huang, Xingtao Huang

Shandong University

May, 2019

**Table of Contents**

1. Introduction 4

2. Architecture 5

3. Status and Release 7

SVN Repository 7

Current Version 8

4. Getting Started 9

How to setup Environments 9

How to use CMT 10

How to use SVN 11

How to Run HelloWorld 12

How to develop detector simulation 14

A Class to get access to G4 14

Run/Event/Stacking/Tracking/Stepping Actions 14

DetSimFactory 15

How to define Event Data 17

Event Data Header and Object 17

XOD Tool 18

EDM Book 20

Config and make 21

How to write out trees and Ntuples 22

How to write out Histograms: 23

5. Main Components 24

Data processing procedure 24

Algorithm 25

What’s the Algorithm? 25

How to create an Algorithm 25

Service 28

What’s the Service? 28

How to create a Service. 28

How to use a Service. 30

Task 31

What’s the Task? 31

How to configure a Task. 31

Tool 33

What’s the Tool? 33

How to create a Tool 33

How to use a Tool 34

Property 35

What’s the Property? 35

How to use Properties 35

6. Data Model 36

7. Input/Ouput System 37

Input system 37

Output System 37

8. Generators 39

9. Simulation 40

Example:KM2ASim 42

10. Calibration 43

11. Reconstruction 44

How to get Geometry information in root file 44

12. Physics Analysis Tools 45

13. Appendix 46

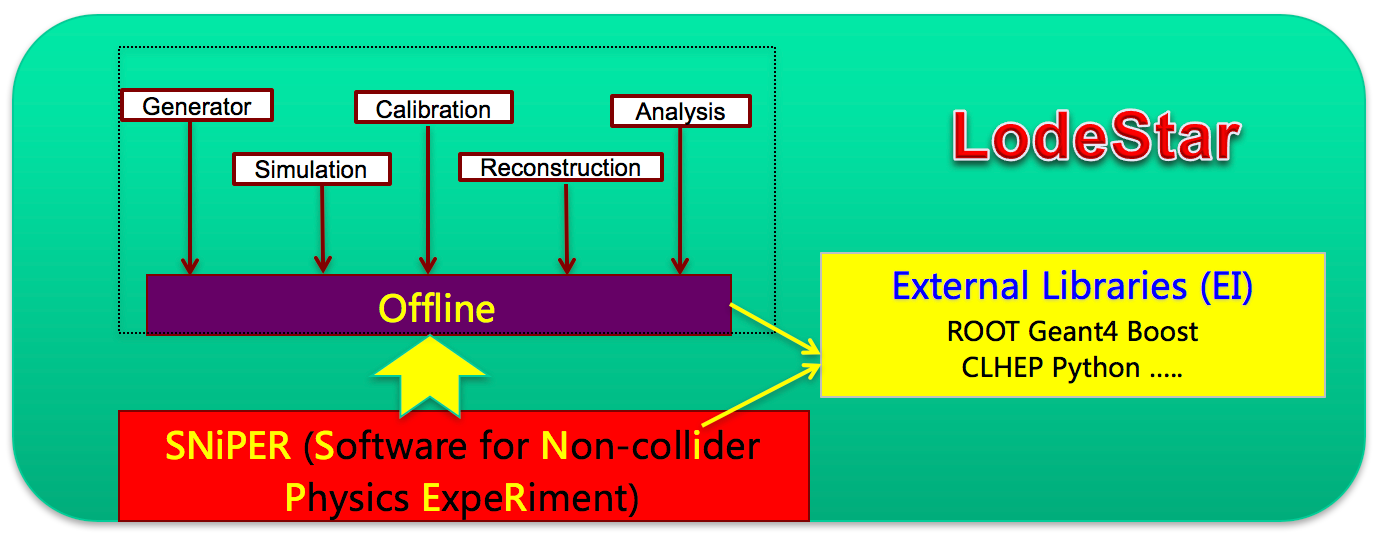
Installation 46

Coding Conventions 47

# Introduction

LodeStar is developed for the LHAASO offline data processing, including Monte Carlo (MC) data production as well as experimental data processing and physics analysis.

# Architecture



LodeStar consists of three CMT projects:

* Offline
  + this project is specific with LHAASO, it contains all the packages developed for LHAASO, for example, Generator, Simulation, Calibration and Reconstruction so on.
* SNiPER
  + this project is for the underlying framework to provide the common functionalities, such as data procedure controlling and common services.
  + Offline uses this project.
* ExternalLibs
  + this project is to build the interfaces for external libraries, therefore SNiPER and Offline can easily use external libs via the interfaces defined in this project.

These three projects will be automatically installed by the LodeStar, users usually create their own project (such as workarea), via the cmt command:

 cmt create\_project workarea

in their home directory. The new project will have a “cmt” subdirectory, which contains one file, project.cmt, which is used to define the relation with other projects. Because your own project definitely will use the offline project, you need put one line:

use offline

in the project.cmt file.

Each cmt project contains a set of cmt packages, which can be created via the command:

cmt create HelloWorld v0

Then you can write your own algorithms in this package.

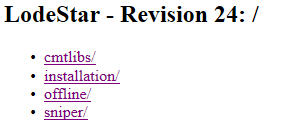
# Status and Release

## SVN Repository

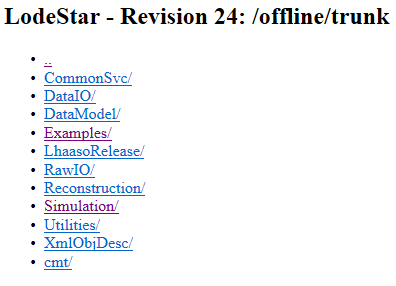
As explained in last chapter, all the codes of Offline, SNiPER and ExternalLibs projects are developed and owned by the LHAASO offline software group. The tool, SVN, is used to manage the development and release of the LodeStar. you can find all the codes at the url:

<http://svn.lhaaso.ihep.ac.cn/LodeStar/>

There are four software systems as illustrated below:



Currently, the Offline includes about 11 packages, each package, as its name, is corresponding some specific functionality for the LHAASO data processing.



In near future, the persons who is in charge the development of the package have the writing privilege to this package, other users only have the reading privilege.

Please contact with Wenhao Huang ([whyellow@mail.sdu.edu.cn)](mailto:whyellow@mail.sdu.edu.cn)) to open an SVN account if you need submit your codes into the SVN.

## Current Version

We install the official LodeStar into the public directory, users are suggested directly use the LodeStar from the public directory.



# Getting Started

## How to setup Environments

* **Set GCC (the GNU Compiler Collection) Environments**

The LodeStar software requires GCC version 4.8.5 or newer to use c++ 11 standard. While the default GCC version on SLC6 is 4.4.7. There must be a newer GCC installed.

Then the setup script for GCC can be check out from svn:

$svn export

<http://svn.lhaaso.ihep.ac.cn/LodeStar/installation/trunk/setup-gcc.sh>

Change the “path/to/gcc” to your GCC install path:



After sourcing this script, the GCC environments have been set up.

* **Set LodeStar Environments**

Set LodeStar environments （including GCC environment and other environments） are simple, and a LodeStar has been installed in a reachable path:

/afs/ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/L18-Pre1

So the command is:

$ source /afs/ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/L18-Pre1/setup.sh

## How to use CMT

Once you setup environments, you can have try to use cmt command:

$ cmt help

If the help information shows up, it means that your environments has been setup very well.

* **Several cmt commands are needed**
* cmt br: means broadcast. It will trigger all the dependencies to execute the same command.
* cmt config: will generate the scripts to setup the runtime environment and Makefile to build this package.
* cmt make: will invoke the make to build the package.

## How to use SVN

* **LodeStar SVN Server has been setup at the following URL:**

**<http://svn.lhaaso.ihep.ac.cn/LodeStar/>**

* **SVN Commands**

# checkout the code :

$ svn co http://svn.lhaaso.ihep.ac.cn/LodeStar/offline/trunk/

# if already exists , should update the code :

$ cd offline

$ svn update

# see current info

$ svn info

# if modify the code , compare the difference

$ svn status

$ svn diff

# if you want to track a new file :

$ svn add new\_file

# if you have the permission to check in the code :

$ svn ci -m " some notes about this commit "

## How to Run HelloWorld

* Login the computing node:

ssh –Y username@lxslc6.ihep.ac.cn

* Set the GCC version with the below command:

source /afs/ source /afs/ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/setgcc494.sh

* Setup the environment of LodeStar, which has been installed in public directory:

source /afs/ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/LPre1/setup.csh

* Create your own software project:

cmt create\_project workarea

The directory, workarea, will be automatically created with the sub-directory, cmt.

* Enter the directory:

cd workarea/cmt/

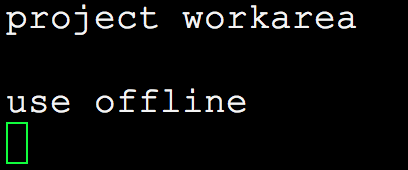
Modify the project configuration file, project.cmt, and add one line:

use offline

For example:

vim project.cmt

As below:



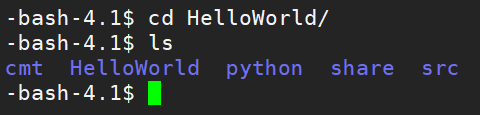
* Go back to the parent directory:

cd ../

* Checkout the Examples package from SVN server:

svn co http://svn.lhaaso.ihep.ac.cn/LodeStar/offline/trunk/Examples/HelloWorld

* The HelloWorld package will be available in your current directory, and it contains several sub-directory in the following:



* cmt: a requirements file, which is used for generate setup.\*sh and Makefile; setup.\*sh is used to setup the runtime environment for this package; and Makefile is used to compile and install this package.
* HelloWorld: for the public header files (\*.h).
* src: for the implementation files (\*.cc) or private header files (\*.h).
* python: for the default python file for loading the shared libraries of this package into framework.
* share: for the python scripts to run this example.
* Enter this package:

cd HelloWorld/cmt

There is a configuration file, requirements, which define the way to compile this package.

* The command below is used to configure this package according to the requirements file:

cmt br cmt config

* The command below is used to compile this package:

cmt br cmt make

* Setup the environments to run this package:

source setup.sh

* The share directory for each package is to place the python script:

cd ../share/

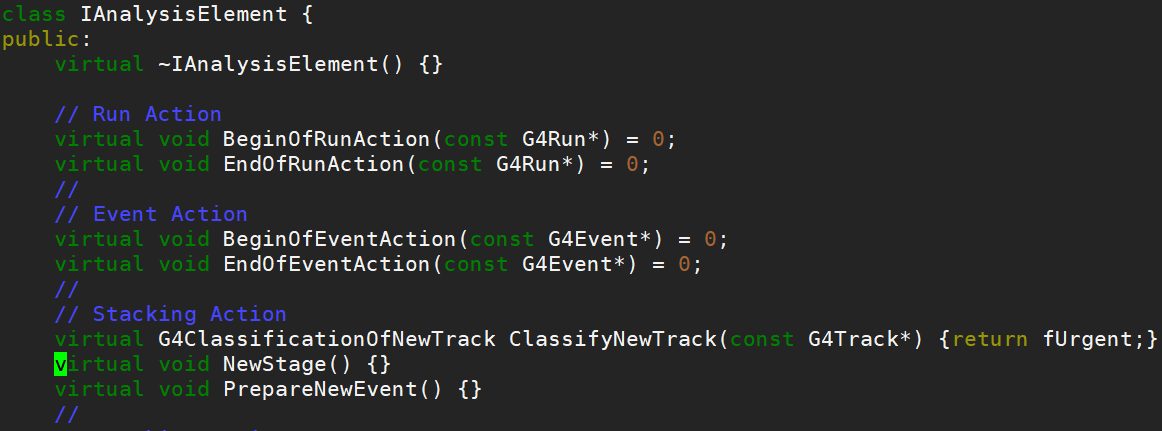
* Run HelloWorld example:

python run.py

## How to develop detector simulation

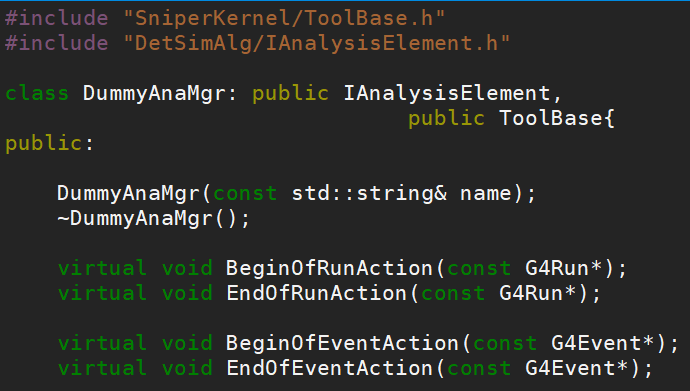
### A Class to get access to G4

We define a base class named IAnalysisElement to get access to G4 functions.



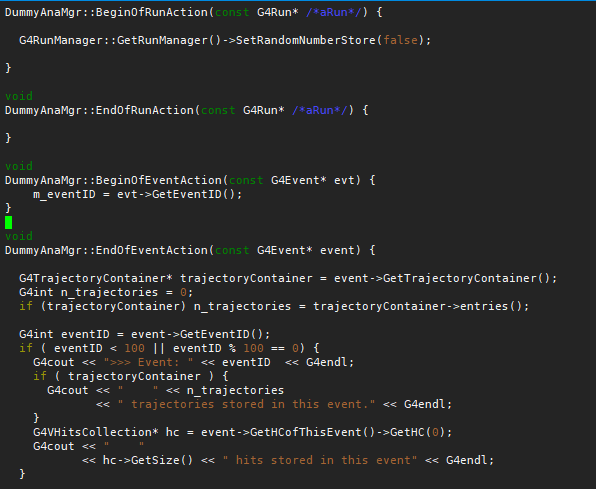
You need to define your own AnalysisElement by inheriting from it. There is a G4 example in basic “B2a” directory. Now we define this detector simulation in framework way.

Firstly, we define a class named “DummyAnaMgr”.



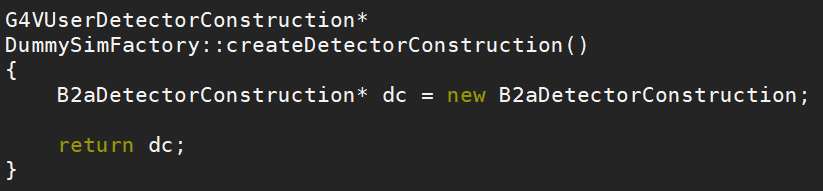
### Run/Event/Stacking/Tracking/Stepping Actions

If you want to use custom functions in these user actions, just define your own code in the class. For the “B2a” example, it uses custom “Run Action” and “Event Action”, so we just need to redefine the four functions above.

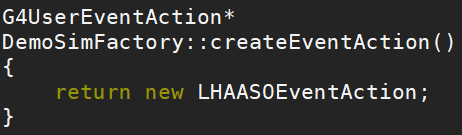


### DetSimFactory

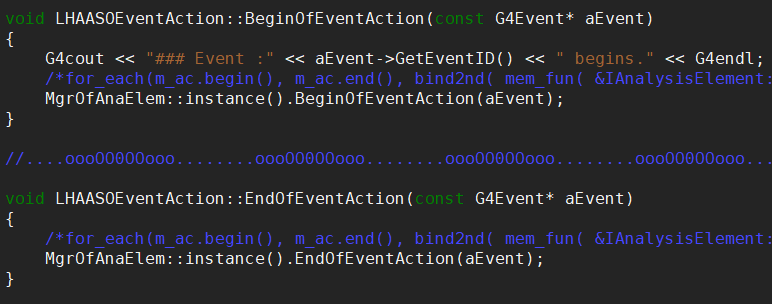
DetSimFactory is a user-end service to set up and organize all the Geant4 related classes. Usually users need to notice that user actions must correspond to the definition: G4VUserDetectorConstruction, G4VUserPhysicsLis and G4VUserPrimaryGeneratorAction has the format of:



For the User Action, LHAASOEventAction, which has been defined in AnalysisElement, you can use as below:



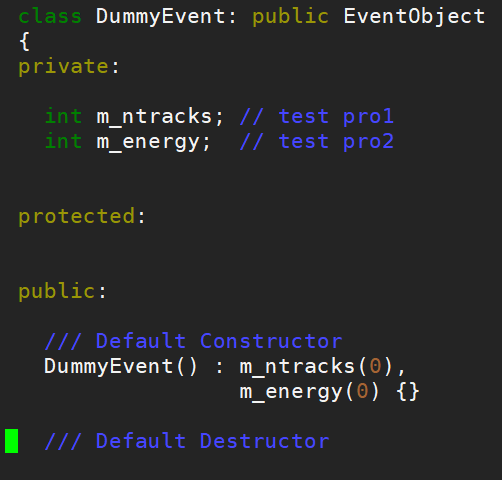
Our user actions are defined like these:



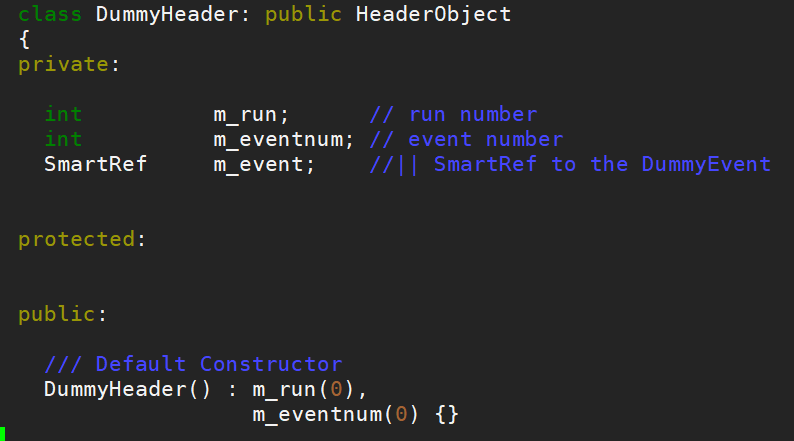
## How to define Event Data

### Event Data Header and Object

LordStar provides two base classes for event data model definition – EventObject and HeaderObject. Custom event data model should have inherited from these two classes. Here is an example of DummyEvent.h:



DummyHeader.h:



DummyEvent contains the real data, while DummyHeader has a reference to DummyEvent, only containing meta data.

In the directory “src” which contains DummyEvent.cc and DummyHeader.cc, you need to create files DummyEventLinkDef.h and DummyHeaderLinkDef.h if you want ROOT dictionary to be produced when compile. A statement in cmt/requirement is also needed: apply\_pattern cint\_dictionary files=../Event/\*.h

In src, you also need to add RawEventEDMDef.cc:

#include "EDMUtil/BookEDM.h"

LHAASO\_BOOK\_EDM(LHAASO::RawHeader, LHAASO::RawEvent, 777, /Event/RawEvent);

A standard and easy tool, XOD, is provided for defining Event and Header automatically.

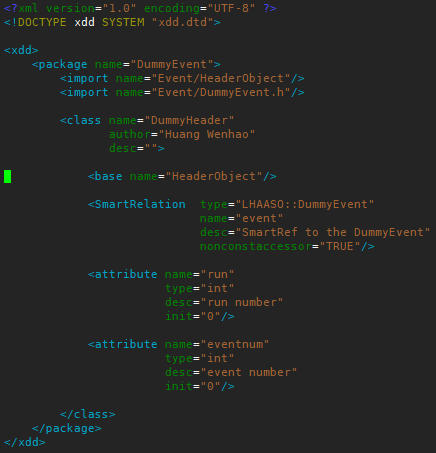
### XOD Tool

XOD tool is to generate cpp codes from xml files, and users only need to define xml files in a certain format. We already define the pattern of cmt about XOD. Please follow the steps to define a dummy event package.

1. Create a cmt package using command “ cmt create ”. If you forget it, just refer to “ How to use CMT ”). Your package should have two sub-directories: “cmt” and “src”.
2. Create a sub-directory named “xml”, in which xml files should be put in.
3. Create xml files in “xml” directory. There are two files need: header file and event file. Here we create “DummyHeader.xml” and “DummyEvent.xml”.

Below are the examples of the xml definitions, then we will explain the format.

DummyHeader:



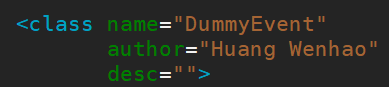
DummyEvent:



For most users, there is no need to concern. Here we just explain the xml description to cpp description.

“<import />”: Means “#include” in cpp. For example, “<import name="Event/EventObject"/>” in xml means “#include "Event/EventObject.h"” in cpp.

“<class/>”: Definition of class. In the example,

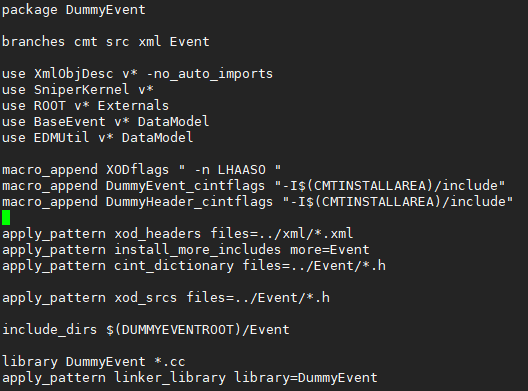


means the class name we defined in cpp.

“<base/>”: The class which new class inherit from.

“<attribute/>”: Members of the class.

“<SmartRelation/>”: Only needed in header xml file. Define reference to event object class.

1. Write requirement file in “cmt” directory. Here give an example of requirement. file: 

We have defined the pattern “xod\_headers” and “xod\_srcs” in framework, so you just need to use this pattern like above. “xod\_headers” means xml files while “xod\_src” means cpp files.

There’s another pattern you need pay attention, “cint\_dictionary”. As both header class and object class inherit from TObject, so we must generate dictionaries using “rootcint” or “rootcling”. Here we define pattern “cint\_dictionary” and you just tell it the cpp files.

### EDM Book

We also need another cpp file used to book event data model so that you can have access to it.

Usually, we define “XXXEDMDef.cc”. Here gives the example of “DummyEDMDef.cc”:



In the macro, the four parameters are header name, object name, priority and storage path.

### Config and make

After that, you can refer to “How to use CMT” to configure and make this package.

A compiled package should be like this:



Then you can use your event data in your algorithm.

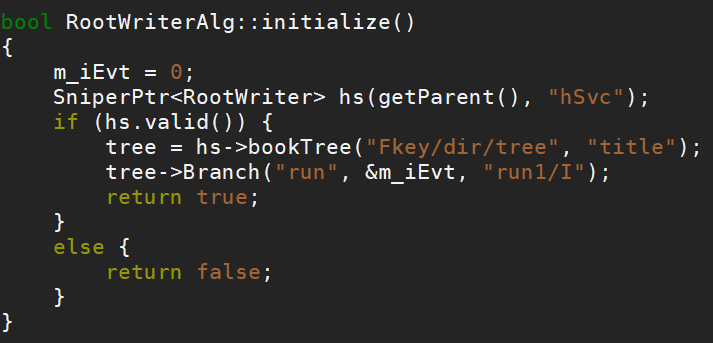
## How to write out trees and Ntuples

RootWriter is a service to write trees，Ntuples or histograms into specified root file. Services should be called by algorithms.There is an example how to fill a tree by RootWriter in /offline/Examples/IOTestAlg.

In your algorithm: In requirement file, add “ use RootWriter v\* SniperSvc”

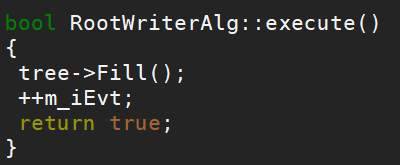
In src file add: “include RootWriter/RootWriter.h”

the service should be accessed as following:

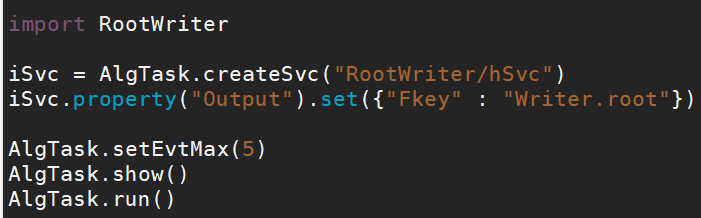


Notice that the method “bookTree” should be executed only once in initialization().

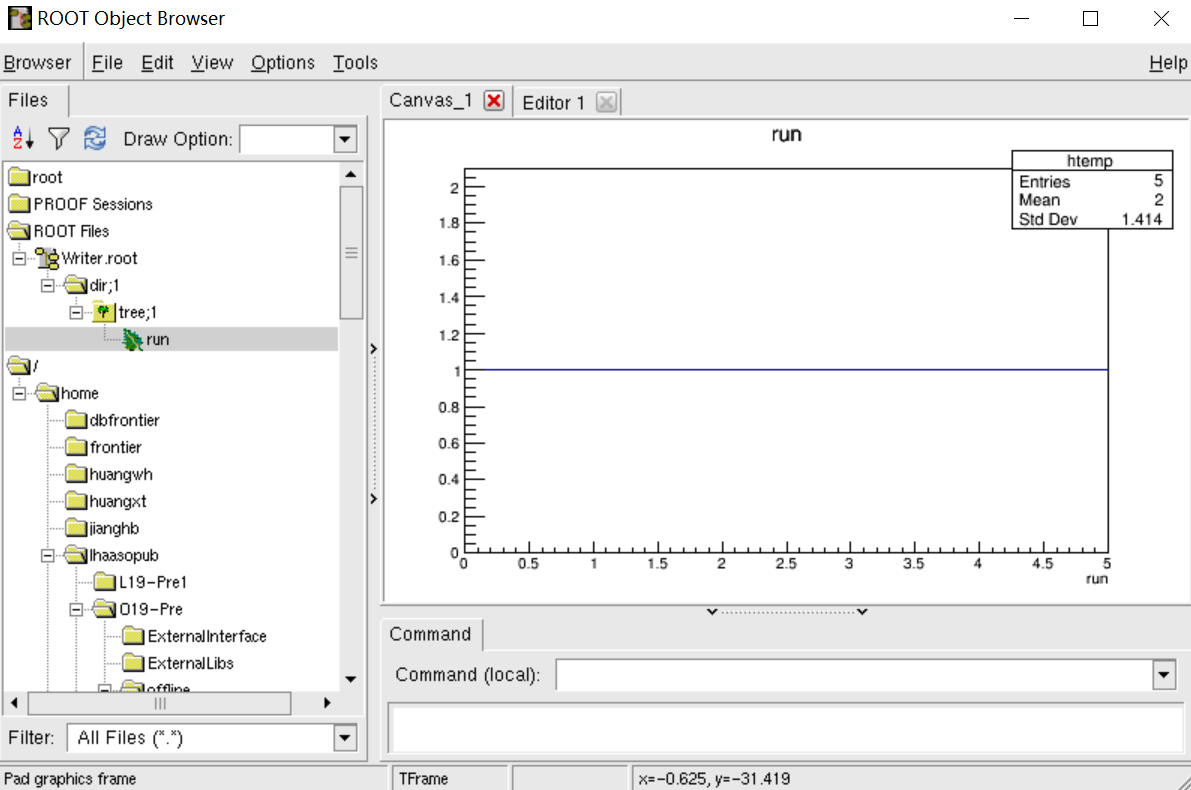
Here we fill value in execute() by looping events.



And in python script, we set the key and name of the file:



Finally, we get a file named “Writer.root” and it appears:



To save ntuple, similar with TTree, Just change the “bookTree()” function to “bookNtuple()”：

In class

TNtuple\* ntuple;

In initialize():

ntuple=hs->bookNtuple(“Fkey/dir/nt1/”,”reconstructed ntuple”, "Nevent:mjd:NhitE");

In execute():

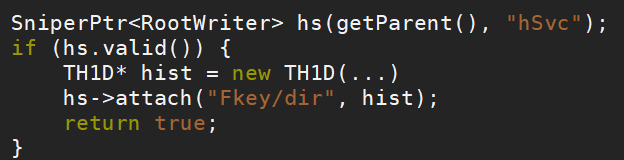
Double ntu[3];

ntu[0]=xxxx; ntu[1]=xxx; ntu[2]=xxxx;

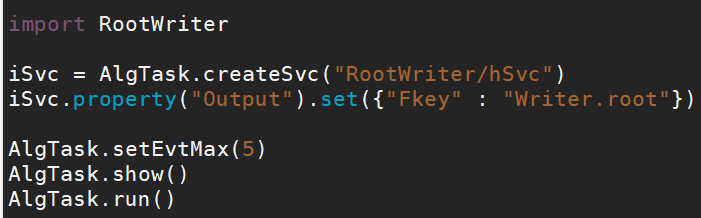
ntuple->Fill(ntu);

## How to write out Histograms:

“RootWriter” is also used to define histograms. It's similar to TTree and TNtuple:



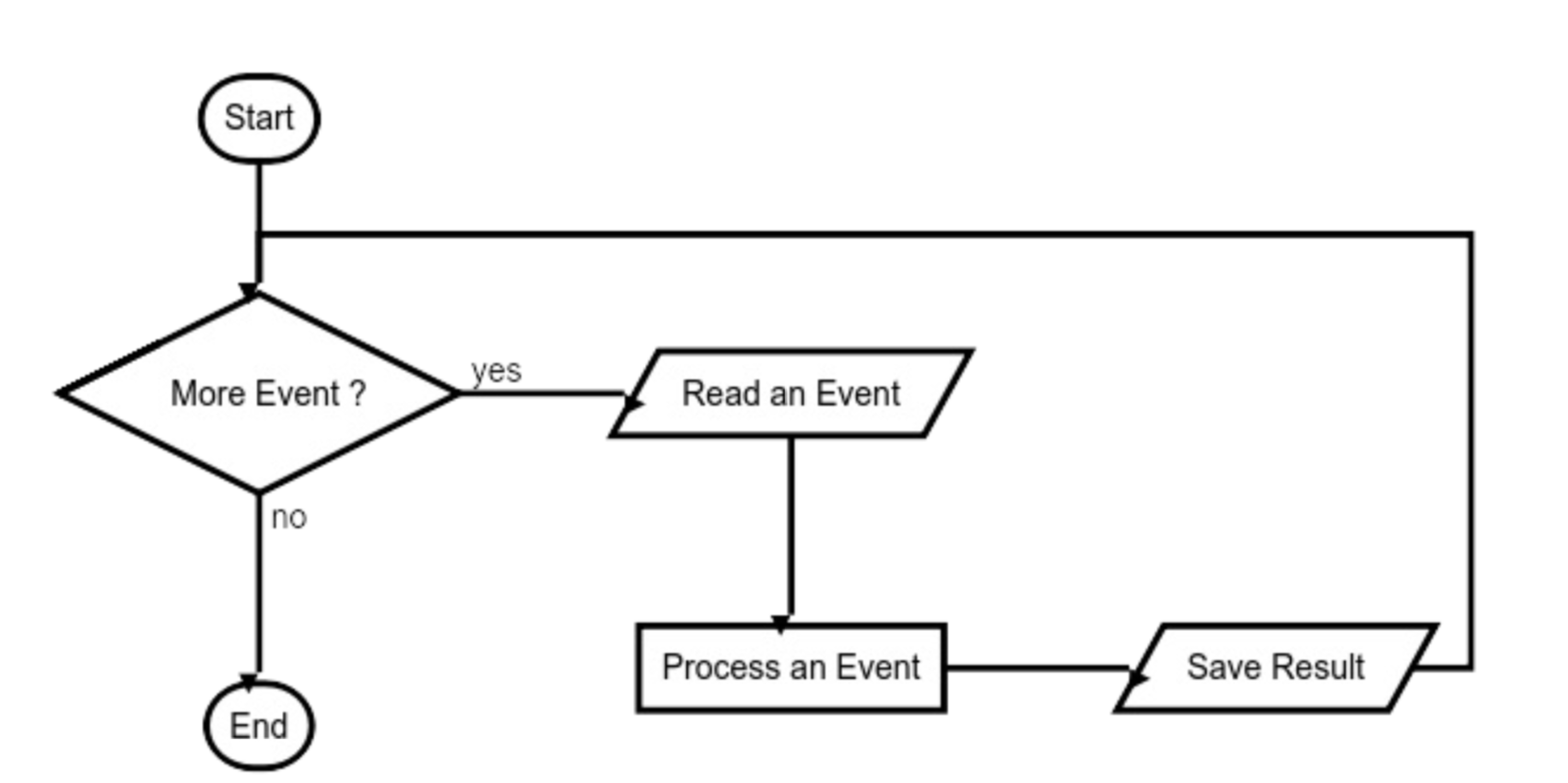
Then in python script, we set the key and name of the file:



# Main Components

## Data processing procedure

LHAASO has the similar offline data processing procedure like other high energy physics experiments, such as ATLAS, BESTIII and so on.



The typical procedure is as illustrated above: read in the event data, process the event via launching some subroutines, and then write out the result. These subroutines are implemented with Algorithms, Services and Tools.

## Algorithm

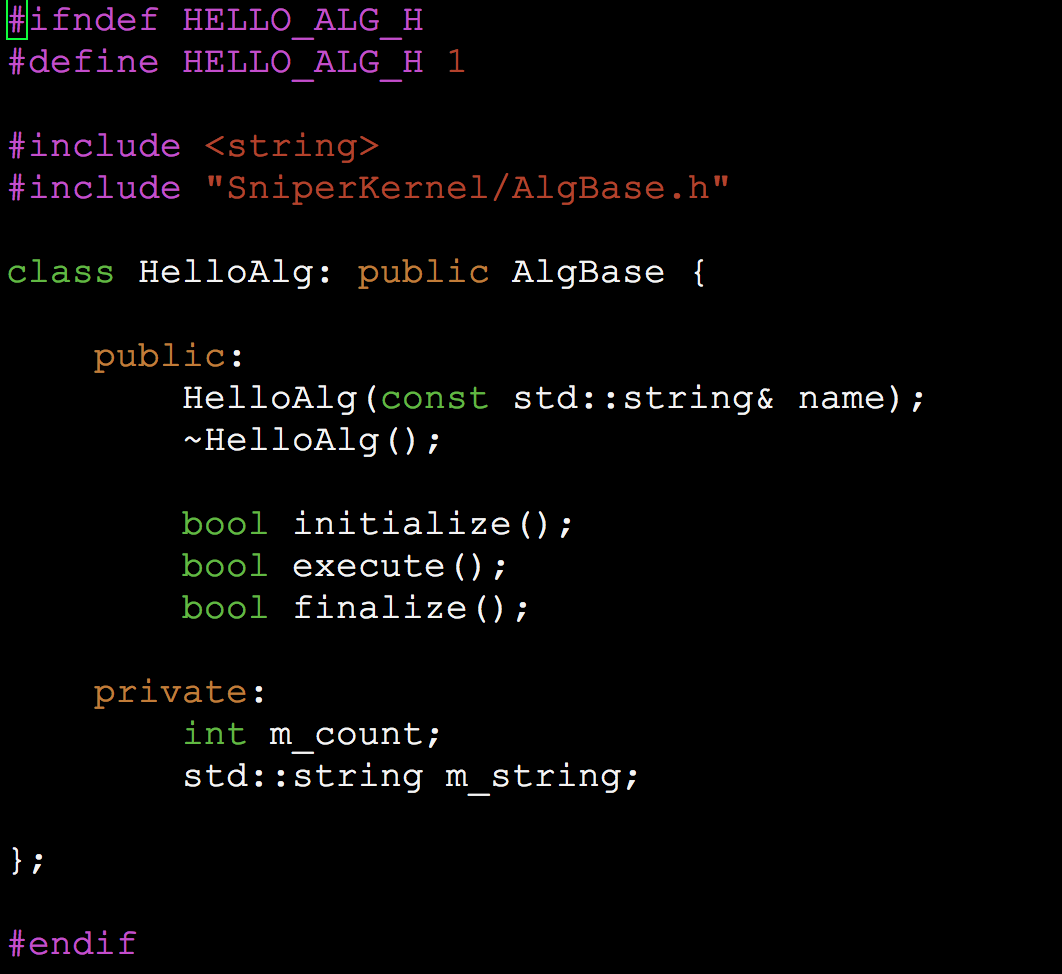
### What’s the Algorithm?

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

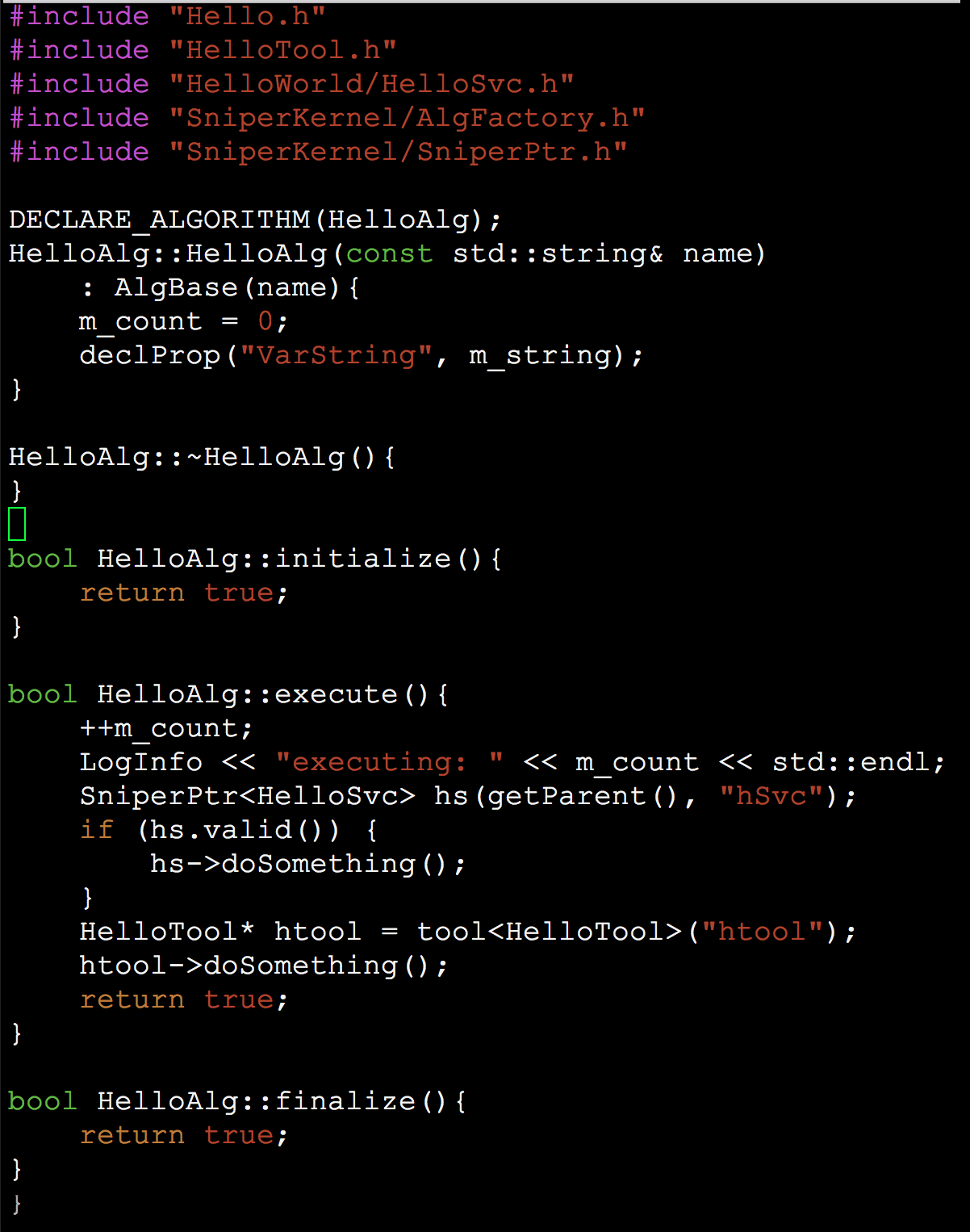
### How to create an Algorithm

The steps below are used to create a new Algorithm once you have known how to create a new cmt package.

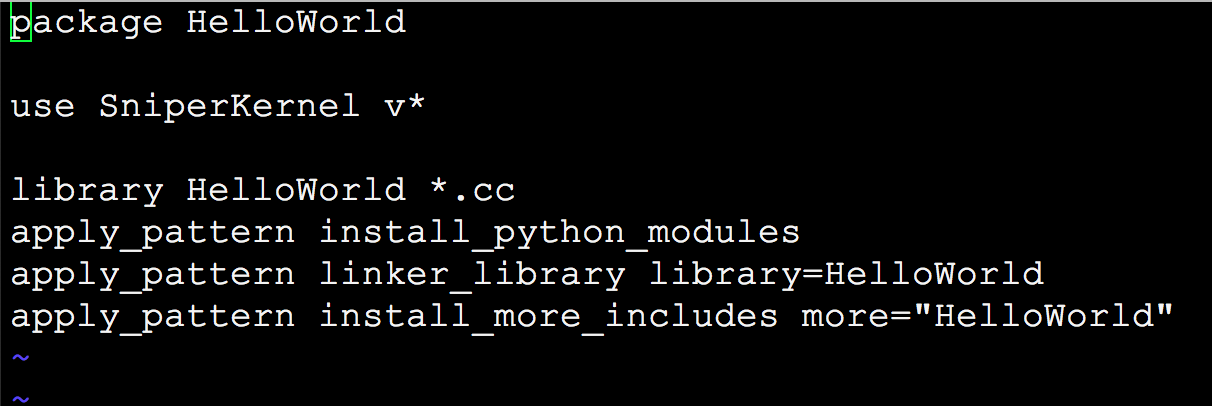
* A new algorithm must inherit from the base class, AlgBase, which is provided by the framework.
* The constructor must take a parameter of the std::string type
* three member functions must be implemented
* bool initialize() : called once per Task (at the beginning of a Task)
* bool execute() : called once per Event
* bool finalize() : called once per Task (at the end of Task)
* The header file of the HelloAlg is illustrated as below:



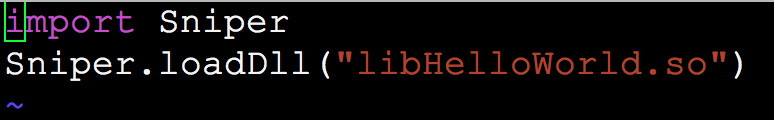
* The macro “DECLARE\_ALGORITHM”must be used in the source file to declare this new algorithm



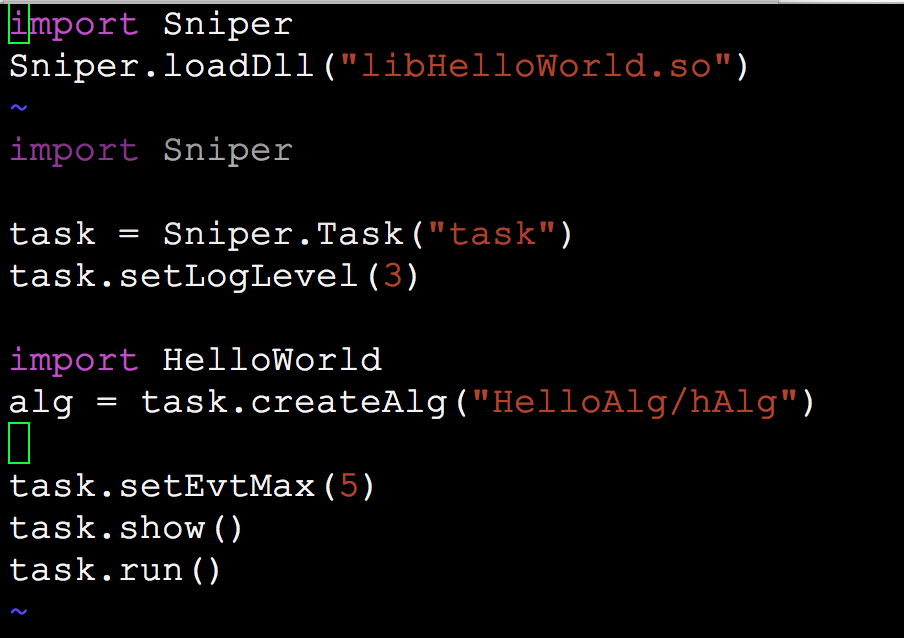
* modify the file in the HelloWorld/cmt directory



* Line 3 is used for compiling \*.cc file in the src directory of this package.
* Line 4 is to copy the \_\_init\_\_.py of the python directory into the common python directory of this project: InstallArea/python.
* Line 5 is used to link the \*.o file into a share library with the name libHelloWorld.so.
* Line 6 is to copy the header file (\*.h) in the HelloWorld directory into the common include directory of this project: InstallArea/include.
* create the “python/HelloWorld” sub-directory of this package and add the “\_\_init\_\_.py”file in order to automatically load this packages shared library when using “import HewlloWorld”. The contents is as below:



* Write python script file to run this algorithm.



## Service

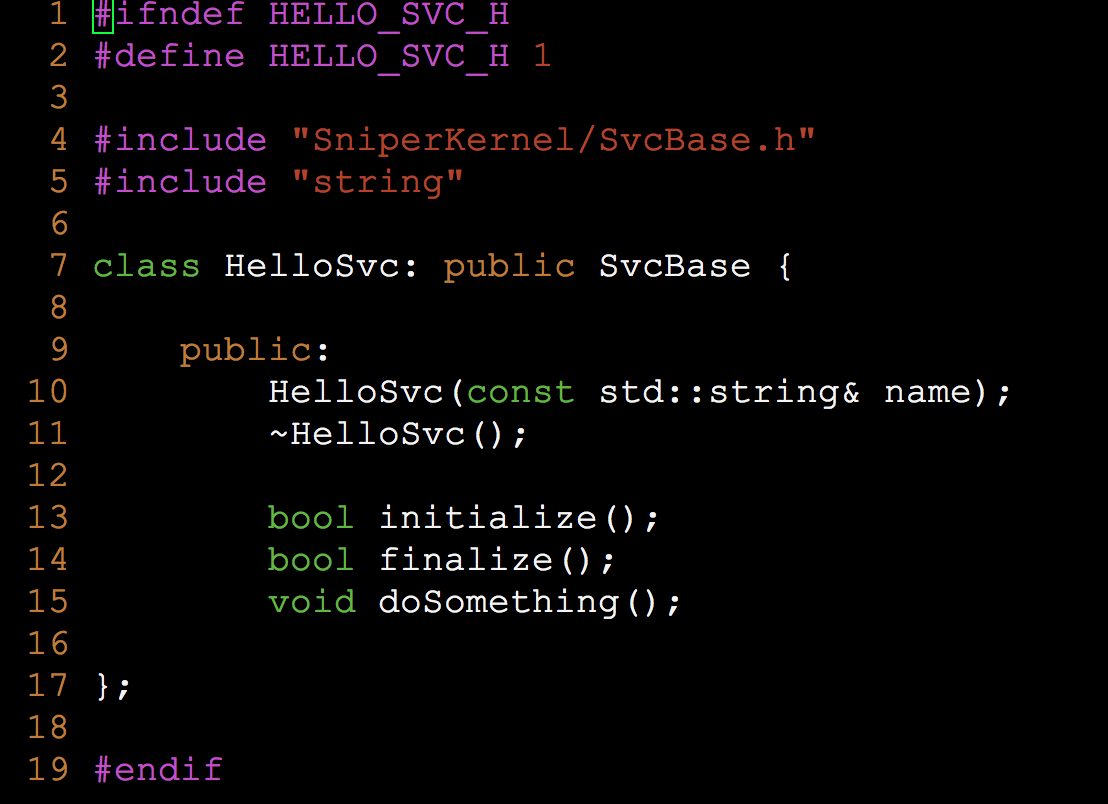
### What’s the Service?

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

### How to create a Service.

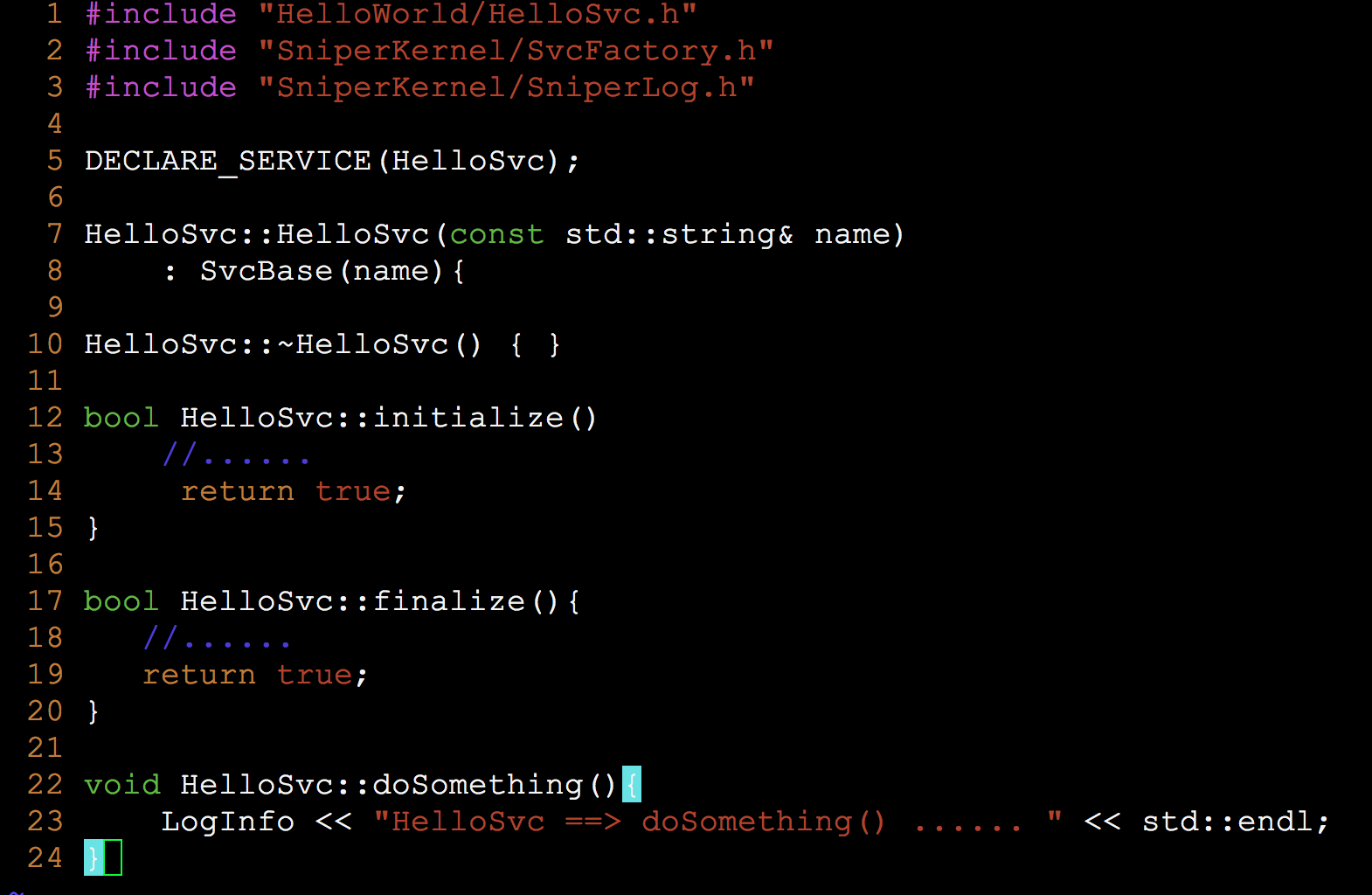
The steps below are used to create a new Service.

* A new service must inherit from the base class, SvcBase, which is provided by the framework.
* The constructor must take a parameter of the std::string type
* Two member functions must be implemented
* bool initialize() : called once per Task (at the beginning of a Task)
* bool finalize() : called once per Task (at the end of Task)
* The header file of the HelloSvc in Examples/HelloWorld package is illustrated as below:

****

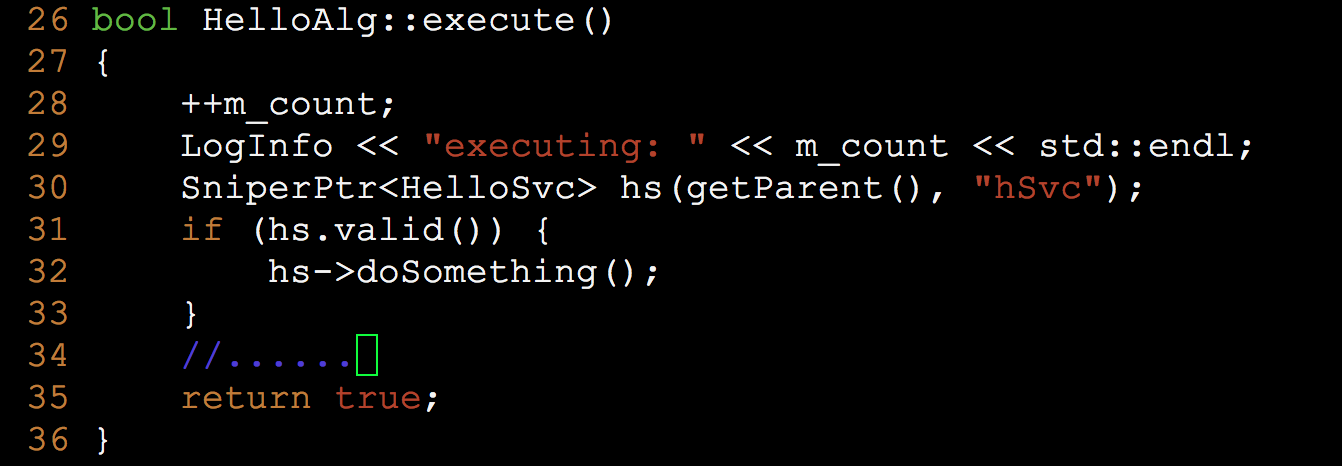
The function, doSomthing() in the line 15 is an optional function, users can add more this kind of functions according to user’s situation.

* The macro “DECLARE\_SERVICE” must be used in the source file to declare this new service as shown in the 5th line below.



### How to use a Service.

The service can be used in the algorithms, services and tools. Below shows how to use a service in the HelloAlg algorithm:



the line 30 shows how to get access to the HelloSvc.

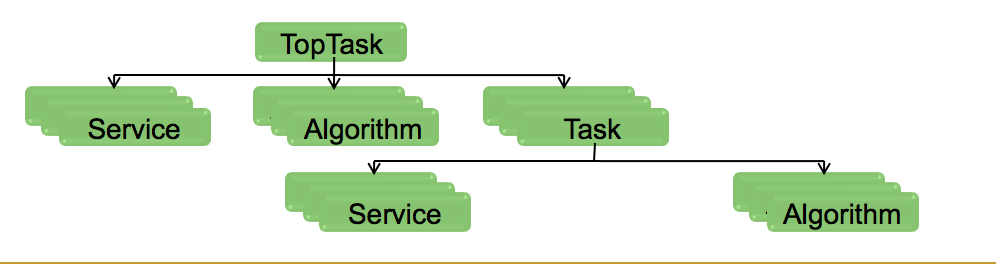
the line 31 is to check if the HelloSvc is valid or not, if yes, the line 32th is to call the HelloSvc’s member function, doSomthing().

## Task

### What’s the Task?

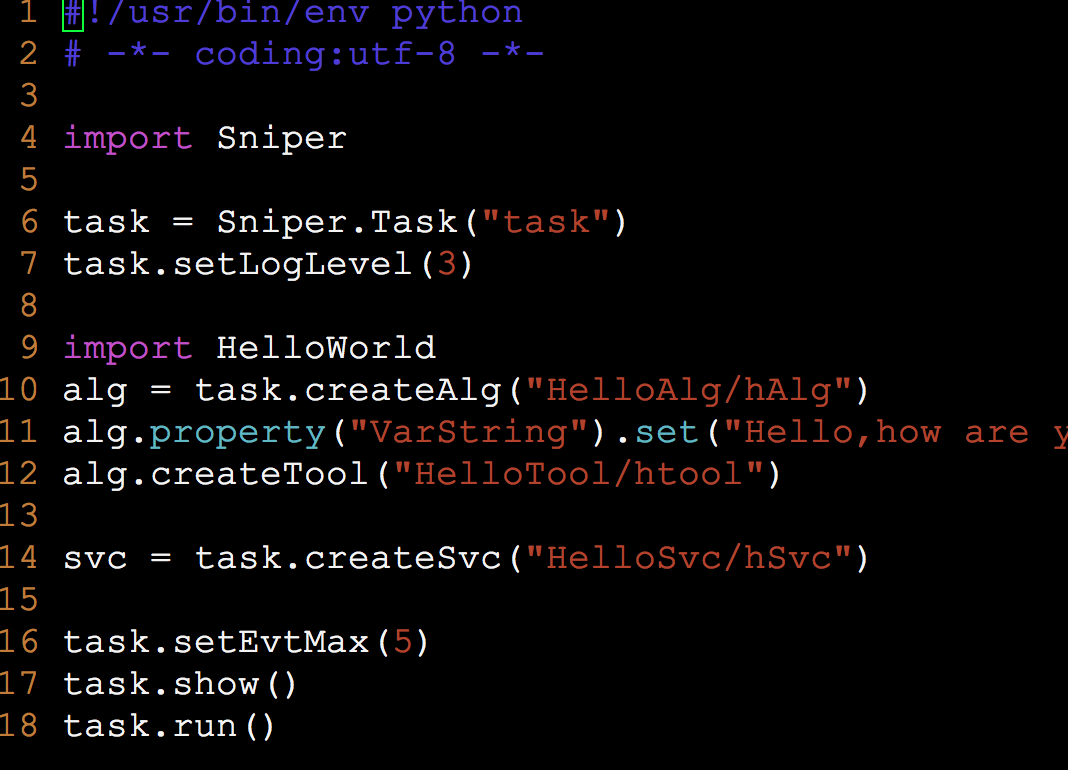
Task is a lightweight application manager, it is a controller for the event looping and the entrance of a job. As illustrated in the figure below, Task consists of algorithms, service, and sub-tasks, it controls the execution of all algorithms belonging to it according the order of the algorithms when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

Task has its own data store management and Input/Output management, which will be covered later.



### How to configure a Task.

Python script is used to configure a task as below:



the line 6: create a task with the name “task”.

the line 7: set the message output level as 3 (LogInfo)

the line 10: task creates a new algorithm, hAlg.

the line 12: task creates a new too, htool.

the line 14: task creates a new sevice, hSvc.

the line 16: set the number of events to be processed.

the line 17: output the configuration of the task when running

the line 18: starts to run the algorithms in this task.

## Tool

### What’s the Tool?

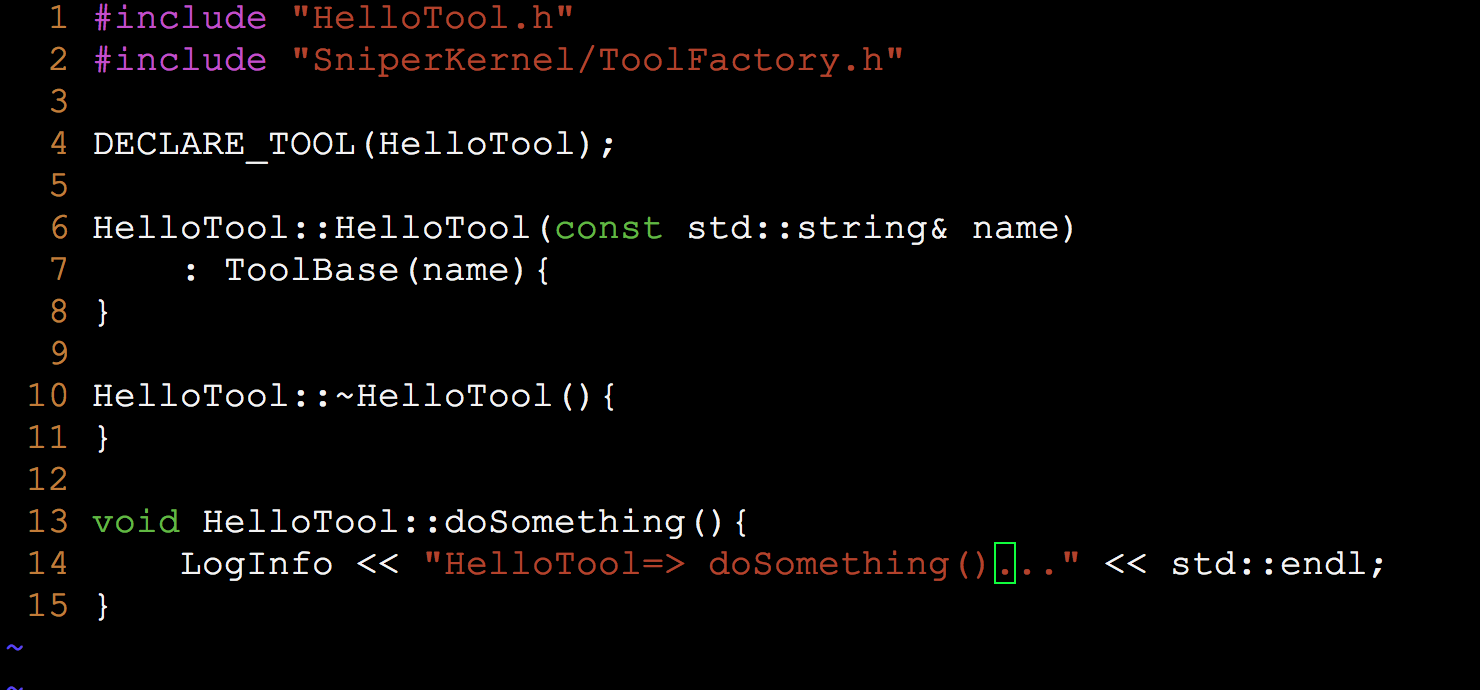
the tool is similar with algorithm and service, it is also a dynamically loadable element, but it belongs to an specific algorithm and helps the algorithm to organize code more flexibly.

### How to create a Tool

* A new tool must inherit from the base class, ToolBase, which is provided by the framework.
* The constructor must take a parameter of the std::string type

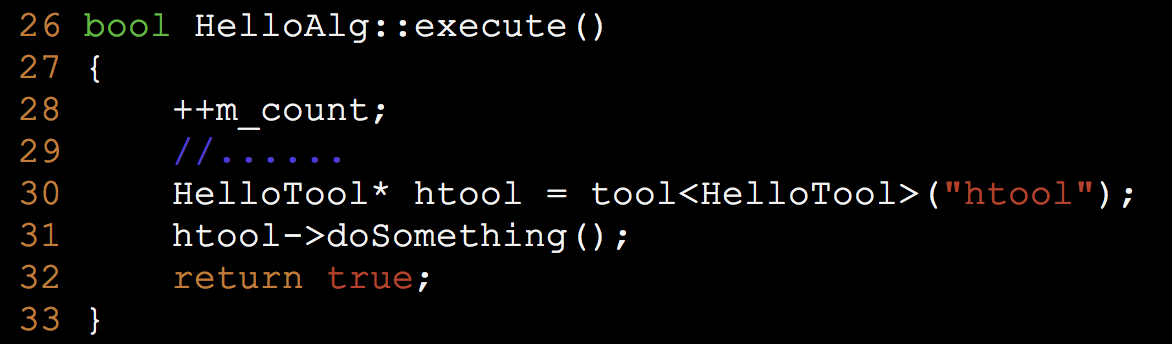


* The macro “DECLARE\_TOOL” must be used in the source file to declare this new service as shown in the 4th line below.
* all member functions are defined by users and these functions can be called freely by users in the algorithms.



### How to use a Tool

One algorithm can have one or more tools, a tool can be accessed via its name.



In python code, you need add following:

Import HelloWorld

Alg.createTool(“HelloTool/name”)

## Property

### What’s the Property?

Property is a configurable variable. Users can customize its value during the job configuration, without code modification and re-compiling.

### How to use Properties

All algorithms, services, tools and tasks can have their own properties by declaring a variable as a property in their constructor functions as below:



for example, the variable, m\_str, has been declared as the property, MyString. then you can set or re-set its value via the MyString in the python scripts



Now we support the following types to be declared as properties

* scalar: C++ build in types and std::string
* std::vector with scalar element type
* std::map with scalar key type and scalar value type

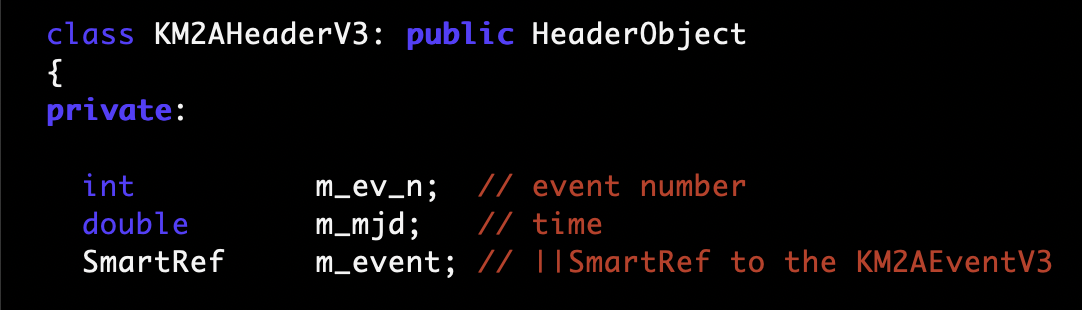
# Data Model

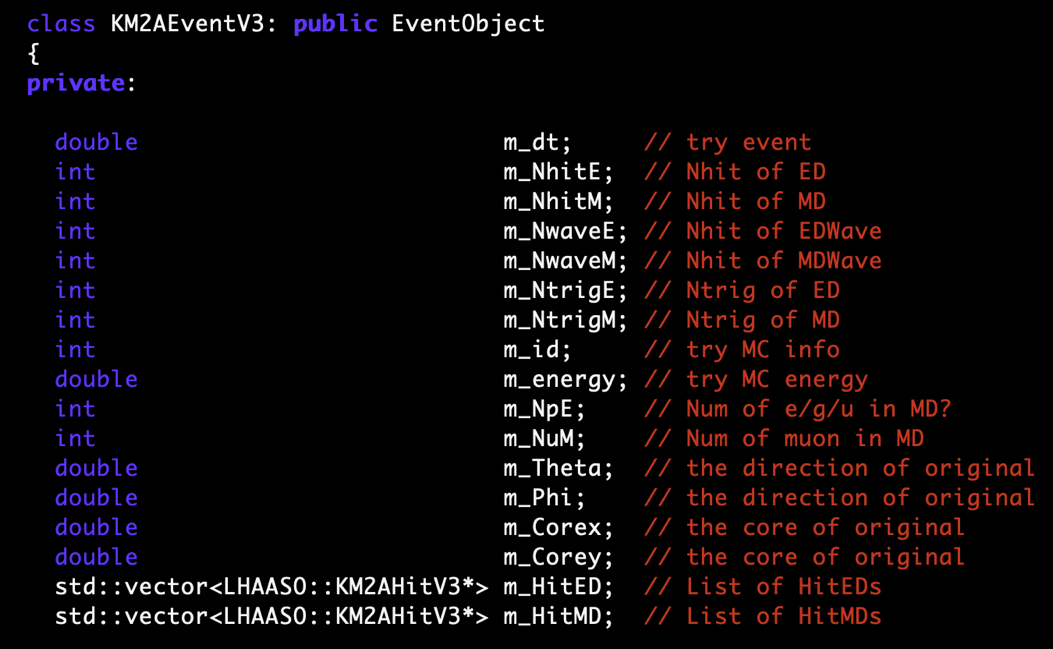
### KM2AEventV3

In offline/DataModel/KM2AEventV3 package, we provide the third version of KM2AEvent.

* KM2AHeaderV3 and KM2AEventV3

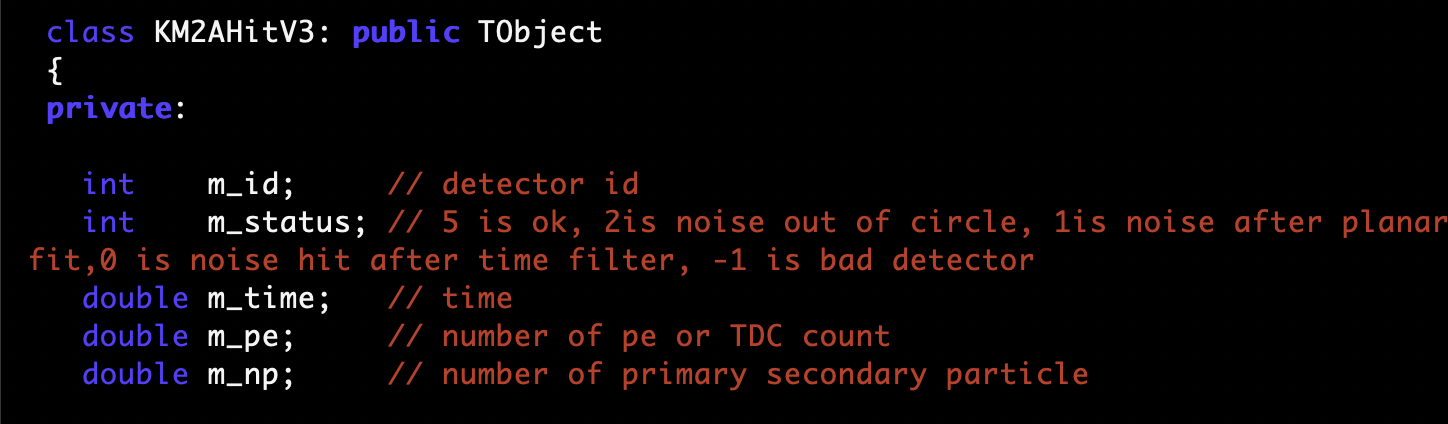
We define a unified EDM for MCtruth of geant4 simulation, including two main classes named KM2AHeaderV3 and KM2AEventV3.





* KM2AHitV3

KM2AHitV3 inherited from Tobject directly.



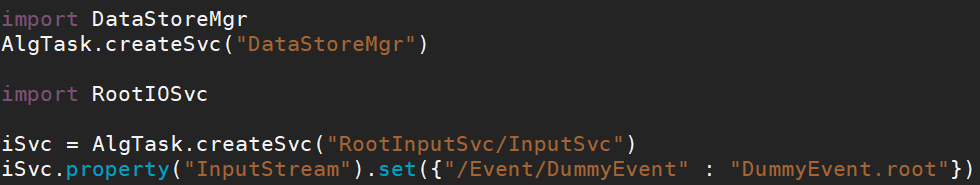
# Input/Ouput System

As our event data is defined in a standard way (Event Object and Event Header), we provide a standard system to read and write event data. It is Root Input/Output

System.

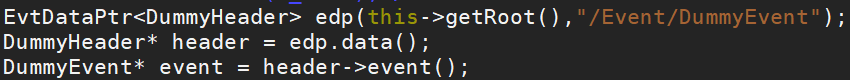
### Input system

For event data in root file, we can read it using “RootInputSvc”.



In this script, “/Event/DummyEvent” stands for the path of the data and “DummyEvent.root” is the name of root file. The path is used to distinguish event in memory so that the framework knows how to find it.

Then we can get the event data in memory with codes like this:

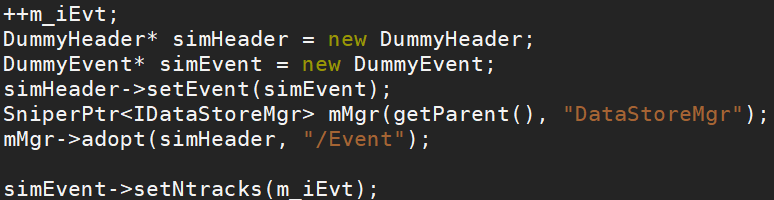


Notice that the path should be same, or the algorithm can not access it.

### Output System

Similar to the input system, output system can write event data to root file with “RootOutputSvc”.

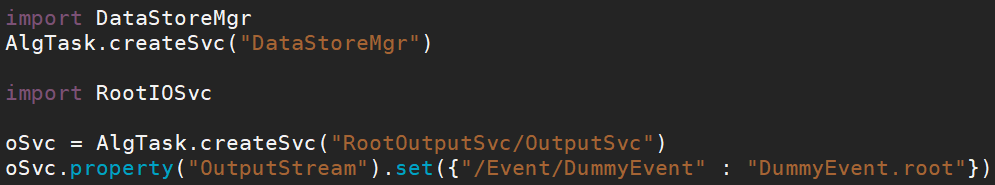
Before writing out, we should create event data in our algorithm:



“SniperPtr<IDataStoreMgr>” means smart pointer of sniper, which can handle memory automatically. Here we use it to handle out event data.

“adopt()” means we pass out event data to “DataStore”(memory management system of sniper). Notice that the first parameter of “adopt()” is the pointer of event header and the second parameter is the path (about path can refer to input system) of event data.

Then we can write out python script:



Similarly, the “/Event/DummyEvent” is the path and “DummyEvent.root” is the name of root file.

To save data of more than one data model, use the syntax following:

oSvc.property("OutputStream").set({"/Event/LHEventTmp":"test1.root","/Event/LHRecEvent" : "test1.root"})

If want to save part of the variables in the data model other than all the variables, add “//!” after the variables definition which you don’t want to save in the data model class. This is a feature of ROOT.

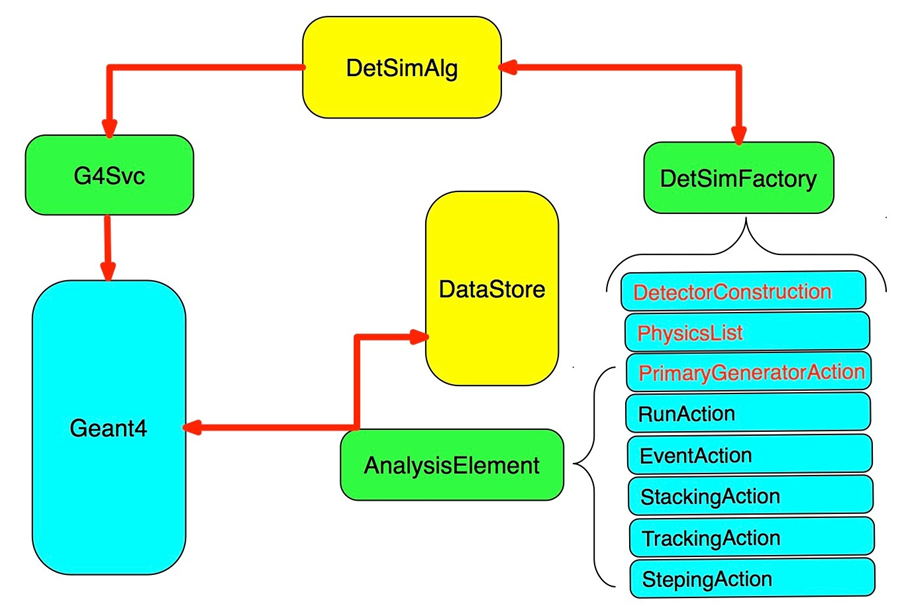
# Generators

# Simulation

* LodeStar simulation
* Detector Simulation

LodeStar manages detector simulation with a Task，which consists of Algorithms and services:

* A dedicated algorithm (DetSimAlg) for all sub-detectors simulation
* A dedicated service (G4Svc) for launching Geant4 within LodeStar
* A user-end service(DetSimFactory) to set up and organize all the Geant4 related classes
* A user-end service(AnalysisElement) for access to G4Event to create EDM



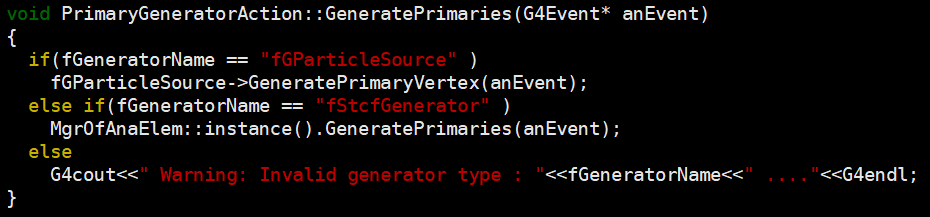
More details see in [HOW TO DEVELOP DETECTOR SIMULATION](#_How_to_develop).

* DetectorConstruction and PhysicsList

These two classes can be copied to LodeStar DetSim from geant4 code without any change.

* PrimaryGeneratorAction

This class can also be copied to LodeStar DetSim without any change. But if you want to use some Generator, like Babayaga, some changes need to be made.



Add below code,

*MgrOfAnaElem::Instance().GeneratePrimaries(anEvent);*

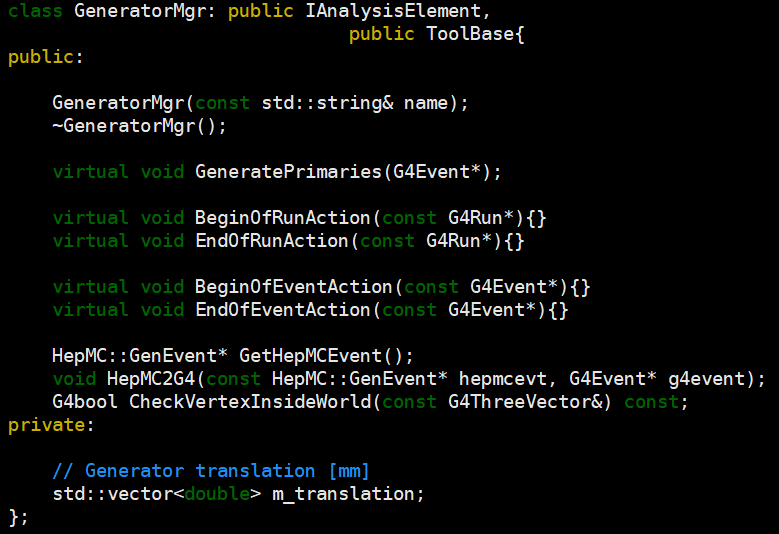
in PrimaryGeneratorAction::GeneratePrimaries(G4Event\* anEvent).

* DetSimFactory

A class inherit from DetSimFactory should be created to set up and organize all the Geant4 related classes.

* AnalysisElement

A DetSimAlg can contains many different AnaMgrs inherit from AnalysisElement. These AnaMgrs implement different functions. For example, GeneratorMgr for generating event, GeoAnaMgr for storing geometry in root file. Other sub-detectors can also have their own AnaMgr to get simulation truth information.



* Python script

A python script is used to configure task and run algorithm.

# Calibration

# Data for calibration and analysis

In the data preparation and physics analysis, some condition data or calibration data are used at many points, such as detector position, data quality, single particle peak. These data are sealed in service package for user. In principle the user will not directly look at the data, but the service package only. The service package maintainer will take care of the reading of data into service package.

Classify these data into two types: small data, not changing with time, such as detector position; big data, which are huge and change with time, such as single particle peak.

To centrally store the small data, service maintainer should ask the LodeStar manager to put in the common, lodestar-release independent directory: /afs/[ihep.ac.cn/soft/LHAASO/CalibData](http://ihep.ac.cn/soft/LHAASO/CalibData) . The service will read these data in initialize().

To store the big data, we use data base in a separate server. To read these data, a example is:

/afs/[ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/L19-Pre1\_v1r1/offline/Database/DatabaseCalibSvc](http://ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/L19-Pre1_v1r1/offline/Database/DatabaseCalibSvc). The service will not read in all data in the initialize(), but when the uses request specific data, such as single particle peak of an ED in a specific time duration, the service will fetch the piece of data from server over internet. The piece of data will be buffered in the time of job, so that the services don’t have to fetch the same data repeatedly over internet.

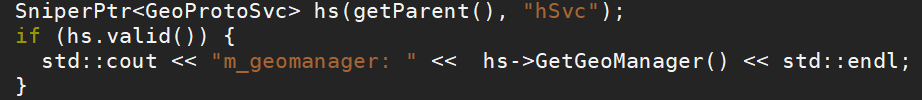
The data stored in either cases should be tagged with “version”, a version of service package will connected to specific version of data. When user use a specific version of service, the data he/she is using is fixed.

Tip:when open a mac terminal, .bashrc is not run. Put your command into .bash\_profile.

# Reconstruction

## How to get Geometry information in root file

Now we define a service “GeoProtoSvc” to get geometry information in root file. In our algorithm, we can use “GeoProtoSvc” like this:



Function “GetGeoManager()” is a interface to get the pointer of TGeoManager.

The name of the file and path should be set in python script.

# Physics Analysis Tools

# Appendix

## Installation

* Set GCC Environments

Please refer to “How to setup Environments” for GCC environment setting.

* Install LodeStar with script
* Get the install script from svn:

$ svn export <http://svn.lhaaso.ihep.ac.cn/LodeStar/installation/trunk/setup-trunk.sh>

* Run this script, it will install the whole LodeStar in your current path.

(So remember to choose a correct path)

$ source setup-trunk.sh

* Add *source setgcc494.sh* into the setup.sh in LodeStar top directory, so users can setup GCC environment and LodeStar environment at the same time by source setup.sh only.



## 开发者使用SVN

开发中能够编译成功的版本号,写入tag，给一个版本号

可以对外使用的版本号：vxx.xx

每一次大的修改，需要其他使用者修改自己程序的，主版本号+1

每一个小的修改，不需要其他使用者修改自己程序的，根据需要次级版本号+1

## Coding Conventions

1. **Package Naming Conventions**

* All packages begin with a word or an abbreviated word which should be meaningful and with the first letter capital, for example, DetectorSimulation or DetSim.
* Service package should end with Svc, such as DetPosSvc
* Tool package should end with Tool, such as DetPosTool
* Encourage a package contains only one tool/service/algorithm. The package name is same to the tool/service/algorithm name.
* Each CMT package includes sub-directories: cmt, PackageName, src, share, python.
  + cmt: automatically generated and used for“requirements”file
  + PackageName: header files of each class: \*.h
  + src: implementation files: \*.cc
  + python: default python configuration: \*.py
  + share: python scripts for running and testing
  + In Package directory： edit a file “HowToUse”，record the change between versions and procedure how to use it。

1. **Coding Conventions**

* Data members begin with m\_: m\_energy
* Static variables begin with s\_: s\_count
* Constants begin with c\_: c\_gain
* Global variables begin with g\_: g\_env
* Function members begin with the capital letter: SetEnergy()
* Functions should correlate with variables: m\_energy /SetEenergy()/ Energy()
* Locals and parameters begin with a lower case: nbytes
* Getters and setters begin with get and set: SetLast(), GetFirst()
* One header file usually for one class, except internal class definitions
* Use macro predefinition to avoid duplication

#ifndef LASODetSim\_H

#define LASODetSim\_H

…

#endif

* Inline functions should be defined in the header files
* Including external header files: #include < <package>/header.h >
* Including local header files: #include “header.h”
* All data members should be initialized in the construction function of the class.

1. **Comments convention**

* Line comments use //
* Block comments use /\* …\*/
* Each header file should have comments include the class definition, developer and e-mail
* Each function are encouraged to be written not more than one screen.
* Each function member and data member should have line comments

## Package Relations

1. RawIO service将LHAASO的二进制原始数据文件读入放进一个event一个event地放进RawEvent中，EventCalibration对数据进行清理和刻度，存成LHCaliEvent，EventReconstruction继续使用内存中的LHCaliEvent，重建后存在LHRecEvent里
2. 获取Svc或者Tool指针如果出错，应该立刻停止event的循环，如何做？

In your code:

#include “SniperKernel/Task.h”

this->getParent()->stop() //get the task pointer and stop the task.

1. Svc 指针定义在类成员函数中，以供所有成员函数使用：

在类定义中，定义成员变量DetGeometrySvc\* \*geom

在initialize（）中初始化，

SniperPtr<DetGeometrySvc> temp(getParent(), "DetGeo");

geom=temp.data();

Tip: when get the pointer to any tools/service, it has to be done in initialize() or any other subroutines other than constructor functions.

Tip: to link missed Root library, add in requirement file:

macro\_append ROOT\_linkopts " -lMinuit"

Tip: In bool initialize() of service package. If it return false, then the whole task will be terminated. 这是一个好的方法避免初始化不正确，程序继续运行。

Tip: mjd is the time shift from 1970.01.01(40587), 它的整数部分精确到天，小数部分精确到天以下，决定于小数的位数。 ns是每天中的时间，精确到纳秒。 两者相加绝对时间精确到纳秒。

Tip：所有的刻度小型数据存放在lodeStar release 外的公共固定区域/afs/[ihep.ac.cn/soft/LHAASO/CalibData](http://ihep.ac.cn/soft/LHAASO/CalibData)，读取时由service package解析。 所有的刻度大型数据存放在山大数据库，读取时，通过网络获取，获取例子在/afs/[ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/L19-Pre1\_v1r1/offline/Database/DatabaseCalibSvc](http://ihep.ac.cn/soft/LHAASO/LodeStar-SLC6/Pre-Release/L19-Pre1_v1r1/offline/Database/DatabaseCalibSvc)

Tip:

大家为合作组开发的pacakge统一放到一个SVN中，管理员给每一个开发者分配相应的具有写权限的svn目录。

任务：写一个macro帮助大家将需要的环境，需要下载的pacakge，自己的工作目录一次性全建立起来。

Tip: use TClonesArray to reduce program running time.

TClonesArray\* a=TClonesArray(“TTest”,100)； 为100个TTest开辟内存空间（TTest必须是从TObject继承来的）。 这100个内存空间编号从0到99。当使用下面任何一种方法开辟一个编号大于99的空间时,甚至引用0到99以外的编号时，TClonesArray的长度会自动的增长一倍, 即200，来容纳新的编号，如果增长一倍不足，会按照定义的某种方式增长到囊括新编号为止。因此TClonesArray将不会出现溢出的现象。 没有被访问过或者new过的编号，虽然内存存在，但是TClonesArray认为那是空的Object

只有delete a； 会收回这些内存空间。否则，这100个TVectorF的内存空间将会一直存在。

a->RemoveAt(i)不会收回空间，仍然存在。因此这儿的问题是TClonesArray不具备想vector<TTest>那样的删除的功能。而且应当使用时将所有Object从0依次向后排列，中间不留空隙。

在中间过程中,可以使用以下击中方法创建TTest，和一般的new方法相比，区别是在已经定好的内存空间上创建object，而不是去用新的内存。

TTest\* c =(TTest\*)((\*a)[i]); 简单的将TObject\*转化为TTest\*即可使用

(\*a)[i]=new TTest(); 在已经开辟的内存空间上，创建object，调用构造函数

TTest\* b= (TTest\*) (a->ConstructedAt(i)); 在已经开辟的内存空间上，创建object，调用构造函数，但是第二次及以上次执行时，将不再重复调用构造函数。这是建议的开辟新object的方法。

new((\*a)[i])TTest(); 在已经开辟的内存空间上，创建object，调用构造函数

每次使用完以便给下一次使用时，都应当调用 a->Clear(); Clear不会对Object摧毁，不会收回内存。只会[SetOwner](https://root.cern.ch/doc/master/classTClonesArray.html" \l "a366d58d2a4f2102203b18a3a061f8057)([kFALSE](https://root.cern.ch/doc/master/RtypesCore_8h.html" \l "a7e568baf910535e8ffd438acb843434d));

最好调用a->Clear("C"); 因为他们将会启动你自定义的TTest::Clear(Option\_t）。在你的Clear中你可以收回TTest中定义的指针变量开辟的内存空间。而且复位一些标志位[ResetBit](https://root.cern.ch/doc/master/classTObject.html" \l "aedd58b7272292b17ae9275ac13167d93)( [kHasUUID](https://root.cern.ch/doc/master/classTObject.html" \l "af1b5d608a90d3839828e3d8fc210ef33a3f2614db9e38a6d2d956054d06fbe9ce) ); [ResetBit](https://root.cern.ch/doc/master/classTObject.html" \l "aedd58b7272292b17ae9275ac13167d93)( [kIsReferenced](https://root.cern.ch/doc/master/classTObject.html" \l "af1b5d608a90d3839828e3d8fc210ef33a67e09494490e39777a928f7d904ae1ac) ); [SetUniqueID](https://root.cern.ch/doc/master/classTObject.html" \l "ac394d57fe93e8a70d6a396be0cc6e8a0)( 0 ); 这是建议的“销毁”本Object的方法。

也可以调用delete ((TTest\*)((\*a)[i])); //删除object，调用destructor，但是空间被TClonesArray保留

或者((TTest\*)((\*a)[i]))->Delete() // clear内存空间（实际上内存在那，内存内的内容保持不变），不调用destructor，速度比clear慢。

总体来说，TClonesArray是个TTest内存的管理员。New/delete看起来执行了开辟和释放内存，但是都是在TClonesArray预留的内存中，并没有真正的开辟新内存和释放掉内存。

**[TClonesArray](https://root.cern.ch/doc/master/classTClonesArray.html" \o "An array of clone (identical) objects. )** allows you to "reuse" the same portion of memory for new/delete。 Every time the memory of the **[TClonesArray](https://root.cern.ch/doc/master/classTClonesArray.html" \o "An array of clone (identical) objects. )** has to be reused, the **[Clear()](https://root.cern.ch/doc/master/classTClonesArray.html" \l "ab7c10cc62a4805b8cef97adfd2cd4b98" \o "Clear the clones array. )**method is used.

When investigating misuse of [TClonesArray](https://root.cern.ch/doc/master/classTClonesArray.html" \o "An array of clone (identical) objects. ), please make sure of the following:

Use [Clear()](https://root.cern.ch/doc/master/classTClonesArray.html" \l "ab7c10cc62a4805b8cef97adfd2cd4b98" \o "Clear the clones array. ) or Clear("C") instead of [Delete()](https://root.cern.ch/doc/master/classTClonesArray.html" \l "a025645e1e80ea79b43a08536c763cae2" \o "Clear the clones array. ). This will improve program execution time.

[TClonesArray](https://root.cern.ch/doc/master/classTClonesArray.html" \o "An array of clone (identical) objects. ) object classes containing pointers allocate memory. To avoid causing memory leaks, special Clear("C") must be used for clearing [TClonesArray](https://root.cern.ch/doc/master/classTClonesArray.html" \o "An array of clone (identical) objects. ). When option "C" is specified, [ROOT](https://root.cern.ch/doc/master/namespaceROOT.html" \o "VSD Structures. ) automatically executes the [Clear()](https://root.cern.ch/doc/master/classTClonesArray.html" \l "ab7c10cc62a4805b8cef97adfd2cd4b98" \o "Clear the clones array. ) method (by default it is empty contained in [TObject](https://root.cern.ch/doc/master/classTObject.html" \o "Mother of all ROOT objects. )). This method must be overridden in the relevant [TClonesArray](https://root.cern.ch/doc/master/classTClonesArray.html" \o "An array of clone (identical) objects. ) object class, implementing the reset procedure for pointer objects.

If the objects are added using the placement new then the Clear must deallocate the memory.

If the objects are added using [TClonesArray::ConstructedAt](https://root.cern.ch/doc/master/classTClonesArray.html" \l "ac331a2b98b2f4d63682d3eb9c7a3e3c2" \o "Get an object at index 'idx' that is guaranteed to have been constructed. ) then the heap-based memory can stay allocated and reused as the constructor is not called for already constructed/added object.

To reduce memory fragmentation, please make sure that the TClonesArrays are not destroyed and created on every event. They must only be constructed/destructed at the beginning/end of the run.

When use XML to create class, it will create the classLinkDef.h in “src” automatically. If you create the class by yourself, for the ROOT dictionary creation, you have to create by hand the classLinkDef.h

#include "SniperKernel/SniperLog.h"

