Using Python as Configuration Language

Xia Xin
xiax@hepg.sdu.edu.cn

Shandong University

5th July 2013

Outline

Motivation

Framework

C++

- User Algorithm
- SniperMgr

CMT

Python

Summary

Outline

Motivation

Framework

C++

- User Algorithm
- SniperMgr

CMT

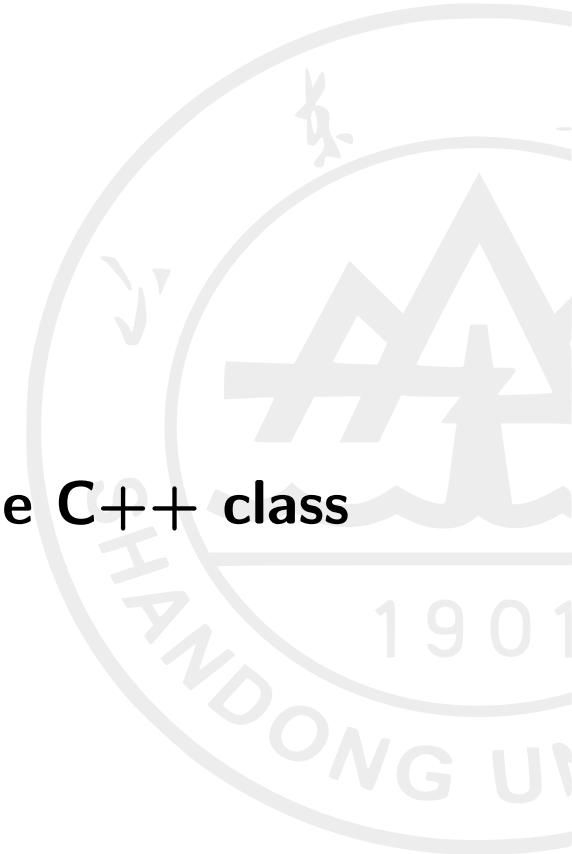
Python

Summary

Motivation

1. We need a new framework.
2. Gaudi is too complex for us.
3. Have good compatibility with C++.
4. Good user experience is also important.

Using Python with PyCintex package to handle C++ class



Outline

Motivation

Framework

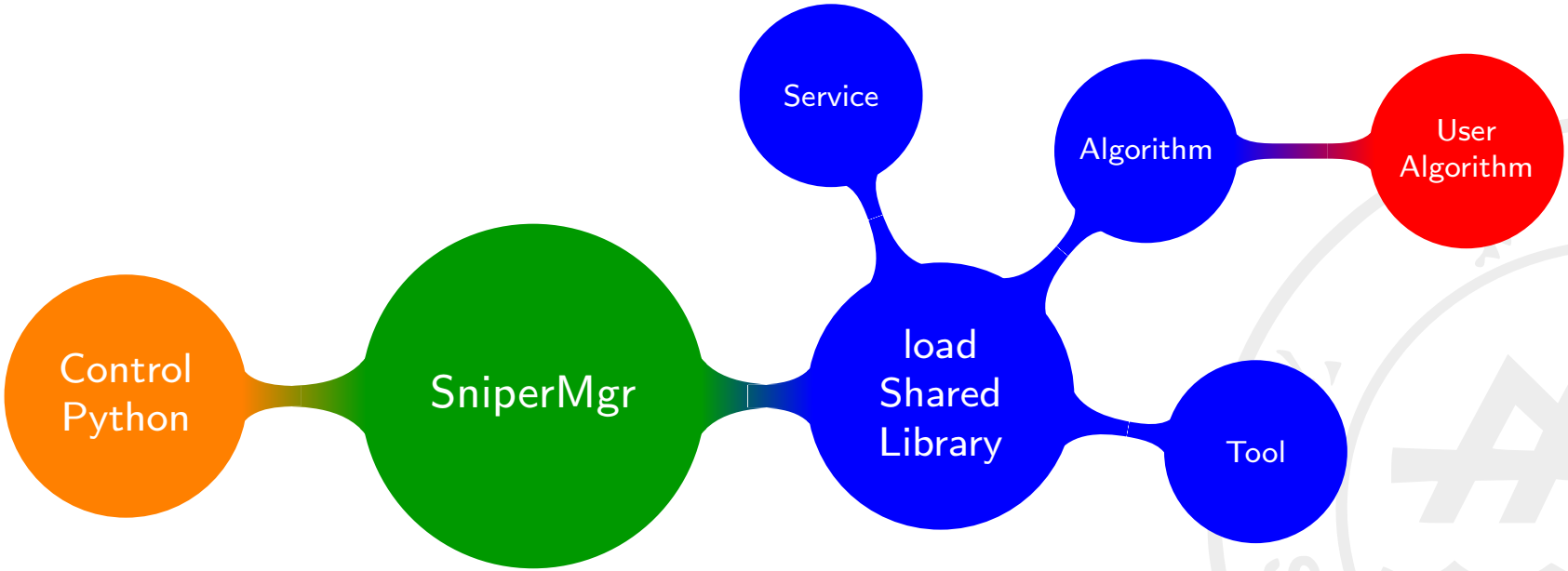
C++
- User Algorithm
- SniperMgr

CMT

Python

Summary

Framework



Control Python is mainly used for configuration.
User Algorithm needs to describe which variables to config.
SniperMgr is compiled with reflection informations, has interface to Python.

Outline

Motivation

Framework

C++
- User Algorithm
- SniperMgr

CMT

Python

Summary

Outline

Motivation

Framework

C++

- User Algorithm
- SniperMgr

CMT

Python

Summary

User Algorithm

User Algorithm is always a class loaded by the framework. We can not give a value to class member before instantiated.
Here we use global variables to receive the configurations from framework.

For user

```
1   #include "setOption.h"
2
3   using namespace std;
4   Option(int, mint)          //==========>>>
5   Option(string, mend)
6
7   MyClass::MyClass(const std::string& name)
8       : AlgBase(name)
9   {
10      m_myint = py_mint;
11      m_end   = py_mend;
12  }
```

```
1   // setOption.h
2   #define Option(type, var) \
3       char* _Typeof##var = #type; \
4       type py_##var;
```

```
1   char* _Typeofmint = "int"; int py_mint;
2   char* _Typeofmend = "string"; string py_mend;
```

Outline

Motivation

Framework

C++

- User Algorithm
- SniperMgr

CMT

Python

Summary

SniperMgr

SniperMgr loads the *so*, and then transports values into library.

1. Set a property of SniperMgr, i.e. m_evtMax, interface to python:

```
void SetEvtMax(int i) { m_evtMax = i; }
```

2. Set the value of loaded *so*, interface to python:

```
void SetOptions(std::string Class, std::string Var, std::string Value);
```

3. Get the type of variable from *so*, and translate value in python into proper type. Interface to shared library:

```
char** pType = (char**)dlsym(dl_handler, VarNameInC);

template <class TYPE>
bool SniperMgr::SetValue(void* dl_handler, string VarName, TYPE value);
```

Detials of some functions

```
1  void SniperMgr::SetOptions(string Class, string Var, string Value)
2  {
3      OPTIONS newOption;
4      newOption.ClassName = Class;
5      newOption.VarName   = Var;
6      newOption.Value     = Value;
7      Options.push_back(newOption);
8  }
9
10 template <class TYPE>
11 bool SniperMgr::SetValue(void* dl_handler, std::string VarName, TYPE value)
12 {
13     TYPE *pVar = static_cast<TYPE*>(dlsym(dl_handler, VarName.c_str()));
14     *pVar = value;
15
16     return true;
17 }
```

Make SniperMgr visiable to python

While C++ does not support the reflection well, we can not determine which functions and data members are available for a certain class. **Reflex** can add a type description on C++.

A C++ shared library can be called in third language with a simple API, such as python, if it was compiled with reflection information generated by Reflex.

```
genreflex ../app/SniperMgr.h --selection=../selection.xml
```

genreflex is a tool to parse header files and extract the reflection information, included in Reflex and also ROOT.

Generated reflex cpp file should be linked in when building the reflection library. (Following just used as example)

```
g++ -fPIC -rdynamic -O2 -shared -I$REFLEXHOME/include
SniperMgr_rflx.cpp SniperMgr.cc -o libSniperMgrDict.so
-L$REFLEXHOME/lib -lReflex
```

Outline

Motivation

Framework

C++
- User Algorithm
- SniperMgr

CMT

Python

Summary

CMT

Using CMT as the package manager, we should declare a pattern in requirements file for doing the above action.

```
1  package SniperKernel
2
3  use ReflexInterface v*
4  ...
5  apply_pattern reflex_sniper_dictionary dictionary=SniperMgr \
6          headerfiles=$(SNIPERKERNELROOT)/app/SniperMgr.h \
7          selectionfile=$(SNIPERKERNELROOT)/dict/sniper_dictionary.xml
```

This pattern just needed by the SniperMgr class. For users' and the other packages, nothing changes in CMT config.

Outline

Motivation

Framework

C++

- User Algorithm
- SniperMgr

CMT

Python

Summary

Python calls C++ library

Thus, we can access the SniperMgr library in python script.
Two choices: Python with PyCintex & PyPy with cppyy

```
1  import PyCintex
2  PyCintex.loadDictionary('libSniperMgrDict.so')
3  Sniper = PyCintex.gbl.SniperMgr("test.txt")
4
5  Sniper.SetEvtMax(15)
6  Sniper.SetOptions("MyClass", "mint", "21")
7  Sniper.SetOptions("MyClass", "mend", "END!")
8
9  Sniper.initialize()
10 Sniper.run()
11 Sniper.finalize()
```

PyPy

```
1  import cppyy
2  cppyy.load_reflection_info('libSniperMgrDict.so')
3  Sniper = cppyy.gbl.SniperMgr("test.txt")
```

Keep libSniperMgrDict.so available in LD_LIBRARY_PATH.

Option file

Option file will be not used in the future, but now we still need it.

```
1  Sniper.Cycler = "NormCycler";
2  Sniper.InputSvc = "NONE";
3
4  Sniper.Dlls += { "MyClass" };
5  AlgMgr.Contents += { "MyClass" };
6
7  #include "$ROOTWRITERROOT/share/RootWriter.txt"
8
9  RootWriter.Output = { "FILE1" : "output1.root",
10 "FILE2" : "output2.root" };
11
12 Sniper.LogLevel = 3;  //Info
```

Output

```
> python run.py
**********************
** Welcome to SNiPER ** :
**********************
Sniper.SniperMgr            INFO: test.txt
...
Sniper.SniperMgr            INFO: Select InputSvc : NONE
Sniper.SniperMgr            INFO: Select Cycler   : NormCycler
SvcMgr.SvcMgr               INFO: Load service    : RootWriter
SvcMgr.SvcMgr               INFO: Load service    : NormCycler
AlgMgr.AlgMgr               INFO: Add  algorithm  : MyClass
Sniper.initialize           INFO: Successfully initialized
fibonacci(21) = 10946
...
fibonacci(35) = 9227465
Sniper.run                  INFO: Total processed events 15
END!
Sniper.finalize             INFO: Successfully finalized
Sniper.~SniperMgr           INFO: Terminated
```

Outline

Motivation

Framework

C++
- User Algorithm
- SniperMgr

CMT

Python

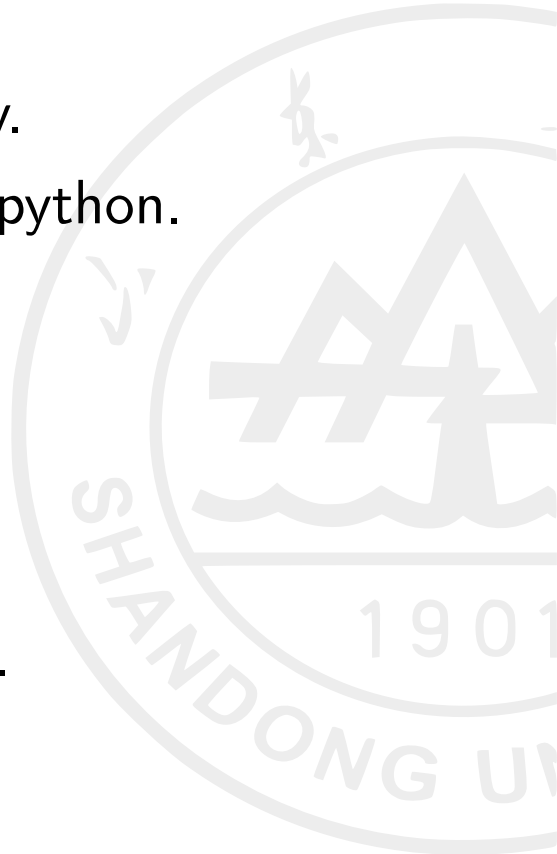
Summary

Summary

1. For SniperMgr, a python interface is defined.
2. Give an interface to the dynamic loaded library.
3. Make the user algorithm property available in python.

Next to do...

Make any user given classes available in python.
Increase the robustness of the interface code.
Give a more pythonic interface in the python script.



That's all

Thanks for your attention!