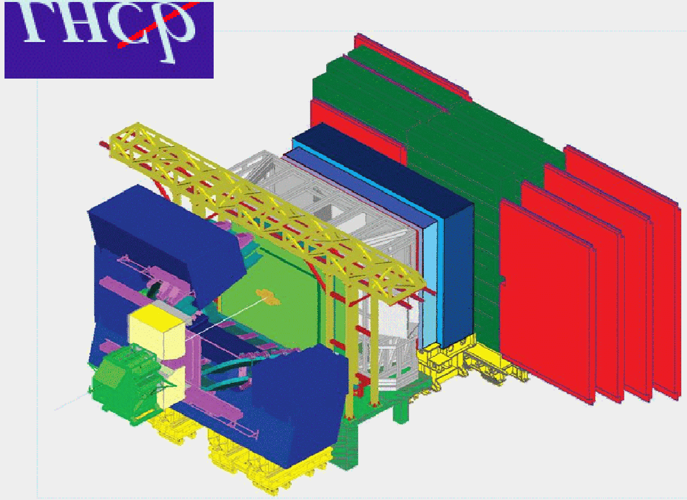
Introduction to Python

Eduardo Rodrigues
University of Glasgow

Course given at Glasgow, 12-18 March 2008



Part III

GaudiPython

Part III: GaudiPython

I. The GaudiPython package

II. Environment set-up

III. Getting started

IV. Typical applications

V. Advanced applications

The GaudiPython package

What is GaudiPython?

What is Gaudi?

- **General software framework for HEP C++ programs**
  - e.g. digitization, reconstruction, analysis
- **Used by LHCb and ATLAS**
- **In essence a Gaudi program is a "main" program that reads in options … and runs …**

What is GaudiPython?

- **Python bindings to Gaudi**
- **Re-implemented using PyRoot since Gaudi v18r0**

Why use it?

- **For interactive analysis, testing of code being developed, etc.**
- **Makes the development cycle much much faster!**
- **Access to full C++ classes from Python:**
  - Gaudi algorithms, tools, services, ATLAS/LHCb event classes, etc.

The GaudiPython package

GaudiPython/__init__.py

GaudiPython/Bindings.py

GaudiPython/Pythonizations.py

"core" modules:

import GaudiPython

GaudiPython/GaudiAlgs.py

GaudiPython/HistoUtils.py

GaudiPython/TupleUtils.py

extra modules: e.g.

from GaudiPython
import GaudiAlgs

C++ to (Gaudi)Python mapping

:: (the global namespace)	**GaudiPython.gbl**
Namespace::Class	**GaudiPython.gbl.Namespace.Class**
object = new Class(…)	**object = Class(…)**
enum::item	**enum.item**
Null pointer	**None**

Many examples will be given
in the next slides …

Environment set-up

Setting up the environment

- Depends very much on the experiment: LHCb, ATLAS
- Depends also on program environment:

  reconstruction / digitization / … program
- E.g. certain classes are only available in the reconstruction program
- I will assume you have the adequate environment …
- Most often provided “for free” to users

Getting started

Importing GaudiPython

```
# what does GaudiPython has to offer?
>>> import GaudiPython

>>> dir(GaudiPython)
['AppMgr', 'Bindings', 'CallbackStreamBuf', 'FAILURE', 'Helper',
'Interface', 'InterfaceCast', 'PropertyEntry', 'PyAlgorithm',
'Pythonizations', 'ROOT', 'SUCCESS', '__builtins__', '__doc__',
'__file__', '__name__', '__path__', 'deprecation', 'gbl',
'getClass', 'iAlgTool', 'iAlgorithm', 'iDataSvc',
'iHistogramSvc', 'iNTupleSvc', 'iService', 'loaddict',
'makeClass', 'makeNullPointer', 'setOwnership', 'toArray',
'toIntArray', 'toShortArray']
```

- **“import GaudiPython” imports in reality the module “__init__.py” of the GaudiPython package**
- **One finds above many well-known Gaudi components …**

The application manager

```
 # the usual very many methods of the application manager
 >>> dir(GaudiPython.AppMgr)
 ['__class__', '__delattr__', '__dict__', '__doc__',
'__getattr__', '__getattribute__', '__hash__', '__init__',
'__module__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__str__', '__weakref__',
'addAlgorithm', 'algorithm', 'algorithms', 'config', 'configure',
'createSvc', 'datasvc', 'declSvcType', 'detSvc', 'detsvc',
'evtSel', 'evtSvc', 'evtsel', 'evtsvc', 'execute', 'executeEvent',
'exit', 'finalize', 'getInterface', 'histSvc', 'histsvc',
'initialize', 'isValid', 'loaddict', 'name', 'ntupleSvc',
'ntuplesvc', 'optSvc', 'partSvc', 'partsvc', 'properties',
'property', 'readOptions', 'reinitialize', 'removeAlgorithm',
'retrieveInterface', 'run', 'service', 'services',
'setAlgorithms', 'state', 'tool', 'toolSvc', 'toolsvc']
```

❑ Many of these methods will be looked at in the following slides …

The global namespace

```
# “gbl” stands for the “global namespace”
>>> dir(GaudiPython.gbl)
['AIDA', 'ContainedObject', 'DataObject',
'GaudiHandleArrayProperty', 'GaudiHandleProperty', 'GaudiPython',
'IUpdateManagerSvc', 'ObjectContainerBase', 'SimpleProperty',
'SimplePropertyRef', 'StatusCode', '_Bit_reference', '__doc__',
'__module__', 'std']
```

The STL classes
are available !

```
>>> dir(GaudiPython.gbl.std)
['__doc__', '__module__', 'complex', 'deque', 'exception',
'list', 'map', 'multimap', 'multiset', 'name', 'pair', 'queue',
'set', 'stack', 'stlclasses', 'vector']
```

The STL library (1/2)

```
>>> GaudiPython.gbl.std.vector( ‘int’ )
<class ‘PyCintex.vector<int>’>

>>> v_int = GaudiPython.gbl.std.vector( ‘int’ )()
>>> dir(v_int)
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__',
...
'__str__', '__weakref__', '_getitem__unchecked', '_vector__at',
'assign', 'at', 'back', 'begin', 'capacity', 'clear',
'createCollFuncTable', 'empty', 'end', 'erase', 'front',
'insert', 'max_size', 'pop_back', 'push_back', 'reserve',
'resize', 'size', 'swap']
>>> v_int
<ROOT.vector<int> object at 0x156f8f0>
>>> v_int.size()
0L
>>> v_int.push_back(10); v_int.push_back(20); v_int.push_back(30)
>>> v_int.size()
3L
>>> for i in v_int:
...     print i,
10 20 30
>>> v_int.clear()
>>> len( v_int )
0
```

The STL library (2/2)

```
# 2 equivalent ways of defining classes
>>> GaudiPython.gbl.std.pair( ‘int’, ‘int’ )
<class ‘PyCintex.pair<int,int>’>
>>> GaudiPython.gbl.std.pair( int, int )
<class ‘PyCintex.pair<int,int>’>

# only 1 possible way of defining the class
# since “double” is not a Python type !
>>> GaudiPython.gbl.std.pair( ‘int’, ‘double’ )
<class ‘PyCintex.pair<int,double>’>

>>> GaudiPython.gbl.std.vector( ‘std::vector<double>’ )
<class ‘PyCintex.vector<vector<double> >’>
>>> std = GaudiPython.gbl.std
>>> std.vector( std.vector('double') )
<class 'PyCintex.vector<vector<double> >'>

# Note: some combinations will not work simply because there
# are no dictionaries for them !

>>> p = GaudiPython.gbl.std.pair( int, int )( 1, 2 )
>>> print p.first, p.second
1 2
```

Fancier manipulations …

Units and physical constants

GaudiKernel package

```
# “All” the units
>>> from GaudiKernel import SystemOfUnits

>>> SystemOfUnits.mm    # the millimeter is a basic unit
1.0
>>> SystemOfUnits.m
1000.0

>>> from GaudiKernel.SystemOfUnits import meter, millimeter
>>> meter == 1000. * millimeter
True

# “All” the physical constants
>>> from GaudiKernel import PhysicalConstants
```

```
# printing in specified units !
>>> from GaudiKernel.SystemOfUnits import meter, second
>>> from GaudiKernel.PhysicalConstants import c_light
>>> print c_light
299.792458
>>> print c_light / (m/s)
299792458.0
```

Typical applications

A Gaudi job that does nothing

```
>>> from GaudiPython import AppMgr
# instantiate the application manager
>>> appMgr=AppMgr()

ApplicationMgr    SUCCESS
============================================================================================
========================================
                                                   Welcome to ApplicationMgr $Revision: 1.71
$
                                          running on lxplus208.cern.ch on Mon Mar 17
22:13:37 2008
============================================================================================
========================================
ApplicationMgr       INFO Application Manager Configured successfully
# initialise the application manager
>>> appMgr.initialize()
HistogramPersis...   INFO  'CnvServices':[ 'HbookHistSvc' , 'RootHistSvc' ]
HistogramPersis...WARNING Histograms saving not required.
ApplicationMgr       INFO Application Manager Initialized successfully
SUCCESS
# run over the full sample
>>> appMgr.run(-1)
EventSelector        INFO End of event input reached.
EventLoopMgr         INFO No more events in event selection
# finalise / exit
>>> appMgr.exit()
EventLoopMgr         INFO Histograms converted successfully according to request.
ToolSvc.finalize()   INFO Removing all tools created by ToolSvc
ApplicationMgr       INFO Application Manager Finalized successfully
ApplicationMgr       INFO Application Manager Terminated successfully
```

Note: program ran piece-by-piece !

A trivial GaudiPython job

One can now use “standard” or Python job options

```
import GaudiPython

# instantiate the application manager
# with an options file
appMgr = GaudiPython.AppMgr( joboptions = ‘MyOptions[.opts|.py]’ )

appMgr.run( 1 )
```

❑ **Can be a handy way of running, for trivial tests**

Adding options

```
appMgr = GaudiPython.AppMgr( joboptions = ‘MyOptions[.opts|.py]’ )

# simple way of adding extra files of options ...
# ... and extra options / properties ...
appMgr.config( files = [ ‘$MYONEPACKAGE/ExtraOpts1.opts’,
                         ‘$MYOTHERPACKAGE/ExtraOpts2.opts’ ] ,
               options = [ ‘EventSelector.PrintFreq=100’ ]
             )

appMgr.run( 1 )
```

Reading and re-reading an input file

```
>>> appMgr = AppMgr( joboptions = ‘MyOptions[.opts|.py]’ )

# get hold of the event selector
>>> evtSel = appMgr.evtSel()

# open your file
>>> evtSel.open( [ ‘PFN:/my/path/to/file/AlocalDataFile.dst’ ] )

# run e.g. one single test event
>>> appMgr.run( 1 )

 # perform at this level some tests on the data !

# rewind the file
>>> evtSel.rewind()

# run again the first event
>>> appMgr.run( 1 )
```

Can be a physical file name, a CASTOR path, etc.

- ❑ **Particularly useful to test a piece of analysis code on a user-made DST**

Accessing the Transient Event Store (TES)

```
# event data service
>>> evtSvc = appMgr.evtSvc()

# “dump” the event (only shows sub-sample of TES structure!)
>>> evtSvc.dump()

# retrieve data containers from the TES
>>> mcparts = evtSvc[ ‘/Event/MC/Particles’ ]
>>> tracks  = evtSvc[ ‘Rec/Track/Best’ ]

>>> for track in tracks:
...     print ‘Track #’, track.key(), ‘: p =‘, track.p()
```

- **One can equally access detector geometry information via the detector data service !**

```
# detector data service
>>> detSvc = appMgr.detSvc()

# “dump” the detector structure (be aware: very large output!)
>>> detSvc.dump()

# check geometry or conditions information
>>> detSvc[ ‘/some/path/to/geometry’ ]
>>> detSvc[ ‘/some/path/to/a/condition’ ]
```

Getting hold of tools and services

```
# instantiate the tool
>>> myTool = appMgr.toolsvc().create( ‘MyTool’, \
...                             interface = ‘MyToolInterface’ )


# initialize the tool
>>> myTool.initialize()

# use it from now on ...
```

```
 # instantiate a service
>>> magSvc = appMgr.service( ‘MagneticFieldSvc’, \
 ...                         ‘IMagneticFieldSvc’ )


 # use it ...
```

Event model classes

```
# LHCb event model classes: in “LHCb::” namespace
>>> GaudiPython.gbl.LHCb
<class ‘__main__.LHCb’>

# some examples
>>> GaudiPython.gbl.LHCb

>>> LHCb.MCParticle
<class ‘__main__.LHCb::MCParticle’>

>>> LHCb.Track
<class ‘__main__.LHCb::Track’>

# instantiating an event model class ...
>>> track = LHCb.Track()

# ... and a STL vector of event model classes
>>> trackVector = GaudiPython.gbl.std.vector( ‘LHCb::Track *’ )()

>>> trackVector
 <ROOT.vector<LHCb::Track*> object at 0x22aa460>

>>> trackVector.push_back( track )
>>> trackVector.size()
1L
```

It should be similar for ATLAS event model classes

Advanced applications

Changing options

```
>>> appMgr = AppMgr( joboptions = ‘MyOptions.opts’ )

# check the algorithms known to the application manager ...
# ... at this level the appMgr only knows about configuration
>>> appMgr.algorithms()
[]
>>> appMgr.initialize()
# ... at this level the appMgr is initialized
>>> appMgr.algorithms()
# it prints out a large list of strings of algorithm names ...

 # at this level you can get access to options !
>>> dstWriter = appMgr.algorithm( ‘DstWriter’ )
>>> dstWriter
<GaudiPython.Bindings.iAlgorithm object at 0x2a9d92ec90>

# list all the properties of this algorithm
>>> for k, prop in dstWriter.properties().items() :
...     print k, prop.value()

# finally change an option
>>> print dstWriter.OutputFile
‘PFN:SomeFileame.dst’
>>> dstWriter.OutputFile = ‘AnotherFilename.dst’
# now run the program ...
```

User-defined algorithms

You can write full Gaudi algorithms in Python !

```
# basic GaudiPython algorithm class
>>> from GaudiPython import PyAlgorithm


# sophisticated algorithms in the GaudiAlgs module
>>> import GaudiPython.GaudiAlgs

# Python brother of C++ GaudiAlgorithm
>>> from GaudiPython.GaudiAlgs import GaudiAlgo

# Python brother of C++ GaudiHistoAlg
>>> from GaudiPython.GaudiAlgs import HistoAlgo

# Python brother of C++ GaudiTupleAlg
>>> from GaudiPython.GaudiAlgs import TupleAlgo
```

Gaudi histograms and n-tuples

```
# histogram service
>>> histoSvc = appMgr.histSvc()


# n-tuple service
>>> ntupleSvc = appMgr.ntupleSvc()
```

- Check out the many helper methods …

- Very powerful helpers:

```
# other handy modules
>>> from GaudiPython import HistoUtils

>>> from GaudiPython import TupleUtils
```

Python configurables

Main idea:

- ❑ Configuring and running are two different things
- ❑ Should be possible to configure and run an application in two separate steps

Disclaimer:

- ❑ No time to discuss this here ☹
- ❑ For the future ...
- ❑ Check out GaudiPython releases …

Good luck …

… and have fun …