Tracking in Python

E. Rodrigues, NIKHEF

How to get started
What this tutorial is not

- A course on Python
- A course on GaudiPython
- A technical presentation

What this tutorial is

- An incentive to start exploiting Python in our everyday work
- A presentation a la “learning by examples”
- An informal tutorial
Getting started ...
C++ to Python Mapping

<table>
<thead>
<tr>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>:: (the global namespace)</td>
<td>gaudimodule.gbl</td>
</tr>
<tr>
<td>Namespace::Class</td>
<td>gaudimodule.gbl.Namespace.Class</td>
</tr>
<tr>
<td>object = new Class( ... )</td>
<td>object = Class( ... )</td>
</tr>
<tr>
<td>enum::item</td>
<td>enum.item</td>
</tr>
<tr>
<td>Null pointer</td>
<td>None</td>
</tr>
</tbody>
</table>

Examples will be given throughout the next transparencies …
Python Packages available (1/2)

GaudiPython

- Top-level framework for working with Gaudi applications in Python
- Interface of Gaudi to Python
- Contains a series of modules:
  - *e.g.: gaudimodule:* main module, allows instantiation of ApplicationManager
  - *GaudiAlgs:* for using GaudiAlgorithm, GaudiTupleAlg, etc.
  - *units:* standards HEP units as in Geant4

PyRoot

- Exposes ROOT to Python
- RooFit is now also available!
- *ROOT* module

*Purely descriptive – examples will follow …*
**Tr/TrackPython**

- Introduced to expose main tracking tools to Python
- Access to extrapolators, fitter, projectors, clone finder, etc.
- More will be added …
- Note: simple module that is likely to evolve as GaudiPython evolves too …
- \texttt{gtracktools} module

**Event/LinkerInstances**

- Facilitates access to main linker classes LinkedTo & LinkedFrom in Event/LinkerEvent
- Easy manipulation of links to MC-truth in Python
- \texttt{eventassoc} module
Setting the Environment

Software management

- LHCb software managed via CMT
- CMT sets up the consistent environment for a given application

In the CMT requirements file

- use GaudiPython v^*
- use TrackPython v^* Tr
- use EventInstances v^* Event

For using GaudiPython

For using some tracking tools and the association tables
Some first manipulations
Reading a Gaudi file

Minimum required to read a file

```python
>>> import gaudimodule
>>> appMgr = gaudimodule.AppMgr(outputlevel=3,
                               joboptions='${EVENTSYSROOT}/options/PoolDicts.opts')

>>> SEL = appMgr.evtSel()
>>> SEL.open('PFN:rfio:/castor/cern.ch/user/c/cattanem/Boole/v11r5/0601-11144100.digi')
>>> appMgr.run(1)
>>> EVT = appMgr.evtSvc()
>>> EVT.dump()
```

Content of the TES (not all content is shown by default ...)

OUTPUT OF `EVT.dump()`
Running a Gaudi Application

Example: to run Brunel

```python
>>> import gaudimodule
>>> appMgr = gaudimodule.AppMgr(outputlevel=3, joboptions='./options/v200601.opts')
>>> appMgr.run(1)
# feel like having a look at the TES?
>>> EVT = appMgr.evtSvc()
>>> EVT.dump()
```

Making use of interactivity ...

```python
>>> SEL = appMgr.evtSel()
>>> dir(SEL)
['__class__', '__delattr__', '__dict__', '__doc__', '__getattribute__', '__hash__', '__init__',
'_module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__', '__weakref__',
'_ip', '_isvc', '_name', '_optsvc', '_svcloc', 'finalize', 'g', 'getInterface', 'initialize', 'isnumber', 'isvector',
'name', 'open', 'properties', 'reinitialize', 'retrieveInterface', 'rewind', 'typecnv']
# say you need more printout of the algorithms/tools names ...
>>> MSG = appMgr.service('MessageSvc')
>>> MSG.Format = '% F%60W%S%7W%R%T %0W%M';
>>> SEL.rewind()
# off you go again ...
>>> appMgr.run(1)
```

Just one possibility ...
Retrieving objects from the TES

From now on let us assume we’ve always run at least 1 event …

```python
>>> import gaudimodule
>>> appMgr = gaudimodule.AppMgr(outputlevel=3, joboptions='../options/v200601.opts')
>>> appMgr.run(1)
>>> EVT = appMgr.evtSvc()
```

One might need to load some dictionaries for inspecting objects

```python
appMgr.loaddict('MCEventDict')
appMgr.loaddict('TrackEventDict')
```

This will become unnecessary in the future …

```python
>>> tracks = EVT['Rec/Track/Forward']
>>> print tracks.size()
11
>>> mcps = EVT['MC/Particles']
>>> for mcp in mcps:
    >>> print mcp.particleID.pid(), mcp.p()
```

Looping over the container
Tracking in Python

LHCb Note 2006-014 for further explanations
>>> Track = gaudimodule.gbl.LHCb.Track

>>> Track
<class '__main__.LHCb::Track'>

>>> defaultTrack = gaudimodule.gbl.LHCb.Track()

>>> defaultTrack
{ chi2PerDoF : 0
  nDoF : 0 flags : 0
  lhcbIDs :
  states :
  measurements :
  nodes :
}

Our classes are defined in the LHCb namespace

Get hold of the class definition

Track class instantiation

Track data members

appMgr.loaddict( ‘TrackFitEventDict’ )
appMgr.loaddict( ‘LHCbKernelDict’ )

>>> aLineTraj = gaudimodule.gbl.LHCb.LineTraj

>>> aStateTraj = gaudimodule.gbl.LHCb.StateTraj

>>> otMeas = gaudimodule.gbl.LHCb.OTMeasurement()
Playing with the enums ...

```python
>>> State = gaudimodule.gbl.LHCb.State
>>> State.LocationUnknown
0
>>> State.AtT
5
>>> State.BegRich2
8

>>> Measurement = gaudimodule.gbl.LHCb.Measurement
>>> Measurement.Unknown
0
>>> Measurement.VeloR
1
>>> Measurement.TT
3
```

*Neat way of using the enums:*

*As close as possible to the C++ syntax*

*e.g.: State::AtT  ->  State.AtT*
If you ever thought the Track class cannot answer many questions...
Operations on the enums

```python
>>> track.checkType( Track.Long ), track.type() == Track.Long
(1, True)

>>> track.checkHistory( Track.PatForward )
1

>>> track.checkStatus( Track.Fitted )
0
```

Basic properties

```python
>>> track.charge()
1

>>> track.nDoF()
12
```

Track contents

```python
>>> track.nStates()
2L

>>> state = track.states()[0]

>>> state.x(), state.y(), state.z()
(-0.0321825934759312, 0.11126547797391403, -14.582833595894137)
```
The Track can also be modified …

```python
>>> newtrack = track.clone()
>>> newtrack.setFlag( Track.Clone, True )
>>> newtrack.checkFlag( Track.Clone )
1
```

What’s on the track?

```python
>>> print track.nLHCbIDs()
32
>>> print track.nMeasurements()
0
>>> ids = track.lhcbIDs()
>>> ids
<ROOT.vector<LHCb::LHCbID> object at 0xf4eab24>
```
>>> from gtracktools import setToolSvc, extrapolator
>>> setToolSvc( appMgr )
>>> linprop = extrapolator( 'TrackLinearExtrapolator' )
>>> dir(linprop)
# some lines skipped ...
'__setattr__', '__str__', '__weakref__', 'addRef', 'finalize',
'initialize', 'interfaceID', 'momentum', 'name', 'p', 'parent',
'position', 'positionAndMomentum', 'propagate', 'pt',
'queryInterface', 'release', 'slopes', 'transportMatrix', 'type']

>>> help(linprop.propagate)
# one gets several lines of documentation

# Helper dict. of package with gtracktools module
appMgr.loaddict( 'TrackPythonDict' )

*Needs to be called only once (may become unnecessary ...*)

*Get hold of the Linear extrapolator*
# get hold of the Track to be extrapolated
>>> track = tracks[3]

# instantiate a State, to retrieve the result of the extrapolation
>>> state = gaudimodule.gbl.LHCb.State()

>>> state.x(), state.y(), state.z()
(0.0, 0.0, 0.0)

# instantiate the parabolic extrapolator
>>> parprop = extrapolator( 'TrackParabolicExtrapolator' )

# define the z-position to extrapolate to
>>> znew = 100.

>>> parprop.propagate( track, znew, state )
SUCCESS

>>> state.x(), state.y(), state.z()
(-5.859069220236659, -1.5738179596044015, 100.0)

# "state" variable contains some random State
>>> state.x(), state.y(), state.z()
(491.95847296136429, -39.394353509484759, 9520.0)

>>> newstate = state.clone()

>>> linprop.propagate( newstate, 5000. )
SUCCESS

>>> newstate.x(), newstate.y(), newstate.z()
(72.562676254077573, -21.114433639356392, 5000.0)
Checking on Clone Tracks

>>> from gtracktools import cloneFinder
>>> CLONEFINDER = cloneFinder('TrackCloneFinder')

>>> track0 = tracks[0]
>>> track1 = tracks[1]
>>> CLONEFINDER.areClones(track0, track1) == True
False

>>> track1 = track0.clone()
>>> CLONEFINDER.areClones(track0, track1) == True
True

GETTING HOLD OF THE CLONES FINDER

Pick up some 2 tracks

Are they clones?
Fitting tracks with the default options cannot get simpler.

# The "complete job", i.e. fitting and setting of the appropriate flags, only takes a few lines:

```python
>>> track = tracks[0]
>>> sc = MASTERFITTER.fit( track )
>>> if sc == gaudimodule.SUCCESS:
...    track.setStatus( Track.Fitted )
... else:
...    track.setStatus( Track.FitFailed )
...    track.setFlag( Track.Invalid, True )
```

---

**GETTING HOLD OF THE MASTER FITTER**

---

# Fitting tracks with the default options cannot get simpler.

---

**Pick up some track and fit it**

---

**Done! Set Status flag**

---
<table>
<thead>
<tr>
<th>Step</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&gt;&gt;&gt; location = 'Rec/Track/Velo'</td>
</tr>
<tr>
<td>2</td>
<td>&gt;&gt;&gt; tracks = EVT[ location ]</td>
</tr>
<tr>
<td>3</td>
<td>&gt;&gt;&gt; from eventassoc import linkedTo</td>
</tr>
<tr>
<td>4</td>
<td>&gt;&gt;&gt; Track = gaudimodule.gbl.LHCb.Track</td>
</tr>
<tr>
<td>5</td>
<td>&gt;&gt;&gt; MCParticle = gaudimodule.gbl.LHCb.MCParticle</td>
</tr>
<tr>
<td>6</td>
<td>&gt;&gt;&gt; LT = linkedTo( MCParticle, Track, location )</td>
</tr>
<tr>
<td>7</td>
<td>&gt;&gt;&gt; LT</td>
</tr>
<tr>
<td></td>
<td>&lt;ROOT.LinkedTo<a href="">LHCb::MCParticle,LHCb::Track</a> object at 0xf485460&gt;</td>
</tr>
<tr>
<td>8</td>
<td>&gt;&gt;&gt; LT.notFound() == False</td>
</tr>
<tr>
<td>9</td>
<td>True</td>
</tr>
<tr>
<td>10</td>
<td>&gt;&gt;&gt; track = tracks[7]</td>
</tr>
<tr>
<td>11</td>
<td>&gt;&gt;&gt; mcp = LT.first( track )</td>
</tr>
<tr>
<td>12</td>
<td>&gt;&gt;&gt; mcp.key()</td>
</tr>
<tr>
<td>13</td>
<td>849</td>
</tr>
<tr>
<td>14</td>
<td>&gt;&gt;&gt; mcp</td>
</tr>
<tr>
<td></td>
<td>{ momentum : (-673.62,-416.03,11761.6,11789)</td>
</tr>
<tr>
<td>15</td>
<td>particleID : { pid : 211</td>
</tr>
<tr>
<td>16</td>
<td>}</td>
</tr>
<tr>
<td>17</td>
<td>}</td>
</tr>
<tr>
<td>18</td>
<td>&gt;&gt;&gt; mcp = LT.next()</td>
</tr>
<tr>
<td>19</td>
<td>&gt;&gt;&gt; mcp == None</td>
</tr>
<tr>
<td>20</td>
<td>True</td>
</tr>
</tbody>
</table>
Going « back and forth » with links …

```python
>>> from eventassoc import linkedFrom
>>> LF = linkedFrom( Track, MCParticle, location )
>>> LF
<ROOT.LinkedFrom<LHCb::Track,LHCb::MCParticle> object at 0xf4d6210>
>>> LF.notFound() == False
True
>>> track.key()
7
>>> mcp = LT.first( track )
>>> mcp.key()
849
>>> trk = LF.first( mcp )
>>> trk.key()
7
```

With the looks of the Associators …

```python
>>> range = LT.range( track )
>>> range
<ROOT.vector<LHCb::MCParticle*> object at 0xfe27e90>
>>> range.size()
1L
>>> mcp = range[0]
>>> mcp.key()
849
```
Using what is available …

```python
import gaudi_module
from GaudiAlg import GaudiAlgo

from units import GeV

# Define the histograms
from ROOT import gROOT, TH1F
gROOT.SetBatch()

histo = TH1F('long_p', 'P (GeV) for Long tracks', 100, 0., 100.)

class LongTrackAalgo(GaudiAlgo):
    """Algorithm to plot the momentum of Long tracks from PatForward""

    # Initialize
    def __init__(self, name='LongTrackAalgo'):
        GaudiAlgo.__init__(self, name)
        print(self.name(), '=> Initialize')

    # Execute
    def execute(self):
        print(self.name(), '=> Execute')
        tracks = self.get('Rec/Track/Forward')
        histo.Fill(t.p / GeV) for t in tracks
        return gaudi_module.SUCCESS

    # Finalize
    def finalize(self):
        print('Long tracks P (MeV) mean, histo.SetMean(),
        'rms, histo.SetRMS()')

        # Write out the histogram
        from ROOT import TFile
        f = TFile('LongTrackAalgo.root', 'recreate')
        histo.Write()
        f.Close()

return GaudiAlgo.finalize()
```

**Python version of GaudiAlgorithm**

**Standard HEP units**

**Algorithm definition**

**Algorithm’s main methods**

**Save the ROOT histo to a file**
All of this will execute if the whole previous file is run as `python -i myJob.py` (with this bit appended):

```python
>>> appMgr = gaudimodule.AppMgr( outputlevel=3, joboptions='../../options/v200601.opts' )
>>> appMgr.loaddict( 'TrackEventDict' )
>>> appMgr.loaddict( 'TrackPythonDict' )
>>> EVT  = appMgr.evtSvc()
>>> appMgr.addAlgorithm( LongTrackPAlgo( 'LongTrackP' ) )
>>> appMgr.run( 250 )
>>> from ROOT import TCanvas
>>> histo.Draw()
# not necessary unless you really want to finish everything:
#appMgr.finalize()
# run some more events!
>>> appMgr.run( 100 )
# continue playing interactively ...

# with the « -i » option the user gets back control on the prompt ...
>>> f = ROOT.TFile( LongTrackPAlgo.root', 'READ' )
# play with the contents ...
```