# rootpy: Pythonic ROOT

rootpy.org

Noel Dawe

on behalf of all rootpy contributors

September 20, 2016

## What's the problem?

**Why would we even consider developing a layer on top of PyROOT?**

- PyROOT is mainly bindings (although with some pythonization).
- Python's dynamic nature provides **many possibilities not currently realized by PyROOT**. One might argue a majority of these high-level pythonizations are **beyond the scope of what PyROOT should offer.**
- Certain tasks require awkward code and are error-prone. Similar workarounds are implemented by many people in multiple places.
  ***Why not solve these issues once and for all?***
- There is a lack of integration of ROOT with the vast and growing ecosystem of scientific Python packages.
  ***Why not enable users to benefit from both the power of ROOT and what is offered by the scientific Python community?***

## Scientific Python Applications

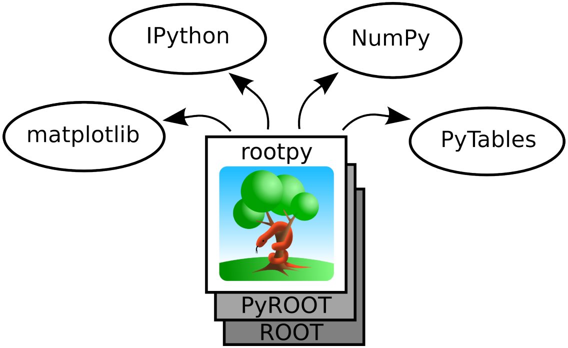
"Can I really perform complicated analysis with Python?"
**Short answer: Yup!**

- ROOT with PyROOT
- Interactive computing: Jupyter Notebook
- Powerful and fast array manipulation: numpy
- Feature-rich publication-quality plotting: matplotlib
- Efficiently and easily handle large amounts of data: PyTables, h5py
- General scientific library: scipy
- Flexible data analysis and manipulation: pandas
- Symbolic mathematics: sympy
- Statistical models and tests: statsmodels
- Fitting: iminuit and sherpa
- Astronomy: astropy
- Image processing: scikit-image
- Machine learning: scikit-learn

Search for packages on the Python Package Index

## Introducing rootpy…

- rootpy aims to provide a ***more pythonic layer*** on top of the PyROOT bindings and to take advantage of advanced features of the Python language.
- **rootpy does not intend to recreate ROOT** or to *severely* alter the default behaviour of ROOT.
- **rootpy is not an analysis framework**, but rather a library that your analysis code might use.
- rootpy provides an **interface with the scientific Python packages**:



## Why the name "rootpy"?

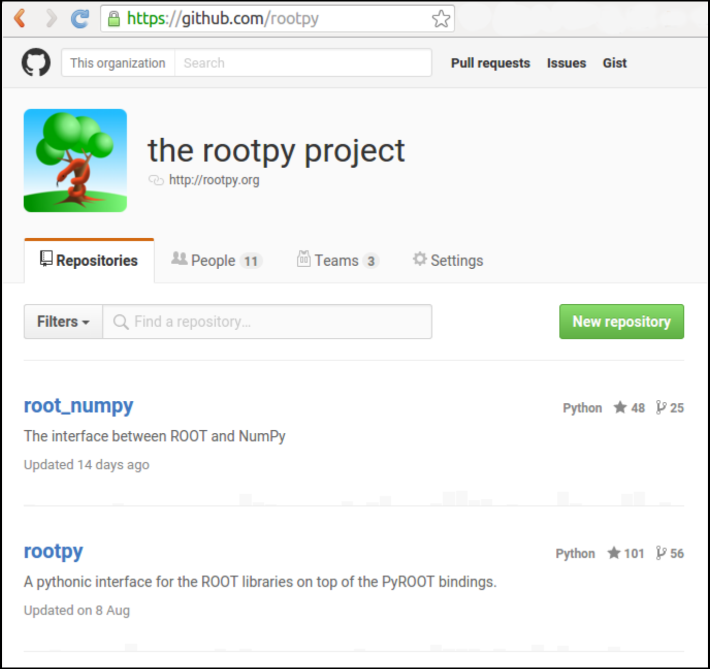
numpy, scipy, h5py, … See the pattern?

"rootpy" is meant to automatically convey the idea "Like PyROOT, but Pythonic?"

*"Pythonic means code that doesn't just get the syntax right but that follows the conventions* of the Python community and uses the language in the way it is intended to be used."*

— on stackoverflow

* Capitalization is important. Never ROOTPy, RootPy, etc.
See PEP8


https://github.com/rootpy

the rootpy project

http://rootpy.org

root_numpy

Python ★ 48 25

The interface between ROOT and NumPy

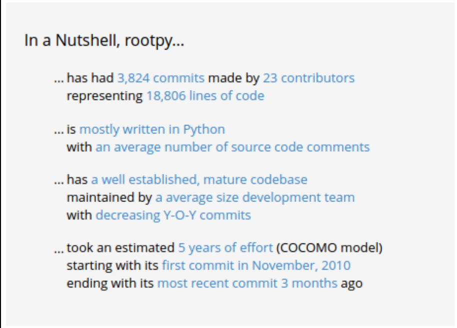
Updated 14 days ago

rootpy

Python ★ 101 56

A pythonic interface for the ROOT libraries on top of the PyROOT bindings.

Updated on 8 Aug



In a Nutshell, rootpy...

... has had 3,824 commits made by 23 contributors representing 18,806 lines of code

... is mostly written in Python with an average number of source code comments

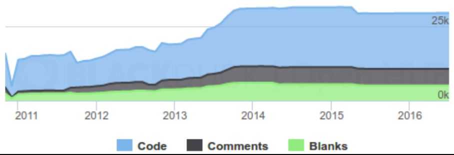
... has a well established, mature codebase maintained by a average size development team with decreasing Y-O-Y commits

... took an estimated 5 years of effort (COCOMO model) starting with its first commit in November, 2010 ending with its most recent commit 3 months ago

Languages

| | | | |
|---|---|---|---|
| Python | 98% | 6 Other | 2% |

Lines of Code



19k lines of code
Over 3800 commits in 535 pull requests
Many users and developers from LHC and non-LHC experiments

## rootpy: Key Concepts and Design Philosophy

- Pythonized **classes in rootpy are subclasses of the corresponding ROOT classes** with the same (or similar) name, but without the "T".
- ROOT methods may be overridden or new methods created to add functionality and objects may be decorated with additional properties.
- **Object names and titles are optional**. Unspecified names default to UUIDs.
- ROOT messages can be routed through Python's logging system with **error messages raised as Python exceptions**.
- **Python has a garbage collector but C++ does not:** this can lead to strange issues. rootpy addresses these problems.
- Anywhere Python is typically slow we try to use **compiled C extension modules**. Through the root_numpy package, rootpy provides very fast conversion of ROOT Trees into NumPy arrays as well as efficiently filling ROOT histograms with NumPy arrays.
- **rootpy can be dropped into existing code.**

## What does rootpy offer?

| | |
|---|---|
| **rootpy.plotting** | histogram, graphs, canvas, pad, legend, and style subclasses with additional pythonizations, including a matplotlib interface. |
| **rootpy.tree** | trees, chains, tree objects, tree models, cuts. |
| **rootpy.io** | file and directory subclasses, utilities, pickler. |
| **rootpy.logger** | route ROOT messages through Python's logging module. ROOT error messages become Python exceptions. |
| **rootpy.memory** | utilities for monitoring TObject deletions and for keeping objects alive when out of scope in Python. |
| **rootpy.interactive** | a wait function for preventing Python from exiting until all canvases have been closed. |
| **rootpy.stl** | automatic STL dictionary compilation and caching. |
| **rootpy.root2hdf5** | conversion of ROOT files into HDF5. |
| **rootpy.context** | utilities for managing ROOT's global state. |
| **rootpy.stats** | RooStats, RooFit, HistFactory. |

…and more …

See the root_numpy package for conversion of TTrees into NumPy arrays.

# Histograms

## PyROOT

```
from ROOT import TH3D
from array import array

# variable width bins
hist3d = TH3D('3d', '3d', 3, array('d', [0, 3, 10, 100]),
                          5, array('d', [2.3, 4.2, 5.8, 10, 20, 25.5]),
                          2, array('d', [-100, 0, 20]))
# ROOT is missing some constructors... (the following will not work)
hist3d = TH3D('3d', '3d', 3, 0, 5,
                          5, array('d', [2.3, 4.2, 5.8, 10, 20, 25.5]),
                          2, array('d', [-100, 0, 20]))
```

## rootpy

```
from rootpy.plotting import Hist3D

# variable width bins
hist3d = Hist3D([0, 3, 10, 100], [2.3, 4.2, 5.8, 10, 20, 25.5], [-100, 0, 20])
# easy to mix variable and fixed width bins with rootpy
hist3d = Hist3D(3, 0, 5, [2.3, 4.2, 5.8, 10, 20, 25.5], [-100, 0, 20])
```

## Histograms and Style

- rootpy reduces ROOT's histogram classes down to Hist, Hist2D, Hist3D.

```
>>> from rootpy.plotting import Hist2D
>>> hist = Hist2D(10, 0, 1, 5, 0, 1, type='F')
>>> hist.__class__.__bases__[-1].__name__
'TH2F'
```

- Attributes can be accessed via properties:

```
hist.title = 'Fit Result'
hist.fillstyle = 'solid'
```

- Colors can also be set using hex, RGB tuples, or SVG names:

```
hist.fillcolor = (32, 178, 170)
hist.linecolor = '#87cefa'
hist.markercolor = 'salmon'
```

- Histograms can be indexed and sliced like multidimensional arrays

```
hist[:] = hist.Clone()[::-1]  # reverse bin content and errors
```

## Cuts

### PyROOT

```
from ROOT import TCut

cut1 = TCut('a<10')
cut2 = TCut('b%2==0')

cut = TCut('(%s)&&(%s)' % (
    cut1.GetTitle(),
    cut2.GetTitle()))

print cut.GetTitle()
```

output:

```
(a<10)&&(b%2==0)
```

### rootpy

```
from rootpy.tree import Cut

cut1 = Cut('a < 10')
cut2 = Cut('b % 2 == 0')

cut = cut1 & cut2
print cut

# expansion of ternary conditions
cut3 = Cut('10 < a < 20')
print cut3

# easily combine cuts arbitrarily
cut = ((cut1 & cut2) | - cut3)
print cut
```

output:

```
(a<10)&&(b%2==0)
(10<a)&&(a<20)
((a<10)&&(b%2==0))||(!((10<a)&&(a<20)))
```

## PyROOT: “Exception Handling”

ROOT is unable to open the file of course and emits an error message but an exception is not raised at this point leading to (sometimes difficult to interpret) issues downstream:

```
myfile = TFile.Open("file_does_not_exist.root")
print myfile
myfile.Get("something")
```

```
Error in <TFile::TFile>: file file_does_not_exist.root does not exist
<ROOT.TFile object at 0x(nil)>
Traceback (most recent call last):
  File "file_open.py.tmp", line 11, in <module>
    myfile.Get("something")
ReferenceError: attempt to access a null-pointer
```

## rootpy: Exception Handling

rootpy routes ROOT messages through Python's logging system and raises error messages as Python exceptions at the point of failure:

```
myfile = root_open("file_does_not_exist.root")
print myfile
myfile.Get("something")
```

```
ERROR:ROOT.TFile.TFile] file file_does_not_exist.root does not exist
Traceback (most recent call last):
  File "file_open_rootpy.py.tmp", line 12, in <module>
    myfile = root_open("file_does_not_exist.root")
  File "/home/endw/.local/lib/python2.7/site-packages/rootpy/io/file.py", line 138, in root_open
    root_file = ROOT.R.TFile.Open(filename, mode)
  File "/home/endw/.local/lib/python2.7/site-packages/rootpy/io/file.py", line 138, in root_open
    root_file = ROOT.R.TFile.Open(filename, mode)
  File "/home/endw/.local/lib/python2.7/site-packages/rootpy/logger/magic.py", line 247, in inter
    raise exception
rootpy.ROOTError: level=3000, loc='TFile::TFile', msg='file file_does_not_exist.root does not exi
```

**An exception is raised at the point of failure!**

## rootpy: Files

- Simple dictionary-like access:

```
hist = myfile['histname']
myfile['new_name'] = hist
myfile.dirname.histname = hist
```

- Can be used as context managers:

```
from rootpy.io import TemporaryFile
with TemporaryFile() as f:
    # current directory is f
    pass
# current directory restored to previous state
# and f is closed (and deleted in this case)
```

- Utilities such as walking the contents of a file:

```
# recursively walk through the file
for path, dirs, objects in f.walk():
    print path, dirs, objects
```

# PyROOT: Filling a Tree

```
from ROOT import TTree, TFile
from array import array
from random import gauss

output_file = TFile.Open('output.root', 'recreate')
some_float = array('f', [0.])
some_int = array('i', [0])
tree = TTree('mytree', '')
tree.Branch('some_float', some_float, 'some_float/F')
tree.Branch('some_int', some_int, 'some_int/I')

for i in xrange(100):
    some_float[0] = gauss(0, 1)
    some_int[0] = i
    tree.Fill()

tree.Write()
output_file.Close()
```

## rootpy: Filling a Tree

**Create and handle branches automatically:**

```
from rootpy.tree import Tree
from rootpy.io import root_open
from random import gauss

f = root_open("test.root", "recreate")

tree = Tree("test")
tree.create_branches(
    {'some_float': 'F',
     'some_int': 'I'})

for i in xrange(10000):
    tree.some_float = gauss(.5, 1.)
    tree.some_int = i
    tree.fill()
tree.write()
f.close()
```

**or with "TreeModels":**

```
from rootpy.tree import Tree, TreeModel
from rootpy.tree import FloatCol, IntCol
from rootpy.io import root_open
from random import gauss

f = root_open("test.root", "recreate")

class Model(TreeModel):
    some_float = FloatCol()
    some_int = IntCol()

tree = Tree("test", model=Model)

for i in xrange(10000):
    tree.some_float = gauss(.5, 1.)
    tree.some_int = i
    tree.fill()
tree.write()
f.close()
```

## rootpy: Tree Models

Easily create complex trees by simple class inheritance (inspired by PyTables):

```
from rootpy.tree import Tree
from rootpy.tree import TreeModel
from rootpy.tree import FloatCol
from rootpy.tree import IntCol

class FourVect(TreeModel):
    eta = FloatCol(default=-1111.)
    phi = FloatCol(default=-1111.)
    pt = FloatCol()
    m = FloatCol()

class Tau(FourVect):
    numtrack = IntCol()

class Event(Tau.prefix('tau1_'),
            Tau.prefix('tau2_')):
    event_number = IntCol()
    run_number = IntCol()

# tree = Tree('data', model=Event)
print Event
```

Branches are constructed according to the requested model:

```
tau1_eta -> FloatCol(default=-1111.0)
tau2_eta -> FloatCol(default=-1111.0)
tau1_phi -> FloatCol(default=-1111.0)
tau2_phi -> FloatCol(default=-1111.0)
tau1_pt -> FloatCol()
tau2_pt -> FloatCol()
tau1_m -> FloatCol()
tau2_m -> FloatCol()
tau1_numtrack -> IntCol()
tau2_numtrack -> IntCol()
event_number -> IntCol()
run_number -> IntCol()
```

Support for default values, automatic STL dictionaries, ROOT objects.

# Object Ownership and Garbage Collection

**PyROOT**

```python
from ROOT import TCanvas, TH1D

def make_plot():
    canvas = TCanvas('plot', 'plot',
                     700, 500)
    hist = TH1D('hist', 'plot',
                10, -3, 3)
    hist.FillRandom('gaus')
    hist.Draw()
    return canvas

canvas = make_plot()
# empty canvas since the histogram
# has been garbage collected!
canvas.Draw()
# hack to keep Python from exiting
# while the canvas is displayed
raw_input()
```

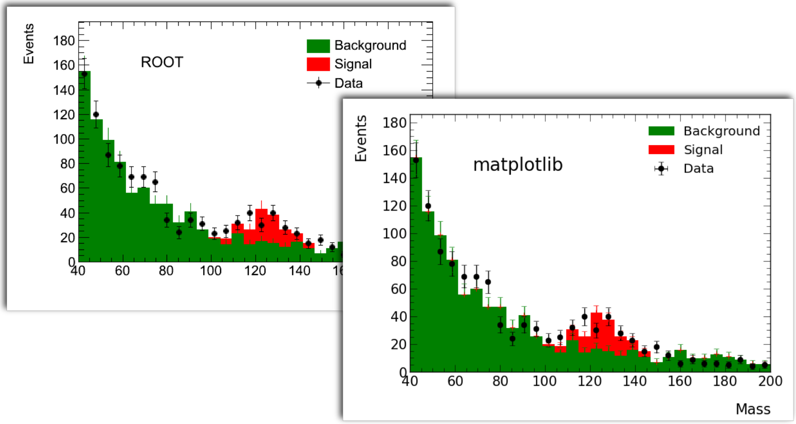
**rootpy**

```python
from rootpy.plotting import Canvas, Hist
from rootpy.interactive import wait

def make_plot():
    canvas = Canvas(700, 500)
    hist = Hist(10, -3, 3)
    hist.FillRandom('gaus')
    hist.Draw()
    return canvas

canvas = make_plot()
canvas.Draw()
wait()
```

Objects are kept alive as long as the Canvas is alive.
**See rootpy's keepalive function.**

# rootpy: Interface with matplotlib

ROOT histograms can be drawn with matplotlib via rootpy's matplotlib interface:



matplotlib: highly customizable, multiple backends, uses your latex engine.

# rootpy.ROOT: drop-in “import ROOT” replacement

- Harmonize access to ROOT and rootpy classes and take advantage of rootpy classes automatically.
- Replace `import ROOT` with `from rootpy import ROOT`
- Maintain backward compatibility with existing use of ROOT’s classes.
- ROOT classes are automatically cast to their rootpy form after the constructor in ROOT has been called:

```
>>> from rootpy import ROOT
>>> ROOT.TH1F('name', 'title', 10, 0, 1)
Hist('name')
```

- Also access rootpy classes under this same module without needing to remember where to import them from in rootpy:

```
>>> ROOT.Hist(10, 0, 1, name='name', type='F')
Hist('name')
```

## Future Objectives

- Larger documentation coverage and more examples. **This is important!**
- Full Python 3 support.
- Better integration with Jupyter Notebooks:
  - Full tab completion and helpful builtin commands like pylab
  - Fetch documentation on-demand for a class/method/function
  - Inline ROOT plots that are interactive

  Great to see progress in ROOTaaS and jsROOT
- Automatic wrapping of ROOT methods by parsing method signatures:
  - Automatic argument/return value conversion. If a method expects a TColor, rootpy can accept any matplotlib/ROOT color and convert it into a TColor before passing to the ROOT method.
  - Automatically create descriptors from ROOT get/setters
  - Reduce the amount of code in rootpy.
- rootpy version 1.0 will be the first "stable" release where we freeze the API and begin proper deprecation cycles if needed. **rootpy is already very close!**

## How do I install rootpy?

Install a released version with pip:

```
pip install --user rootpy
```

See the documentation for full instructions.