PyROOT
Automatic Python bindings for ROOT

Enric Tejedor, Stefan Wunsch, Guilherme Amadio for the ROOT team

PyHEP 2018
Sofia, Bulgaria

ROOT
Data Analysis Framework
https://root.cern
Introduction: What is PyROOT?

New Features

- In 6.14: Interoperability with Numpy
- Coming soon: PyRDataFrame

Experimental PyROOT

Future Plans

- Python 2 & Python 3
- User Pythonizations
- Cppy on Cling
Introduction
- Python bindings offered by ROOT
  - Created by Wim Lavrijsen
- Access all the ROOT C++ functionality from Python
  - Python façade, C++ performance
- Automatic, dynamic
  - No static wrapper generation
  - Dynamic python proxies for C++ entities
  - Lazy class/variable lookup
- Powered by the ROOT type system and Cling
  - Reflection information, JIT C++ compilation, execution
- Pythonizations
  - Make it simpler, more pythonic
Automatic bindings + Pythonizations

```python
import ROOT

f = ROOT.TFile('myfile.root')  # f is a (dynamic) Python proxy of a C++ object

t = f.mytree  # Pythonization: access tree as an attribute

for event in t:
    ...
```

TFile is a (dynamic) Python proxy of a C++ class

Pythonization: iterate over tree events in a Pythonic way
# C++ to Python Mapping

<table>
<thead>
<tr>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_types: short, int, long, float, double, std::string, char*, ...</td>
<td>int, [long], float, str</td>
</tr>
<tr>
<td>basic_type*, C-array</td>
<td>array (module)</td>
</tr>
<tr>
<td>class, template class</td>
<td>class, class generator</td>
</tr>
<tr>
<td>STL classes</td>
<td>std.vector, std.list, std.shared_ptr, ...</td>
</tr>
<tr>
<td>inheritance, dynamic_cast</td>
<td>inheritance, always final type</td>
</tr>
<tr>
<td>namespace</td>
<td>scope (dictionary)</td>
</tr>
<tr>
<td>pointer, reference</td>
<td>reference</td>
</tr>
<tr>
<td>exceptions</td>
<td>exceptions</td>
</tr>
</tbody>
</table>
The ROOT team has increased the effort in PyROOT

- We are aware of the importance of Python for HEP!

Main objective is to improve PyROOT in three ways:

1. Modernize PyROOT with a new implementation on top of Cppyy
2. Consolidate current PyROOT: add new features, fix issues
3. Support better interoperability with data science Python ecosystem (e.g. NumPy)
New Features in 6.14

ROOT
Data Analysis Framework
https://root.cern
Zero-copy C++ to NumPy array conversion

- Objects with contiguous data (std::vector, RVec)
- Pythonization: tell NumPy about data and shape

```python
import ROOT
import numpy as np

vec = ROOT.std.vector('int')(2)
arr = np.asarray(vec)  # zero-copy operation
vec[0], vec[1] = 1, 2

assert arr[0] == 1 and arr[1] == 2
```

New in 6.14
Memory adopted!
Read a TTree into a NumPy array

- Branches of arithmetic types

```python
myTree  # Contains branches x and y of type float

# Convert to numpy array and apply numpy methods
myArray = myTree.AsMatrix()
m = np.mean(myArray, axis = 0)

# Read only specific branches, specify output type
xAsInts = myTree.AsMatrix(columns = ['x'], dtype = 'int')
```
Forthcoming Features
Forthcoming Features

- **RDataFrame to NumPy**
  - All RDataFrame operations available
  - Implicit parallelism

```python
from ROOT import RDataframe

df = RDataframe('myTree', 'file.root')

# Apply cuts, define new columns
df = df.Filter('x > 0').Define('z', 'x*y')

np_arr = df.AsMatrix()
```

JITted C++ expression
Use Python callables in RDataFrame

- For Filter and Define operations
- Implementation with Numba to JIT Python code?

```python
df = RDataFrame('myTree', 'file.root')

df.Filter('x > 0')  # Already possible, jitted C++ expression

def my_cut(x):
    return x > 0

df.Filter(my_cut, ['x'])  # Uses Python callable
```
The New PyROOT
A new (experimental) PyROOT implementation is in the making
- Already available in ROOT master ([link](#))
- -Dpyroot_experimental=ON

Based on current Cppyy
- Set of packages for automatic Python-C++ binding generation
- Forked from PyROOT by Wim Lavrijsen

Goal: benefit from all the new features of Cppyy

ROOT-specific Pythonizations added on top
- A few available at the moment, more will come
The New Structure

PyROOT

User API

ROOT Pythonizations

Cppyy

Automatic Bindings: Proxy Creation, Type Conversion (Python/C API)

STL Pythonizations

ROOT & Cling

Reflection Info, Execution

ROOT Type System (TClass, TMethod, …)
Possible to use C++ lambdas from Python

```python
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine("auto mylambda = [](int i) { std::cout << i << std::endl; };")
140518947094560L
>>> ROOT.mylambda
<cppyy.gbl.function<void(int)>* object at 0x35f9570>
>>> ROOT.mylambda(2)
2
```
New PyROOT: Variadic Templates

- Support for variadic template arguments of functions

```python
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine("""template<typename... myTypes>
int f() { return sizeof...(myTypes); }
"""")
0L
>>> ROOT.f["int", "double", "void"]()
3
```
Future Plans

ROOT
Data Analysis Framework
https://root.cern
PyROOT supports both versions
- Also the new PyROOT

Not in our plans to discontinue support for Python2
- At least in the next few years
- However, end of life for Py2 is very close (2020)

Building ROOT: we will remove the limitation of one Python version per build
- If requested, PyROOT libraries will be generated for both Py2 and Py3
More on Pythonizations

- **User Pythonizations**: allow ROOT users to define pythonizations for their own classes
  - Lazily executed

  ```python
  @pythonization('MyCppClass')
  def my_pythonizor_function(klass):
    # Inject new behaviour in the class
    klass.some_attr = ...
  ```

- **Additional Python layer for extra features in Python**
  - “import ROOTExt”
Both current PyROOT and Cppyy rely on ROOT meta classes (TClass, TMethod, ...)
  - I.e. reflection data from ROOT

Not needed: Cppyy could be rebased on top of Cling
  - Use cling and its clang binding directly
  - Access a more powerful API
Summary

ROOT
Data Analysis Framework
https://root.cern
PyROOT’s automatic Python bindings: unique!

The ROOT team is aware of the growing importance of Python in HEP
- Dedicating more effort to PyROOT
- Our goal is to modernize PyROOT
  - Modern C++ with Cppyy, new features
- Pythonizations are key for usability
  - Being tracked for PyROOT experimental: [JIRA item](#)
- BOF Session @ CHEP: RDataFrame (Wed), join if interested: [link](#)
Backup Slides

ROOT
Data Analysis Framework
https://root.cern
Support for rvalue reference parameters

```python
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine('void myfunction(std::vector<int>&& v) {
    for (auto i : v) std::cout << i << " ";
}"
) 0L
>>> v = ROOT.std.vector['int'](range(10))
>>> ROOT.myfunction(ROOT.std.move(v))
0 1 2 3 4 5 6 7 8 9
>>> ROOT.myfunction(ROOT.std.vector['int'](range(10)))
0 1 2 3 4 5 6 7 8 9
```