Writing TTrees using uproot

Pratyush Das
Institute of Engineering and Management, India.

Jim Pivarski
Princeton University

My history with ROOT

Long non-physics interaction with ROOT.

2017 -

- Spark-ROOT: Viktor Khristenko - CERN OpenLab, Pratyush Das - Institute of Engineering and Management

  (unmaintained – replaced by laurelin?)

2018 -

Current and past DIANA fellows

Pratyush Das, Institute of Engineering and Management (Kolkata) [Undergrad]

- Topic: Add write functionality to uproot - proposal
- Mentor: Jim Pivarski, Princeton University
- Dates/Location: *summer, 2018 (FNAL)*

uproot is able to write TObjStrings and Histograms.

2019 -

Pratyush (Reik) Das
Institute of Engineering & Management (Kolkata)

Jun-Sep 2019

uproot is able to write TTrees with baskets containing flat data.

Building up to TTree writing

Some issues needed to be fixed/features needed to be added before TTree writing could be pursued -

scikit-hep/uproot Fixed xxhash import error ✓
#308 by reikdas was merged on Jul 30

scikit-hep/uproot Comparing checksum while reading lz4 compressed files ✓
#307 by reikdas was merged on Jul 25 • Changes requested

scikit-hep/uproot ROOT 6.18.00 streamers ✓
#301 by reikdas was merged on Jun 28

scikit-hep/uproot Added test to check if directory size is increased ✓
#298 by reikdas was merged on Jun 28

As well as improving histogram writing capability -

scikit-hep/uproot Bin labels can now be written ✓ !!!!
#305 by reikdas was merged on Jul 17 • Approved

scikit-hep/uproot Some Taxis attributes ✓
#297 by reikdas was merged on Jun 26

(only PRs during fellowship period shown)

Histogram fLabels

Writing bin labels of histograms to ROOT files was a big milestone towards beginning to write TTrees.

```
2638  ////////////////////////////////////////////////////////////////////////////////
2639  /// Write object to I/O buffer.
2640  ///
2641  /// This function assumes that the value in 'obj' is the value stored in
2642  /// a pointer to a "ptrClass". The actual type of the object pointed to
2643  /// can be any class derived from "ptrClass".
2644  /// Return:
2645  ///   - 0: failure
2646  ///   - 1: success
2647  ///   - 2: truncated success (i.e actual class is missing. Only ptrClass saved.)
2648
2649  Int_t TBufferFile::WriteObjectAny(const void *obj, const TClass *ptrClass)
```

Required re-implementing ROOT's WriteObjectAny() method – used everywhere in TTree writing.

Huge thanks to Sebastien Binet's Go-HEP, where he re-implemented ROOT's WriteObjectAny() for writing streamers – Drew inspiration from it.

2 phases to writing a TTree

- Declaration phase

```
t = uproot.newtree({"branch1": int,
                    "branch2": numpy.int32,
                    "branch3": uproot.newbranch(numpy.float64, title="This is the title")})
```

```
f = uproot.recreate("example.root")
f["t"] = tree
```

- Filling phase

```
f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [6, 7, 8, 9, 10]})
```

Interface - TTree writing

Basic usage:

[1]:
```
import uproot
import numpy

with uproot.recreate("example.root") as f:
    f["t"] = uproot.newtree({"branch": "int32"})
    f["t"].extend({"branch": numpy.array([1, 2, 3, 4, 5])})
```

Reading it back in uproot:

[2]:
```
f = uproot.open("example.root")
f["t"].array("branch")
```

[2]:
```
array([1, 2, 3, 4, 5], dtype=int32)
```

Reading it back in ROOT:

[3]:
```
import ROOT

f = ROOT.TFile.Open("example.root")
tree = f.Get("t")
treedata = tree.AsMatrix(["branch"])
print(treedata)
```

```
Welcome to JupyROOT 6.18/00
[[1.]
 [2.]
 [3.]
 [4.]
 [5.]]
```

Interface – TTree writing

This is how you create a TTree:

```
t = uproot.newtree({"branch1": int,
                    "branch2": numpy.int32,
                    "branch3": uproot.newbranch(numpy.float64, title="This is the title")})
```

uproot.newtree() takes a python dictionary as an argument, where the key is the name of the branch and the value is the branch object or type of branch.

```
branchdict = {"branch2": ">i8"}
```

We can specify the title, the flushsize and the compression while creating the tree.

This is an example of how you would add a title to your tree:

```
tree = uproot.newtree(branchdict, title="TTree Title")
```

Write it to the file -

```
f = uproot.recreate("example.root")
f["t"] = tree
```

Writing baskets - extend

We have 2 branches in our TTree:

- branch1
- branch2

The suggested interface of writing baskets to the TTree is using the extend method:

```
f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [6, 7, 8, 9, 10]})
```

The extend method takes a dictionary where the key is the name of the branch and the value of the dictionary is a numpy array or a list of data to be written to the branch.

Writing baskets - extend

We have 2 branches in our TTree:

- branch1
- branch2

The suggested interface of writing baskets to the TTree is using the extend method:

```
f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [6, 7, 8, 9, 10]})
```

The extend method takes a dictionary where the key is the name of the branch and the value of the dictionary is a numpy array or a list of data to be written to the branch. Remember to add equal number of entries to each branch, else you will run into an error.

```
f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [6, 7, 8, 9]})
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-18-6767510579cb> in <module>
----> 1 f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [6, 7, 8, 9]})

~/uproot/uproot/write/objects/TTree.py in extend(self, branchdict, flush)
    104         first = next(values)
    105         if all(len(first) == len(value) for value in values) == False:
--> 106             raise Exception("Baskets of all branches should have the same length")
    107
    108         if flush:

Exception: Baskets of all branches should have the same length
```

Writing baskets - flushing

You can specify a flush parameter to True or False in the extend method.

```
]: f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [6, 7, 8, 9, 10]}, flush=True)
```

By default, it is true. This means that these values are immediately flushed to the file.

```
f["t"].extend({"branch1": numpy.array([1, 2, 3, 4, 5]), "branch2": [1, 2, 3, 4, 5]}, flush=False)
```

You can choose not to flush the baskets immediately by setting flush = False.
The baskets are added to a buffer which are flushed to the file depending on the flush size set by the user.

The flush size can be set at the branch level and the tree level.

To set it at the branch level:

```
]: t = uproot.newbranch("int32", flushsize="10 KB")
```

and to set it at the tree level:

```
]: tree = uproot.newtree({"demoflush": t}, flushsize=3000)
```

As demonstrated above, one can specify the flush size in several ways:

- number B/KB/MB/GB. eg - 10 KB
- number of bytes eg - 3000

To use these flushsizes instead of flushing every basket directly to the file:

Writing baskets - append

You can also use the append function to add baskets to your file if you just need to add a single value to the end of your current basket buffer.

```
f["t"].append({"branch1": 1, "branch2": 2})
```

Make sure to add values to every branch, similar to the extend method, as well as just 1 value to each branch.

The append method does not provide a way to explicitly flush the data to the file, the data is added to the buffer and flushed when the flushsize is reached or the user calls extend with flush = True.

Writing baskets – Low level interface

If you want, you can write a basket to only 1 branch. But remember to add equal number of baskets to the other branches as well as ROOT assumes that all the branches have equal number of baskets and will not read the non-uniform baskets.

```
f["t"]["branch1"].newbasket([1, 2, 3])
```

Add 3 more basket data to branch2!

```
f["t"]["branch2"].newbasket([91, 92, 93])
```

The newbasket method does not(cannot) perform any checks on whether the number of basket data being added to the branches are equal in length or even if basket data is being added to all the branches.

Directly flushes the basket to the file.

Be careful when using this method!

Compression

Baskets are compressed according to the file compression, when no compression is specified.

```
branchdict = {"branch": "int32"}
tree = uproot.newtree(branchdict)
with uproot.recreate("example.root", compression=uproot.LZMA(5)) as f:
    f["t"] = tree
    f["t"].extend({"branch": [1]*1000})
```

Baskets compressed using LZMA with level 5.

Baskets are compressed according to the tree compression, when no Branch compression is specified.

```
branchdict = {"branch": "int32", "testbranch": "int64"}
tree = uproot.newtree(branchdict, compression=uproot.LZ4(4)) # <-- LOOK HERE
with uproot.recreate("example.root", compression=uproot.LZMA(5)) as f:
    f["t"] = tree
    f["t"].extend({"branch": [1]*1000, "testbranch": [2]*1000})
```

Baskets in “branch” and “testbranch” compressed using LZ4 with level 4.

Each Branch can have its own compression algorithm.

```
b1 = uproot.newbranch("i4", compression=uproot.ZLIB(5))
b2 = uproot.newbranch("i8", compression=uproot.LZMA(4))
b3 = uproot.newbranch("f4")

branchdict = {"branch1": b1, "branch2": b2, "branch3": b3}
tree = uproot.newtree(branchdict, compression=uproot.LZ4(4))
with uproot.recreate("example.root", compression=uproot.LZMA(5)) as f:
    f["t"] = tree
    f["t"].extend({"branch1": [1]*1000, "branch2": [2]*1000, "branch3": [3]*1000})
```

Baskets in “branch1” compressed using ZLIB with level 5.
Baskets in “branch2” compressed using LZMA with level 4.
Baskets in “branch3” compressed using LZ4 with level 4.

Close your file!

- extend(flush=False) and append() prone to loss of data if file is not properly closed – branch buffer not flushed.

```
f = uproot.recreate("example.root")
f["t"] = uproot.newtree({"branch": uproot.newbranch("i4", flushsize="1 MB")})
f["t"].extend({"branch": numpy.array([1, 2, 3, 4])}, flush=False)

f = uproot.open("example.root")
f["t"].array("branch")
```

```
array([], dtype=int32)
```

- newbasket() method is safe here – always flushes.
- Open your files using “with” – automatically closes your file.

The second way to close a file is to use the `with` statement:

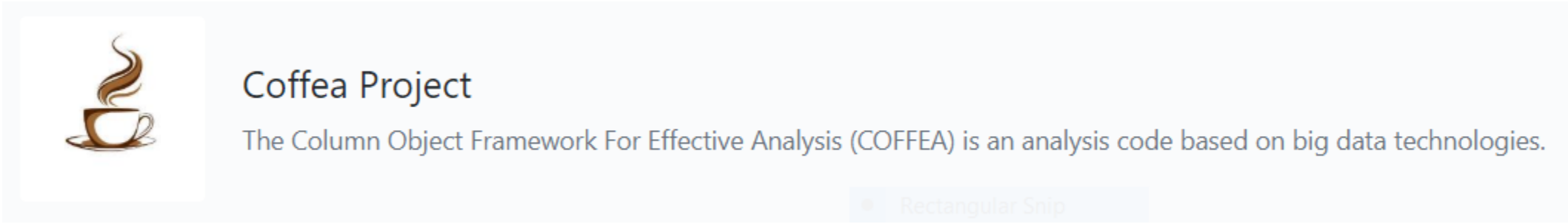
```
Python

with open('dog_breeds.txt') as reader:
    # Further file processing goes here
```

realpython.com

The `with` statement automatically takes care of closing the file once it leaves the `with` block,

Direct relevance - Coffea

Apart from users using uproot directly to write TTrees to ROOT files, some major projects depend on uproot and would directly benefit from the TTree writing features.



Optional output of an analysis facility.

Enables users to move in and out of the columnar ecosystem, letting them to buy-in to whichever format they are more comfortable with.

Future work

- There are some open issues related to writing ROOT files –

Unable to compress data larger than 2^24 bytes
#333 opened 3 days ago by reikdas

ROOT cannot append objects to empty files created by uproot
#316 opened 29 days ago by reikdas

Objects written by uproot and ROOT are read in a different "cycle" order by ROOT bug
#224 opened on Feb 8 by reikdas

Nested TDirectories writing-improvements
#138 opened on Sep 20, 2018 by jpivarski

More histogram, profile, and graph types writing-improvements
#137 opened on Sep 20, 2018 by jpivarski

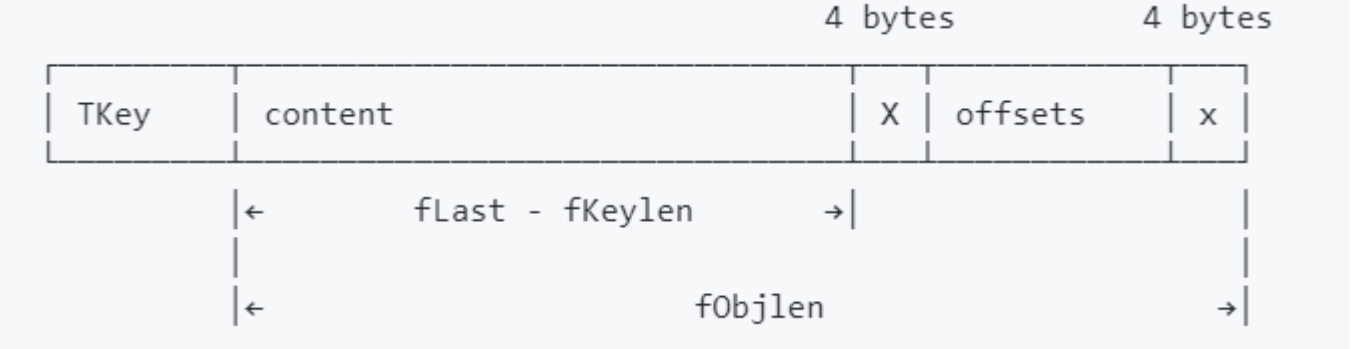
Block management writing-improvements
#135 opened on Sep 20, 2018 by jpivarski

- TTree baskets containing Jagged arrays.
- Writing TLorentzVectors? – Major project.

Jagged Arrays

Implementing baskets with Jagged Arrays is simply an extension of the current interface.

Serialization of Jagged Arrays is almost understood (by Jim) -



Offsets – 32 bit integers that indicate the starting byte of each entry.

Only thing left to learn – The 2 blocks of 4 bytes, marked x in the above diagram.

Ran out of time :(

At the end of the summer

- Project goal achieved.
- uproot writing is now a stable feature – No immediate improvements required.
- Jupyter Notebook tutorial of TTree writing published.
- 2 talks –
  - This one.
  - PyHEP (in October).
- 2 schools attended (personal improvement!) –
  - CoDaS-HEP at Princeton University.
  - FastML at Fermilab.

It has been a great summer for me!

BACKUP SLIDES

Read while writing

You can look at the current state of the TTree when you are putting in data:

```
import uproot

branchdict = {"branch": "int32"}
tree = uproot.newtree(branchdict)
f = uproot.recreate("example.root")
f["t"] = tree
f["t"].extend({"branch": numpy.array([1, 2, 3, 4, 5])})
```

Read it:

```
f["t"].array("branch")
```

```
array([1, 2, 3, 4, 5], dtype=int32)
```

You actually get all the reading methods -

```
for x in dir(f["t"]):
    if x.startswith("_") == False:
        print(x, end=", ")
```

```
allitems, allkeys, allvalues, append, array, arrays, clusters, extend, flush, get, items, iterate, iteritems, iterkeys, itervalues, keys, lazyarray, lazyarrays, matches,
mempartitions, name, numbranches, numentries, pandas, show, title, values,
```

```
for x in dir(f["t"]["branch"]):
    if x.startswith("_") == False:
        print(x, end=", ")
```

```
allitems, allkeys, allvalues, array, basket, basket_compressedbytes, basket_entrystart, basket_entrystop, basket_numentries, basket_numitems, basket_uncompressedbytes, ba
skets, compressedbytes, compressionratio, countbranch, countleaf, get, interpretation, items, iterate_baskets, iteritems, iterkeys, itervalues, keys, lazyarray, mempartit
ions, name, newbasket, numbaskets, numbranches, numentries, numitems, revertstring, title, uncompressedbytes, values,
```

Keep writing:

```
f["t"].extend({"branch": numpy.array([6, 7, 8, 9, 10])})
```

Close the file!

File writing robustness

- Different parts of a ROOT file have pointers to other parts of the file.
  - TKey -> File header
  - TKey -> TTree
  - TTree -> TBranch
  - TBranch -> TBasket
  - Directory -> TKey
  - Directory -> End of file
- While writing, file will always be a readable ROOT file.
- Operations performed in order to allow failed step (disk run out of space?) to not break ROOT file.