Writing files with uproot

Pratyush Das, Jim Pivarski

What is uproot?

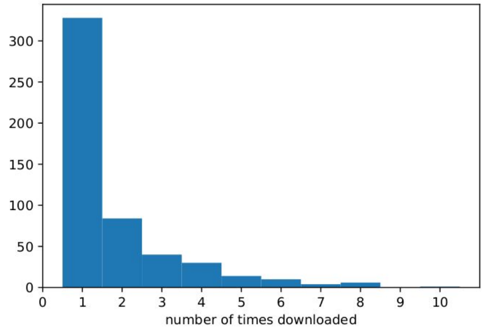
- Python reimplementation of ROOT I/O (until recently, only “input”).
- An array-centric view of ROOT TTree data:
    - branches of simple types are simple arrays
    - branches of complex types are “jagged arrays”
- High performance for large baskets, despite Python’s slowness (because all per-entry operations are performed in Numpy).
- Although originally intended as a temporary replacement for BulkIO, physicists like it for its simplicity: minimal installation, set-up, and affinity with machine learning interfaces.

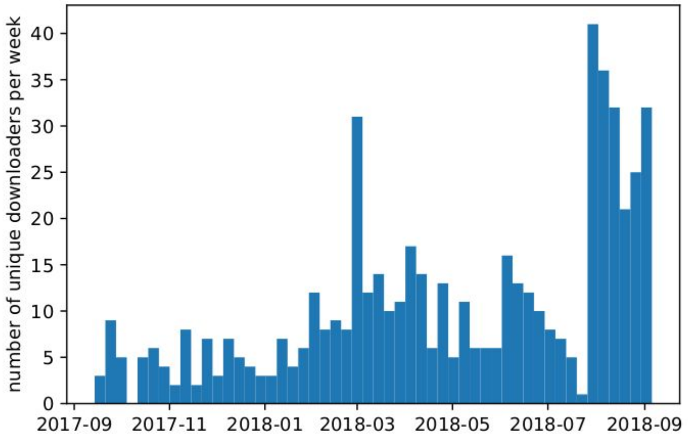
Download statistics

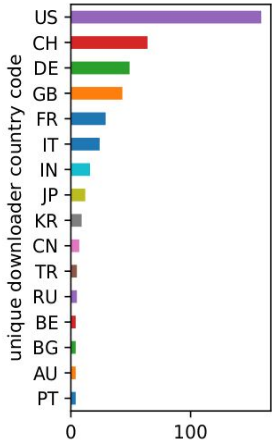
About **550** unique downloaders by pip since version 1.0 was released on September 14, 2017.

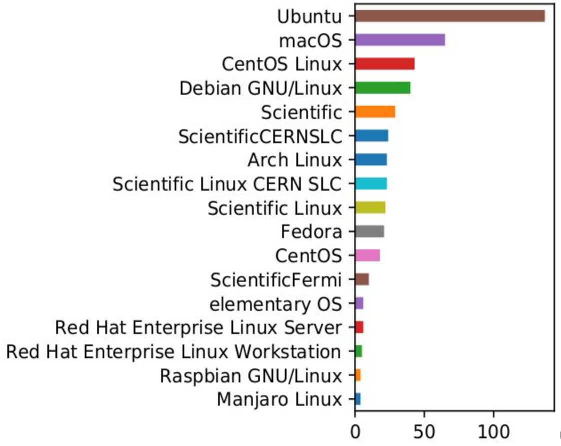
(Uniqueness estimated from country code, OS version, distribution, and release number.)

Primarily U.S. and Linux, but some Europe and MacOS.









uproot 3

- Non-I/O features split into new libraries
  - awkward-array: jagged array operations (Jim's talk)
  - uproot-methods: class methods like histograms and TLorentzVector

- Dependencies handled by pip
  - Numpy (1.13.1+, for awkward-array)
  - lz4: default ROOT compression
  - cachetools: uproot no longer defines its own cache dict

- **Write support**

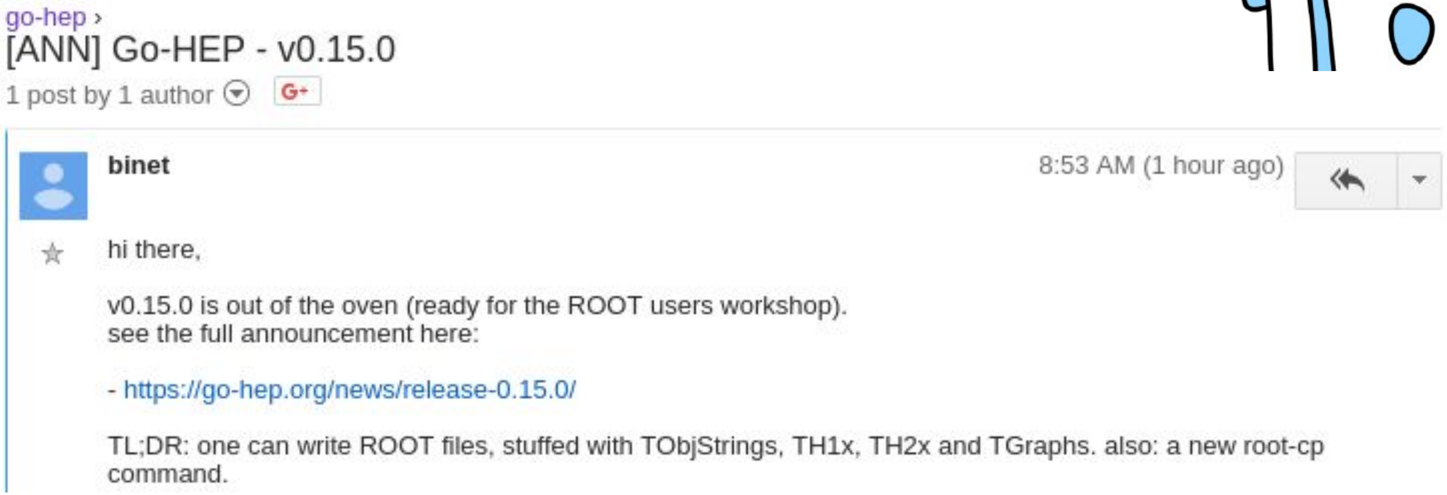
uproot 3 is currently in beta:

```
pip install -U "uproot>=3.0.0b2"
```

Go-HEP

Author - Sebastien Binet

- Similar to uproot, Go-HEP has a pure Go implementation of ROOT I/O (part of a larger ecosystem).
- As of version 0.15.0, Go-HEP can *write* to ROOT files.



go-hep ›
[ANN] Go-HEP - v0.15.0
1 post by 1 author

binet   8:53 AM (1 hour ago)

hi there,

v0.15.0 is out of the oven (ready for the ROOT users workshop).
see the full announcement here:

- https://go-hep.org/news/release-0.15.0/

TL;DR: one can write ROOT files, stuffed with TObjStrings, TH1x, TH2x and TGraphs. also: a new root-cp command.

Planned User interface

```
import uproot
import uproot_methods.classes.TH1
file = uproot.create("myfile.root")
h = uproot_methods.classes.TH1.hist(10, 3.0, 4.4, "th1f
title")
h.fill(numpyarray)
file["th1f"] = h
```

Scope and intended uses

Overturn the notion that uproot is to “escape” the ROOT ecosystem, make it a tool for interoperability in both directions.

However, still focus on the end-user physicist:

- Object types useful for analysis: histograms, simple TTrees, TLorentzVectors, and std::vectors for jaggedness.

  Not complex types like AODs.

- Optimize for “write once,” not “file as a database” or “partial write recovery.” Single-threaded writing.
- Simple dict-like interfaces wherever possible.

Design Choices

- Module structure
  - Reading and Writing are kept separate
- Standard file handle instead of memmap
- Streamer handling
  - Adding all streamers
- Growing buffer reallocation

Module structure

Different classes and different modules for reading and writing.

3 layers of abstraction -

1. Objects - Recognised by their methods(Duck typing) and written by a streamer for a single version of the class (corresponding to a recent ROOT release)
2. Primitives - Strings and numbers of particular types (Handled by the Cursor class)
3. Bytes - Interface with the physical layer, such as file handle and in the future xrootd (Handled by the Sink class - writing, Source class - reading)

Standard file handle vs memmap

Considered two different ways to write:

- memmap
  - Easy manipulation of data due to array like structure.
  - Takes a long time to resize - ROOT file size may continuously grow with the addition of new objects.
  - Work around - Grow file exponentially, starting at a high initial size. Disadvantage - Huge ROOT files.
- Standard file handle
  - Need separate pointers at each variable value location.
  - File resizing is ~100 times faster than when using memmap. (Tested)

uproot uses memmap to read files (no resizing, allows parallel reads) and a standard file handle for writing.

Streamers

Streamer interface for writing is different from reading.

Writer - Static view of streamers

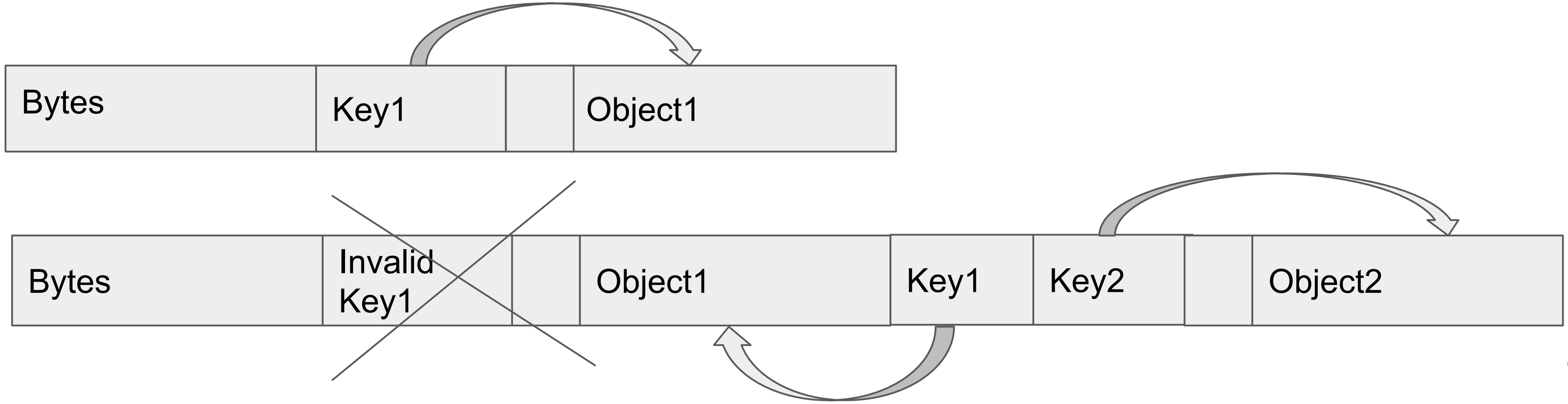
- Design - Writing each streamer as an uninterpreted byte string
  - Only supports a subset of ROOT’s own classes
  - Does not generate any new streamers
- Currently, writes a set of commonly used streamers as one big byte string
  - For simplicity
  - Classes supported - TObjString, TH*, TProfile*, simple TTree, TLorentzVector, TVector2, STL collections

Growing buffer reallocation

Common problem when objects in a fixed byte stream grow: what to do if they grow past their allocated bounds?

We create a new (larger) allocation, point the header to the new one, and consider the old one invalid. Unused space grows only logarithmically with #keys.

Example - TKeys in a TDirectory as objects are added:



Objects supported

The following objects can be written on a testing branch of uproot:

- TObjString - Basic object, used as a first test case
- TAxis - Also fairly simple, stepping up toward full histograms
- TH1F - Useful for physics; demonstrates that TH* will be possible

A usable suite of histograms will be available in early versions of uproot 3.0.x, with simple TTrees to follow.

First-stage TTrees will be flat (numerical types); following that, jagged arrays and object arrays like TLorentzVector will be supported.

Status and future plans

- Status
  - Restructuring codebase
  - Generalising histograms

- Future Plans
  - Compressed objects
  - Support all histogram types
  - Support simple TTrees
  - Generalize streamer writing