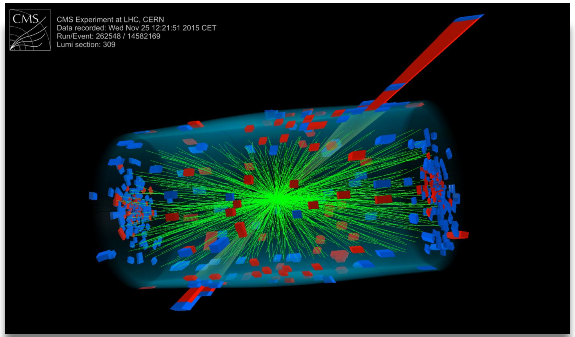
hepfile

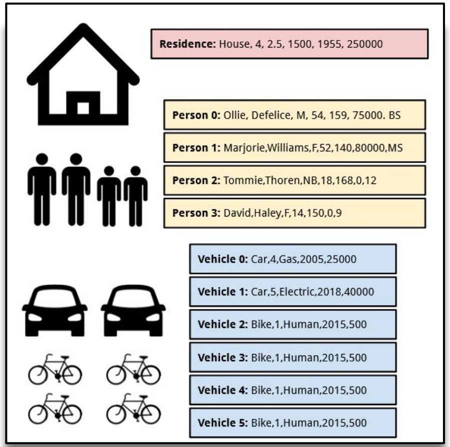
Matt Dreyer (Cornell)
Matt Bellis (Siena College)

What is the problem we're trying to solve?

- HEP data is heterogeneous and complicated!
- HEP-native example is particle physics experiment
  - Sensors find muons, electrons, jets, etc.
  - Each particle has specific data attached to it (momentum, charge, etc.)
  - Each **event** (collision) might have different numbers of these particles


CMS
CMS Experiment at LHC, CERN
Data recorded: Wed Nov 25 12:21:51 2015 CET
Run/Event: 262548 / 14582169
Lumi section: 309

- Consider a census of a town, with data gathered per household
  - Each **household** has people, cars, and place of residence
  - Each person has name, gender, and age
  - Each car has age and license plate
  - Each house has # of bedrooms and bathrooms



Residence: House, 4, 2.5, 1500, 1955, 250000

Person 0: Ollie, Defelice, M, 54, 159, 75000. BS

Person 1: Marjorie,Williams,F,52,140,80000,MS

Person 2: Tommie,Thoren,NB,18,168,0,12

Person 3: David,Haley,F,14,150,0,9

Vehicle 0: Car,4,Gas,2005,25000

Vehicle 1: Car,5,Electric,2018,40000

Vehicle 2: Bike,1,Human,2015,500

Vehicle 3: Bike,1,Human,2015,500

Vehicle 4: Bike,1,Human,2015,500

Vehicle 5: Bike,1,Human,2015,500

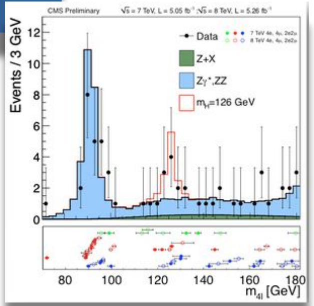
How to solve it

Current solutions: ROOT!

Problems: Monolithic

PyROOT helps but still tied to a file format that is bound to the analysis toolkit

Makes it difficult to interface with non-HEP people (e.g. broader computing community)

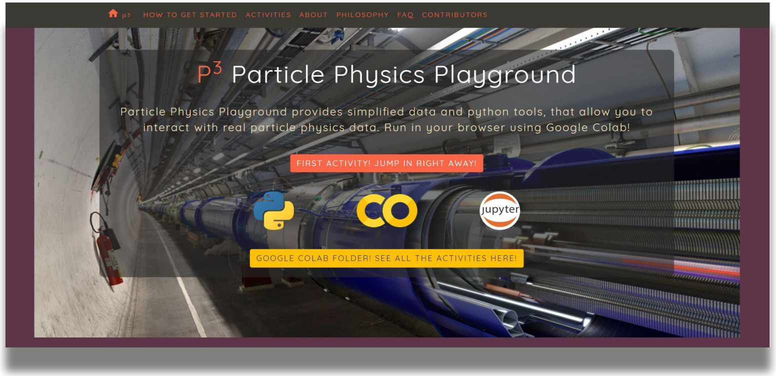


Alternative ROOT-file approaches: uproot/awkward ecosystem

But this is still using the ROOT file format (uproot).

Early attempt - h5hep

- Package originally called h5hep *(2017)*
- Based on organizing HEP-like data into HDF5 file format
    - Hierarchical Data Format 5
    - Open source
    - Datasets stored in group, stored in groups, etc.
- Used on Particle Physics Playground website for physics pedagogy
    - Data from CMS, BaBar, CLEO, and others
    - File format was originally zipped textfiles


p3
HOW TO GET STARTED
ACTIVITIES
ABOUT
PHILOSOPHY
FAQ
CONTRIBUTORS

P^3 Particle Physics Playground

Particle Physics Playground provides simplified data and python tools, that allow you to interact with real particle physics data. Run in your browser using Google Colab!

FIRST ACTIVITY! JUMP IN RIGHT AWAY!
CO
jupyter
GOOGLE COLAB FOLDER! SEE ALL THE ACTIVITIES HERE!

Image credit : https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5

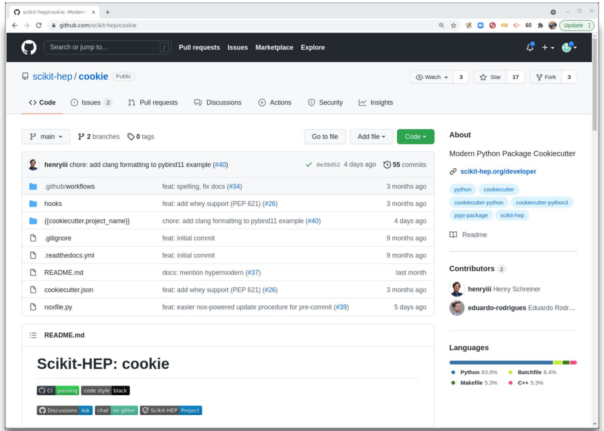
Enter Matt D!

What we had

- Basics of hepfile
- Could be pip installed
- Basics of ReadThe Docs

What we wanted to have

- Add some functionality
  - Hold strings
  - Add attributes
- Get documentation better
- Use Henry Schreiner's scikit-cookiecutter to make the package more robust for distribution
- Add functionality to work with awkward
- Make code more robust and fault-tolerant
- Add necessary unit tests
- Get CI working
- Submit to JOSS! *(Journal of Open-Source Software)*



scikit-hep / cookie

2 branches 0 tags

henryiii chore: add clang formatting to pybind11 example (#40) ✓ 0e39d52 4 days ago 55 commits

.github/workflows	feat: spelling, fix docs (#34)	3 months ago
hooks	feat: add whey support (PEP 621) (#26)	3 months ago
{{cookiecutter.project_name}}	chore: add clang formatting to pybind11 example (#40)	4 days ago
.gitignore	feat: initial commit	9 months ago
.readthedocs.yml	feat: initial commit	9 months ago
README.md	docs: mention hypermodern (#37)	last month
cookiecutter.json	feat: add whey support (PEP 621) (#26)	3 months ago
noxfile.py	feat: easier nox-powered update procedure for pre-commit (#39)	5 days ago

README.md

Scikit-HEP: cookie

About

Modern Python Package Cookiecutter

scikit-hep.org/developer

python cookiecutter cookiecutter-python cookiecutter-python3 pypi-package scikit-hep

Contributors 2

henryiii Henry Schreiner

eduardo-rodrigues Eduardo Rodr...

Languages

Python 83.0% Batchfile 6.4% Makefile 5.3% C++ 5.3%

hepfile is born!

h5hep → hepfile (Summer 2021)

- Define a *schema*
  - How data is organized
  - What metadata needs to be stored to organize the data
- Define minimal *useful* API for flexibility
- *Then*, implement it in
  - Python
  - HDF5
- Hierarchical Data Format 5
  - Open source
  - Groups → datasets
  - Singletons
  - Metadata
- Define structure of two python *dictionaries* to help with packing/unpacking data
  - Analogous to ROOT's TTree/Leaf/Branch

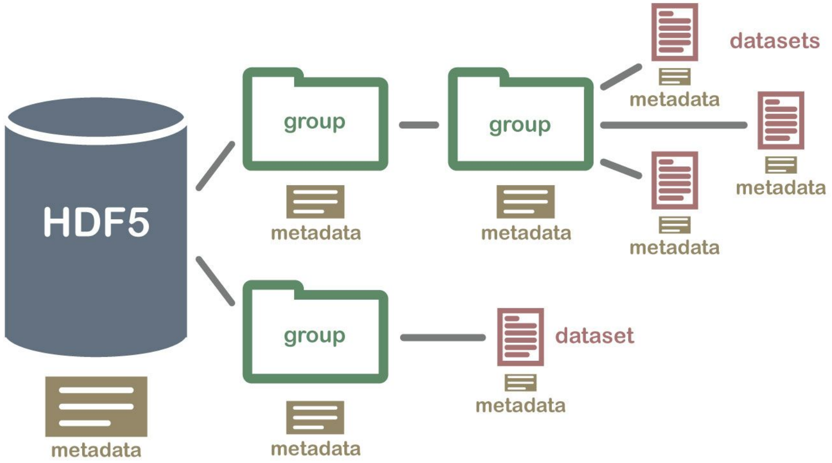
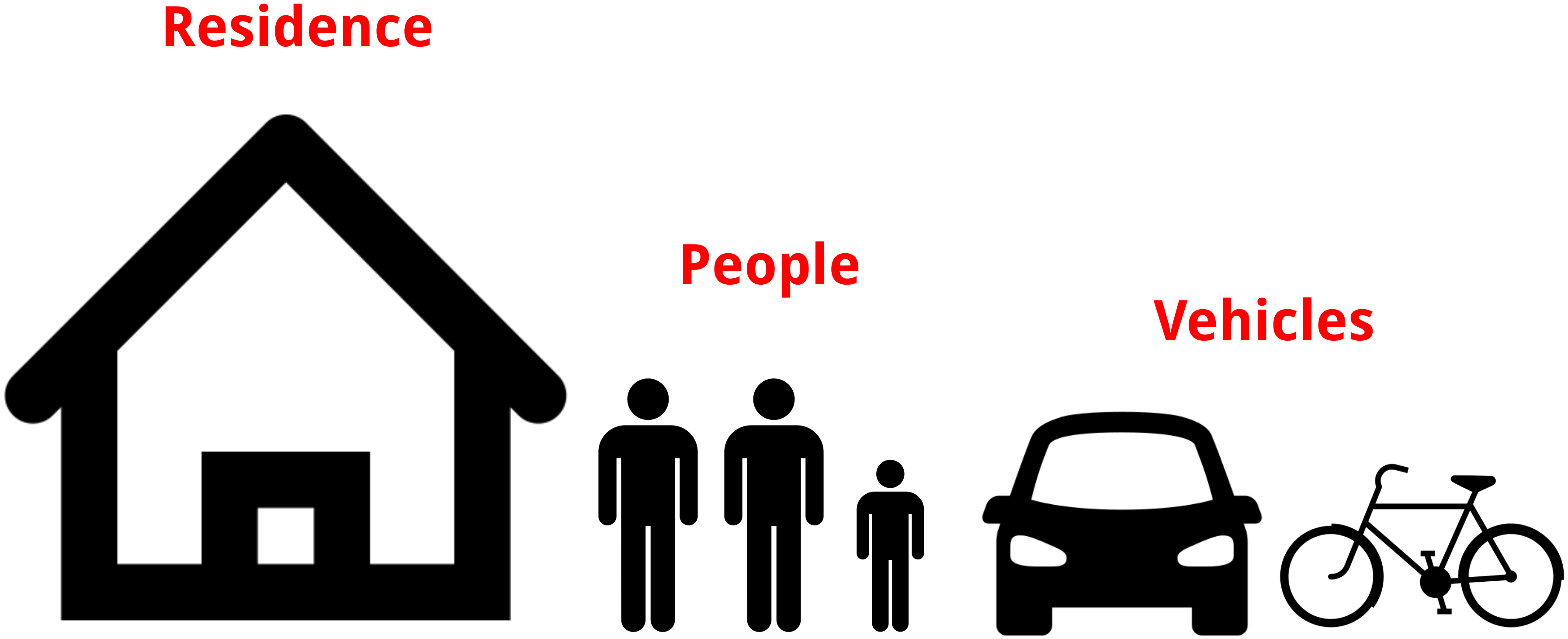


Image credit :
https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5

What is hepfile?

- Data organized into events/buckets
- hepfile groups data of similar types together in datasets
    - Keep bucket data using ‘counter’ field in dataset
- Pack takes buckets -> groups and datasets
- Unpack takes groups and datasets -> buckets
    - Extracts specific bucket ‘i’ from datasets


Residence
People
Vehicles

Residence: House, 4, 2.5, 1500, 1955, 250000

Person 0: Ollie, Defelice, M, 54, 159, 75000. BS

Person 1: Marjorie,Williams,F,52,140,80000,MS

Person 2: Tommie,Thoren,NB,18,168,0,12

Person 3: David,Haley,F,14,150,0,9

Vehicle 0: Car,4,Gas,2005,25000

Vehicle 1: Car,5,Electric,2018,40000

Vehicle 2: Bike,1,Human,2015,500

Vehicle 3: Bike,1,Human,2015,500

Vehicle 4: Bike,1,Human,2015,500

Vehicle 5: Bike,1,Human,2015,500

Residence: House, 4, 2.5, 1500, 1955, 250000

Person 0: Ollie, Defelice, M, 54, 159, 75000. BS

Person 1: Marjorie,Williams,F,52,140,80000,MS

Person 2: Tommie,Thoren,NB,18,168,0,12

Person 3: David,Haley,F,14,150,0,9

Vehicle 0: Car,4,Gas,2005,25000

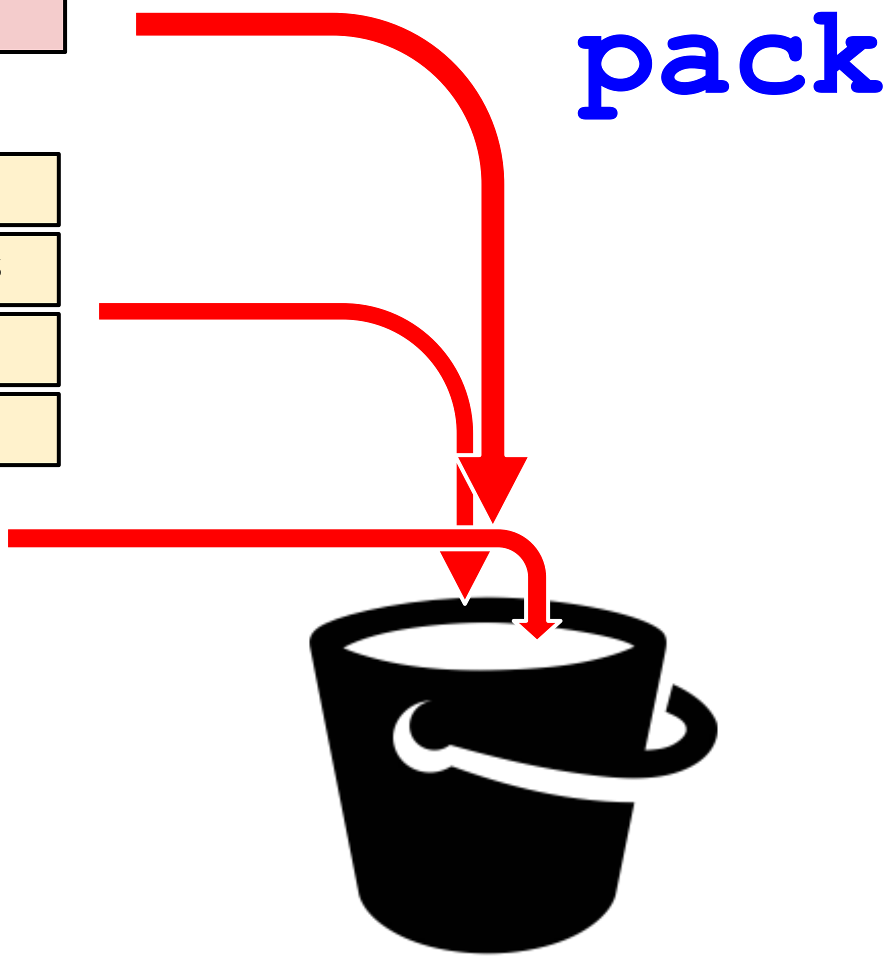
Vehicle 1: Car,5,Electric,2018,40000

Vehicle 2: Bike,1,Human,2015,500

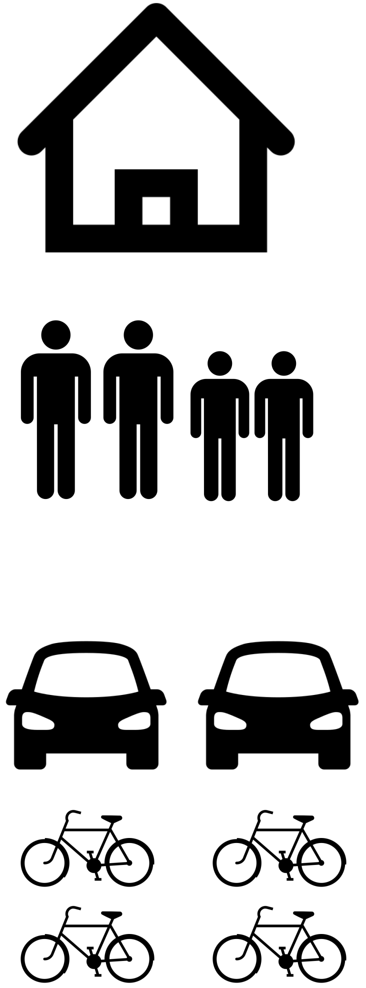
Vehicle 3: Bike,1,Human,2015,500

Vehicle 4: Bike,1,Human,2015,500

Vehicle 5: Bike,1,Human,2015,500

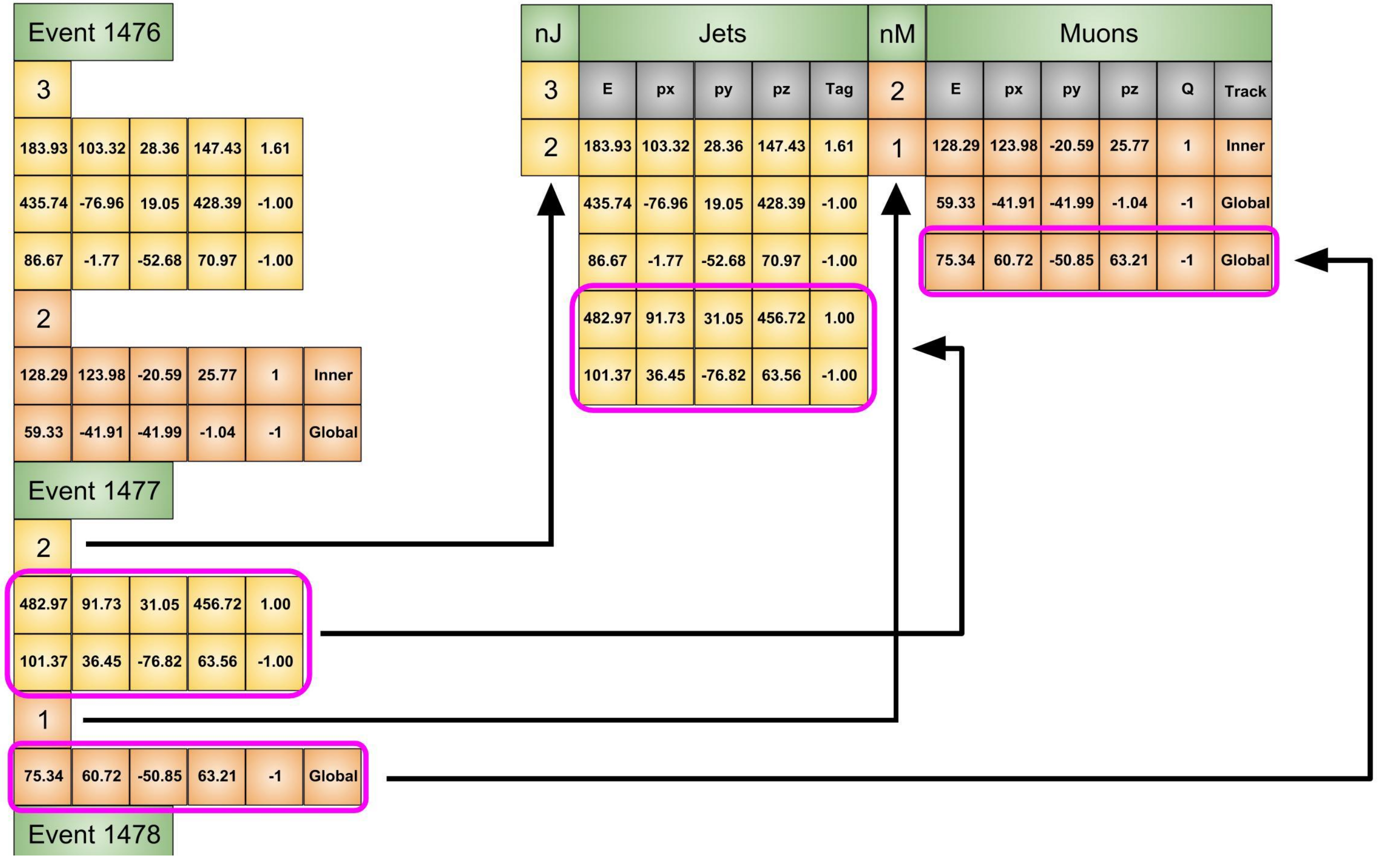


pack



Person 0: Ollie, Defelice,
Person 2: Tommie,Thoren,NB,
Vehicle 3: Bike,
Vehicle 4: Bike,1,Human
ike,1,Human,2015,500
1,Human,2015,500

hepfile data storage



Event 1476					
3					
183.93	103.32	28.36	147.43	1.61	
435.74	-76.96	19.05	428.39	-1.00	
86.67	-1.77	-52.68	70.97	-1.00	
2					
128.29	123.98	-20.59	25.77	1	Inner
59.33	-41.91	-41.99	-1.04	-1	Global
Event 1477					
2					
482.97	91.73	31.05	456.72	1.00	
101.37	36.45	-76.82	63.56	-1.00	
1					
75.34	60.72	-50.85	63.21	-1	Global
Event 1478					

nJ	Jets					nM	Muons					
3	E	px	py	pz	Tag	2	E	px	py	pz	Q	Track
2	183.93	103.32	28.36	147.43	1.61	1	128.29	123.98	-20.59	25.77	1	Inner
	435.74	-76.96	19.05	428.39	-1.00		59.33	-41.91	-41.99	-1.04	-1	Global
	86.67	-1.77	-52.68	70.97	-1.00		75.34	60.72	-50.85	63.21	-1	Global
	482.97	91.73	31.05	456.72	1.00							
	101.37	36.45	-76.82	63.56	-1.00							

hepfile - usage

```
hepfile.create_group(my_data, 'my_group', counter = 'my_counter')
```

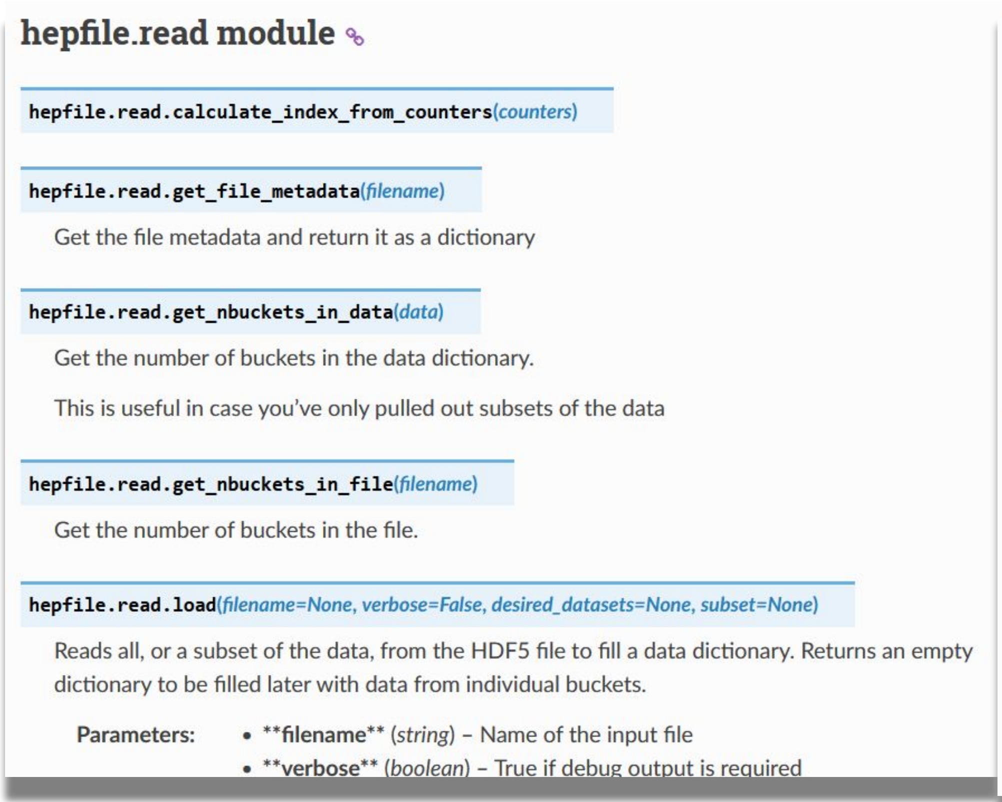
```
hepfile.create_dataset(my_data, 'my_dataset', group = 'my_group', dtype = str)
```

```
hepfile.create_dataset(my_data, ['data1', 'data2'] , group = 'my_group')
```

```
for i in range(5)
    my_bucket['my_group/my_dataset'] = 'yes'
    my_bucket['my_group/data1'] = 1.0
    my_bucket['my_group/data2'] = 2.0

my_bucket['my_unique'] = 3

hepfile.pack(my_data, my_bucket)
```

```
hepfile.write_to_file('my_file.hdf5', my_data)
```

hepfile - Read The Docs is getting populated!



hepfile.read module

`hepfile.read.calculate_index_from_counters`*(counters)*

`hepfile.read.get_file_metadata`*(filename)*

Get the file metadata and return it as a dictionary

`hepfile.read.get_nbuckets_in_data`*(data)*

Get the number of buckets in the data dictionary.

This is useful in case you've only pulled out subsets of the data

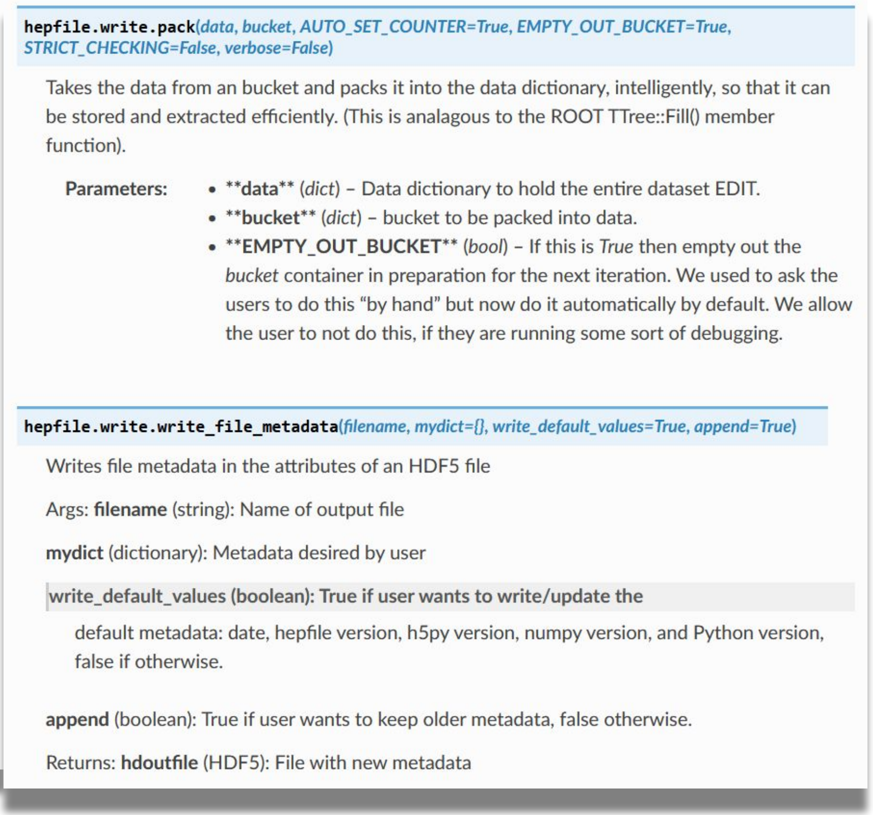
`hepfile.read.get_nbuckets_in_file`*(filename)*

Get the number of buckets in the file.

`hepfile.read.load`*(filename=None, verbose=False, desired_datasets=None, subset=None)*

Reads all, or a subset of the data, from the HDF5 file to fill a data dictionary. Returns an empty dictionary to be filled later with data from individual buckets.

Parameters:
- **filename** (*string*) – Name of the input file
- **verbose** (*boolean*) – True if debug output is required



`hepfile.write.pack`*(data, bucket, AUTO_SET_COUNTER=True, EMPTY_OUT_BUCKET=True, STRICT_CHECKING=False, verbose=False)*

Takes the data from an bucket and packs it into the data dictionary, intelligently, so that it can be stored and extracted efficiently. (This is analagous to the ROOT TTree::Fill() member function).

Parameters:
- **data** (*dict*) – Data dictionary to hold the entire dataset EDIT.
- **bucket** (*dict*) – bucket to be packed into data.
- **EMPTY_OUT_BUCKET** (*bool*) – If this is *True* then empty out the *bucket* container in preparation for the next iteration. We used to ask the users to do this “by hand” but now do it automatically by default. We allow the user to not do this, if they are running some sort of debugging.

`hepfile.write.write_file_metadata`*(filename, mydict={}, write_default_values=True, append=True)*

Writes file metadata in the attributes of an HDF5 file

Args: **filename** (string): Name of output file

mydict (dictionary): Metadata desired by user

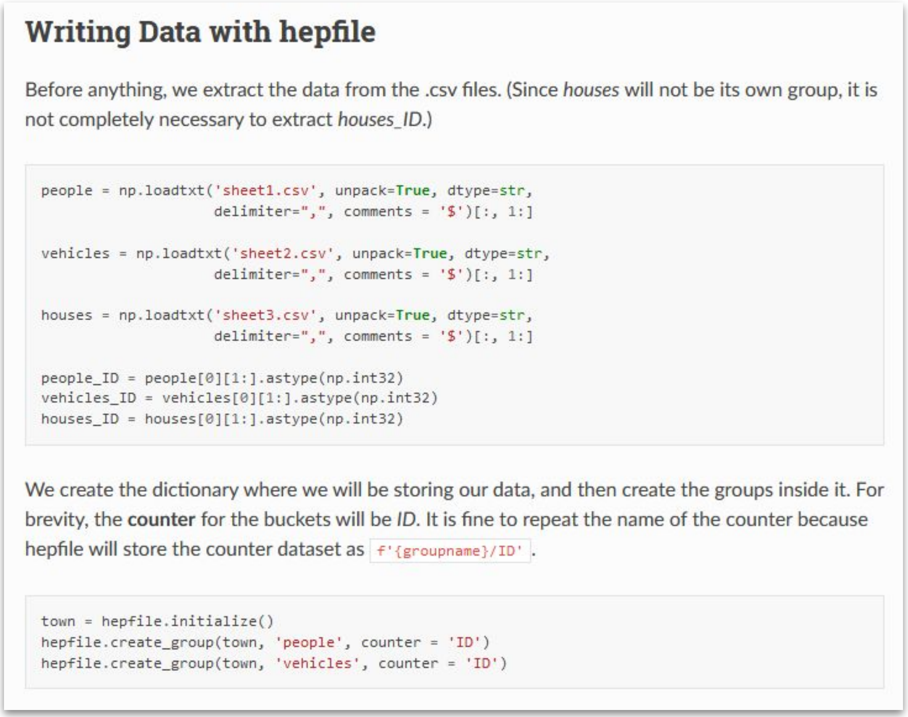
write_default_values (boolean): True if user wants to write/update the

default metadata: date, hepfile version, h5py version, numpy version, and Python version, false if otherwise.

append (boolean): True if user wants to keep older metadata, false otherwise.

Returns: **hdoutfile** (HDF5): File with new metadata

hepfile - Read The Docs is getting populated!



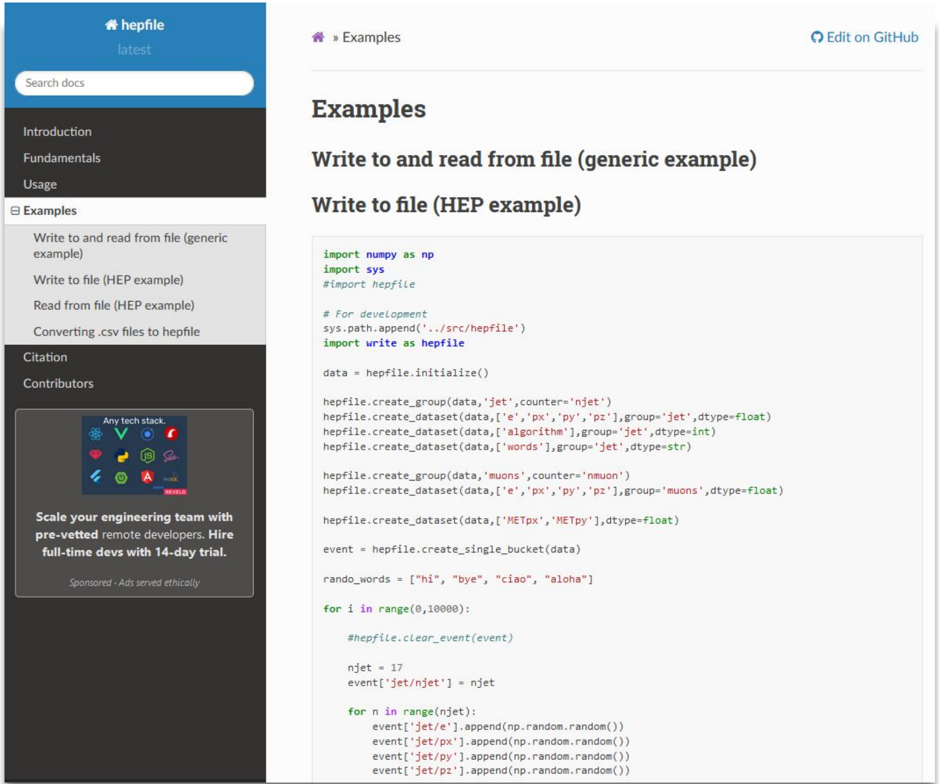
Writing Data with hepfile

Before anything, we extract the data from the .csv files. (Since *houses* will not be its own group, it is not completely necessary to extract *houses_ID*.)

```
people = np.loadtxt('sheet1.csv', unpack=True, dtype=str,
                    delimiter=",", comments = '$')[:, 1:]

vehicles = np.loadtxt('sheet2.csv', unpack=True, dtype=str,
                    delimiter=",", comments = '$')[:, 1:]

houses = np.loadtxt('sheet3.csv', unpack=True, dtype=str,
                    delimiter=",", comments = '$')[:, 1:]

people_ID = people[0][1:].astype(np.int32)
vehicles_ID = vehicles[0][1:].astype(np.int32)
houses_ID = houses[0][1:].astype(np.int32)
```

We create the dictionary where we will be storing our data, and then create the groups inside it. For brevity, the **counter** for the buckets will be *ID*. It is fine to repeat the name of the counter because hepfile will store the counter dataset as `f'{groupname}/ID'`.

```
town = hepfile.initialize()
hepfile.create_group(town, 'people', counter = 'ID')
hepfile.create_group(town, 'vehicles', counter = 'ID')
```


hepfile
latest
Search docs
Introduction
Fundamentals
Usage
⊟ Examples
Write to and read from file (generic example)
Write to file (HEP example)
Read from file (HEP example)
Converting .csv files to hepfile
Citation
Contributors
Any tech stack.
Scale your engineering team with pre-vetted remote developers. Hire full-time devs with 14-day trial.
Sponsored · Ads served ethically
» Examples
Edit on GitHub

Examples

Write to and read from file (generic example)

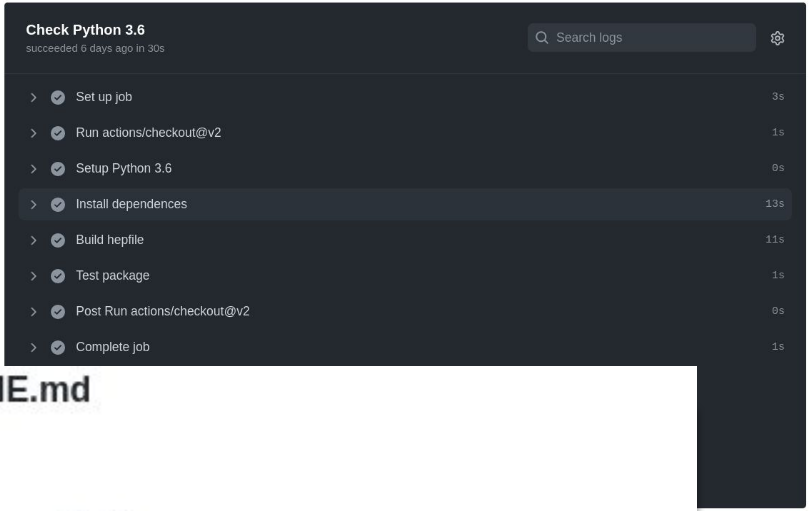
Write to file (HEP example)

```
import numpy as np
import sys
#import hepfile

# For development
sys.path.append('../src/hepfile')
import write as hepfile

data = hepfile.initialize()

hepfile.create_group(data,'jet',counter='njet')
hepfile.create_dataset(data,['e','px','py','pz'],group='jet',dtype=float)
hepfile.create_dataset(data,['algorithm'],group='jet',dtype=int)
hepfile.create_dataset(data,['words'],group='jet',dtype=str)

hepfile.create_group(data,'muons',counter='nmuon')
hepfile.create_dataset(data,['e','px','py','pz'],group='muons',dtype=float)

hepfile.create_dataset(data,['METpx','METpy'],dtype=float)

event = hepfile.create_single_bucket(data)

rando_words = ["hi", "bye", "ciao", "aloha"]

for i in range(0,10000):

    #hepfile.clear_event(event)

    njet = 17
    event['jet/njet'] = njet

    for n in range(njet):
        event['jet/e'].append(np.random.random())
        event['jet/px'].append(np.random.random())
        event['jet/py'].append(np.random.random())
        event['jet/pz'].append(np.random.random())
```

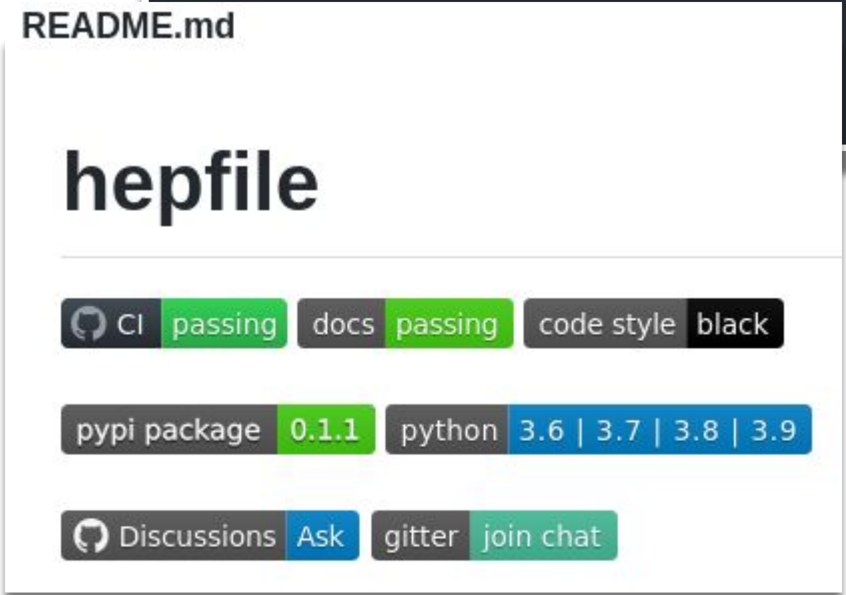
Unit tests

- *Unit tests* ensure functionality of the program and use cases are considered

- Continuous Integration
  - Evaluates whether every github commit keeps all unit tests working
  - Using **Github Actions** for this



```
Check Python 3.6
succeeded 6 days ago in 30s

> Set up job                        3s
> Run actions/checkout@v2           1s
> Setup Python 3.6                  0s
> Install dependences              13s
> Build hepfile                    11s
> Test package                      1s
> Post Run actions/checkout@v2      0s
> Complete job                      1s
```



README.md

hepfile

Installation

Local install and development

- Clone from Github
- https://github.com/mattbellis/hepfile

```
git clone https://github.com/mattbellis/hepfile
cd hepfile
flit install
```

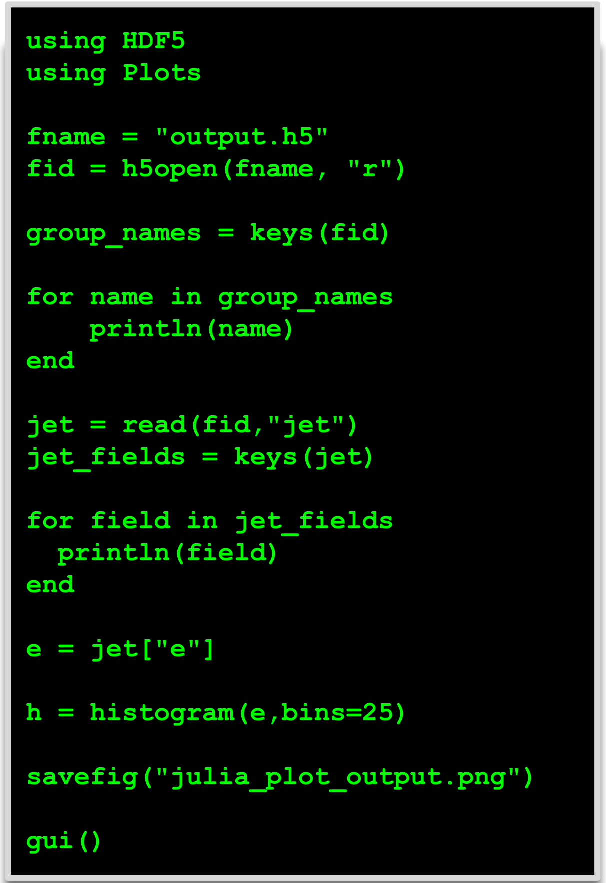
PyPI install

```
pip install hepfile
```

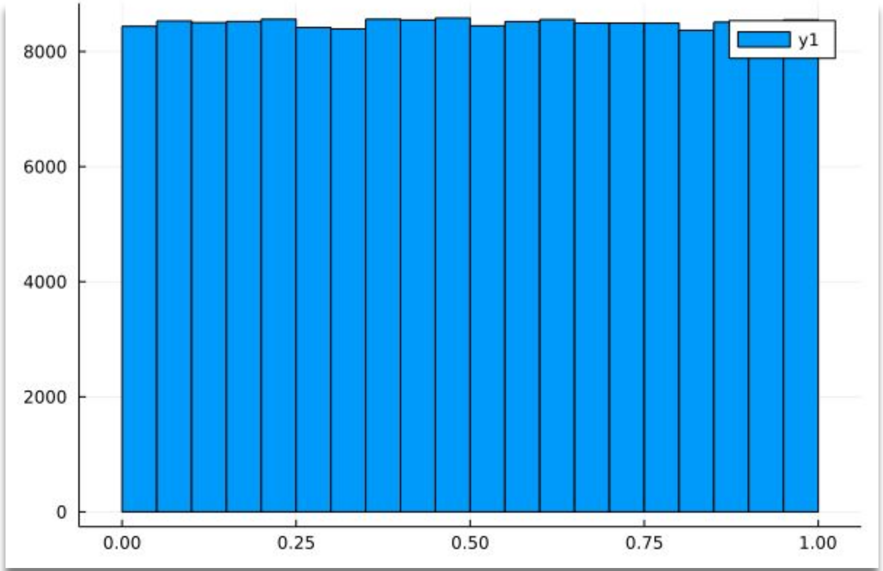
```
pip install hepfile[awkward]
```

In development

Julia



```
using HDF5
using Plots

fname = "output.h5"
fid = h5open(fname, "r")

group_names = keys(fid)

for name in group_names
    println(name)
end

jet = read(fid,"jet")
jet_fields = keys(jet)

for field in jet_fields
  println(field)
end

e = jet["e"]

h = histogram(e,bins=25)

savefig("julia_plot_output.png")

gui()
```



Because we chose a standard underlying file format (HDF5), it makes it easier for other languages to extract data from the file (*assuming there are HDF5 tools written already)*

JOSS

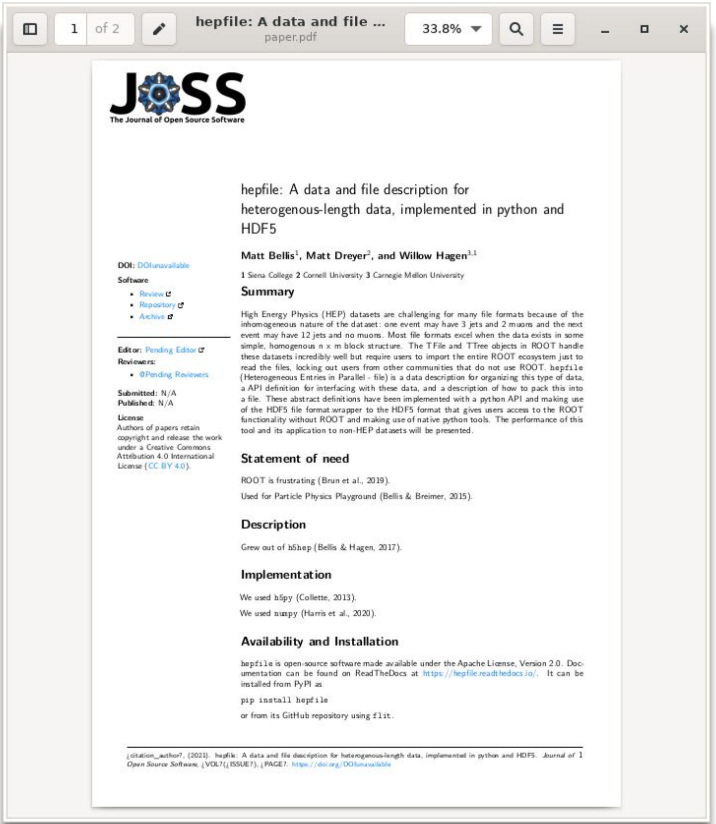
"The Journal of Open Source Software (JOSS) is an academic journal (ISSN 2475-9066) with a formal peer review process that is designed to improve the quality of the software submitted. Upon acceptance into JOSS, a Crossref DOI is minted and we list your paper on the JOSS website."

"JOSS submissions must:

- ✓ Be open source (i.e., have an OSI-approved license). Apache 2.0
- ✓ Have an obvious research application.
- ✓ Be feature-complete (no half-baked solutions) and be designed for maintainable extension (not one-off modifications).
- ✓ Minor 'utility' packages, including 'thin' API clients, and single-function packages are not acceptable."

Working on this!

Hope to finish over winter break (Bellis)



The Journal of Open Source Software

hepfile: A data and file description for heterogenous-length data, implemented in python and HDF5

Matt Bellis[1], Matt Dreyer[2], and Willow Hagen[3,1]

1 Siena College 2 Cornell University 3 Carnegie Mellon University

DOI: DOIunavailable

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Reviewers:

- @Pending Reviewers

Submitted: N/A
Published: N/A

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

High Energy Physics (HEP) datasets are challenging for many file formats because of the inhomogeneous nature of the dataset: one event may have 3 jets and 2 muons and the next event may have 12 jets and no muons. Most file formats excel when the data exists in some simple, homogenous n x m block structure. The TFile and TTree objects in ROOT handle these datasets incredibly well but require users to import the entire ROOT ecosystem just to read the files, locking out users from other communities that do not use ROOT. `hepfile` (Heterogeneous Entries in Parallel - file) is a data description for organizing this type of data, a API definition for interfacing with these data, and a description of how to pack this into a file. These abstract definitions have been implemented with a python API and making use of the HDF5 file format.wrapper to the HDF5 format that gives users access to the ROOT functionality without ROOT and making use of native python tools. The performance of this tool and its application to non-HEP datasets will be presented.

Statement of need

ROOT is frustrating (Brun et al., 2019).

Used for Particle Physics Playground (Bellis & Breimer, 2015).

Description

Grew out of `h5hep` (Bellis & Hagen, 2017).

Implementation

We used `h5py` (Collette, 2013).

We used `numpy` (Harris et al., 2020).

Availability and Installation

`hepfile` is open-source software made available under the Apache License, Version 2.0. Documentation can be found on ReadTheDocs at https://hepfile.readthedocs.io/. It can be installed from PyPI as

```
pip install hepfile
```

or from its GitHub repository using `flit`.

¿citation_author?, (2021). hepfile: A data and file description for heterogenous-length data, implemented in python and HDF5. *Journal of Open Source Software*, ¿VOL?(¿ISSUE?), ¿PAGE?. https://doi.org/DOIunavailable 1

Summary

My perspective on project:

- Learned a lot about package installation
- Learned about Continuous Integration and how useful it is
- Wrote unit tests for full coverage of actual project
- Wrote full documentation for functions
- Clearer perspective on how code should be written for professional use