

WebOOT

Exploring plots in the browser

Peter Waller

ROOT Users Workshop

12th March 2013

Outline

- What is weboot?
- How is it made?
- Where to go from here?

What is WebOOT?

It's an application that you can run
from your home directory in a minute or so:

Example

```
git clone git://github.com/rootpy/WebOOT
python WebOOT/setup.py develop --user
${HOME}/.local/bin/pserve --reload development.ini
```

(then visit <http://localhost:6543>)

What is WebOOT?

“How is it different from that AJAX ROOT interface?”

“How is it different from ROOT-js?”

“Why do I need this?”

What is WebOOT?

There is space for both server and client side applications:

- Client
 - highly responsive
 - can do fancy graphics
 - (probably) has spare CPU
- Server
 - has more knowledge
 - of other available plots
 - cache invalidation
 - already has complete implementation of ROOT

What is WebOOT?

There is space for both server and client side applications:

- Client
 - highly responsive
 - can do fancy graphics
 - (probably) has spare CPU
- Server
 - has more knowledge
 - of other available plots
 - cache invalidation
 - already has complete implementation of ROOT
 - Wouter, fancy implementing RooFitJS?

What is WebOOT?

- × “Souped up TBrowser”?
 - doesn't quite do it justice
- × Static gallery of histograms?
 - nope

What is WebOOT?

- × “Souped up TBrowser”?
 - doesn't quite do it justice
- × Static gallery of histograms?
 - nope
- ✓ Dynamic interface to ROOT files
 - helps you jump to the next thing you'll want to look at
 - make many plots in one go

What is WebOOT?

- × “Souped up TBrowser”?
 - doesn't quite do it justice
- × Static gallery of histograms?
 - nope
- ✓ Dynamic interface to ROOT files
 - helps you jump to the next thing you'll want to look at
 - make many plots in one go
- Science and Validation!
 - Look at same variables/histograms in different datasets and control regions
- Collaboration!
 - “Hey Bob, take a look at this..”
 - *sends bob a link*

quick demo

How does it work?

Basic idea : organization

- The best code is that which is never needs to be written..
- It's usually possible to organize yourself purely on the file system (or within a ROOT file)

Basic idea : organization

- The best code is that which is never needs to be written..
- It's usually possible to organize yourself purely on the file system (or within a ROOT file)
- Self describing files
 - Don't need to introduce information from elsewhere
 - i.e, little need for plotting scripts
 - no need to keep histogramming and plotting code in sync

Basic idea : traversal

WebOOT is based on the Pyramid¹ framework (related to pylons)
.. and here is a quick introduction to “traversal”:

¹<http://www.pylonsproject.org/>

Basic idea : traversal

WebOOT is based on the Pyramid¹ framework (related to pylons)
.. and here is a quick introduction to “traversal”:

The logical content of a URL is represented by an object:

weboot.cern.ch/ ~pwaller /sample.root/higgsmu
"URL fragment"



base [“~pwaller”] [“sample.root”] [“higgsmu”]

¹<http://www.pylonsproject.org/>

Basic idea : traversal

WebOOT is based on the Pyramid¹ framework (related to pylons)
.. and here is a quick introduction to “traversal”:

The logical content of a URL is represented by an object:

weboot.cern.ch/ ~pwaller /sample.root/higgsmu
"URL fragment"



base["~pwaller"] ["sample.root"] ["higgsmu"]

== [ROOT histogram wrapped by WebOOT class]

¹<http://www.pylonsproject.org/>

More traversal

```
base[“~pwaller”][“sample.root”][“higgsmu”]
```

This looks like a nested dictionary..

```
base = {  
  “~pwaller”: {  
    “sample.root”: {  
      “higgsmu”: RootHistogram(... TH1 somehow gets here ...)  
    }  
  }  
}
```

Object hierarchy

... except really they're objects which behave like dictionaries:

```
base = WebOOT({  
  “~pwaller” : FileSystemUserArea({  
    “sample.root” : RootFile({  
      “histogram” : RootHistogram(... TH1 somehow gets here ...)  
    })  
  })  
})
```

the contents of the dictionaries don't really exist.

Object hierarchy

... except really they're objects which behave like dictionaries:

```
base = WebOOT({  
  “~pwaller” : FileSystemUserArea({  
    “sample.root” : RootFile({  
      “histogram” : RootHistogram(... TH1 somehow gets here ...)  
    })  
  })  
})
```

the contents of the dictionaries don't really exist.

(they appear on demand, as if by magic)

Example of resources:

- Tree
- Histogram
- Directory

Given a resource, a view is chosen which can turn it into HTML or PNG or <format of your choice>

Actions

- ! is used to denote actions
- Actions are just python functions defined on resources with the @action decorator

Conceptually:

```
@action
def project(self, parent, key, axes):
    if axes == "x":
        p = self.o.ProjectionX()
    elif :
        p = self.o.ProjectionY()
    else:
        raise BadParameters("Expected x or y")
    return Histogram.from_parent(parent, key, p)
```

.. is called when myhistogram/!project/x/ is visited.

Jump bar

“What does this plot look like with a different set of cuts?”

/ -PWALLER / BROWSE / ALLROOT / ALL_PHS / 0_PRE_EVERYTHING /

Compose

1_pre_fiducial

2_pre_loose

3_post_loose

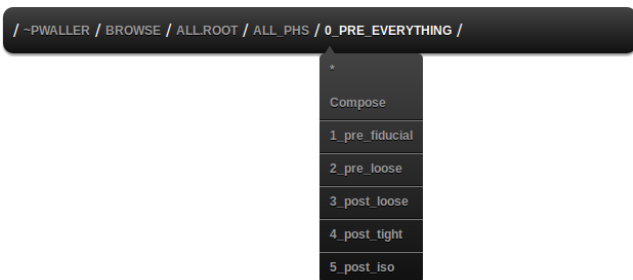
4_post_tight

5_post_iso

Each part of the URL has a dropdown containing valid places to jump to

Jump bar

“What does this plot look like with a different set of cuts?”



Each part of the URL has a dropdown containing valid places to jump to

- “What do these cuts look like on that other variable?”
- “What did this plot look like in the previous iteration of the analysis?”

Now it gets interesting...

`weboot.cern.ch/~pwaller/*.root/really_important_plot`

Now it gets interesting...

```
weboot.cern.ch/~pwaller/*.root/really_important_plot
```

This object is a “Multi-traverser”

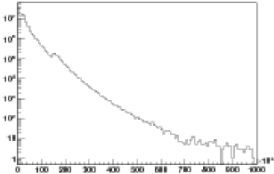
.. It abstractly represents a tree of objects of arbitrary dimension

Multi-traversing and wild cards

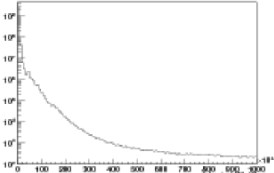
Model:

- 1 Make one (stacked) plot which looks how you want
- 2 Jump to other similar plots using the jump-bar
- 3 Now make the plots
 - for all samples, or
 - for all variables

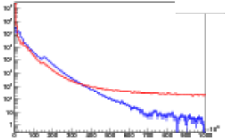
SampleA.root



SampleB.root



!compose/stack



Little tricks which make it work

- Caching
 - Information for jump-bar is computed once and reused
- `TCanvas.SaveAs(".eps")` is faster than `.png`, so `convert` is run.
 - Yields GIL, *faster*, better quality plots with antialiasing
 - Really shines on a multicore machine

`weboot.cern.ch?`

uses CERN's VMM infrastructure. Spinning up a machine is easy.

`weboot.cern.ch?`

uses CERN's VMM infrastructure. Spinning up a machine is easy.

Ideally, user can:

- Place their file on AFS: `~username/weboot/file.root`
- Set appropriate permissions:

```
fs sa weboot:atlas-viewable ~username/weboot  
(or, hypothetically, weboot:user-viewable to keep your super secret  
discovery to yourself)
```

- Visit <https://weboot.cern.ch/~username/file.root>
- See old plots, press F5, see new plots exactly aligned

Potential problems

- People often don't actually want to share data

Potential problems

- People often don't actually want to share data
.. (*except when they do want to*)

Potential problems

- People often don't actually want to share data
 - .. (*except when they do want to*)
- The URL isn't actually a great place to do things with semi-complicated syntax

Potential problems

- People often don't actually want to share data
 - .. (*except when they do want to*)
- The URL isn't actually a great place to do things with semi-complicated syntax
- Security, Stability and Scaling
 - sandboxing? LXC?
 - DOS?
 - ROOT crashes? (rare, but may want to isolate)
 - Many cored non-VM machines run like the wind
 - Many VMs, load balanced
 - Caching and cache invalidation
 - HTTP Proxy?
- UI could use some polish..

(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

- Easy to reuse histogram definitions in functions:

(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

- Easy to reuse histogram definitions in functions:

```
void plot(ObjectStore FS, LorentzVector v) {  
    FS.T<H1>("m")("Mass")(500,0,100,"m [GeV]").fill(v.m());  
    FS.T<H1>("pt")("p_T")(500,0,100,"pT [GeV]").fill(v.pt());  
}
```

(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

- Easy to reuse histogram definitions in functions:

```
void plot(ObjectStore FS, LorentzVector v) {  
    FS.T<H1>("m")("Mass")(500,0,100,"m [GeV]").fill(v.m());  
    FS.T<H1>("pt")("p_T")(500,0,100,"pT [GeV]").fill(v.pt());  
}  
plot(S("initial/muon0_"), mu0); // gives initial/muon0_pt
```

(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

- Easy to reuse histogram definitions in functions:

```
void plot(ObjectStore FS, LorentzVector v) {  
    FS.T<H1>("m")("Mass")(500,0,100,"m [GeV]").fill(v.m());  
    FS.T<H1>("pt")("p_T")(500,0,100,"pT [GeV]").fill(v.pt());  
}
```

```
plot(S("initial/muon0_"), mu0); // gives initial/muon0_pt
```

```
S.T<Cutflow>("cf").passed("initial"); // create cutflow
```

(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

- Easy to reuse histogram definitions in functions:

```
void plot(ObjectStore FS, LorentzVector v) {  
    FS.T<H1>("m")("Mass")(500,0,100,"m [GeV]").fill(v.m());  
    FS.T<H1>("pt")("p_T")(500,0,100,"pT [GeV]").fill(v.pt());  
}
```

```
plot(S("initial/muon0_"), mu0); // gives initial/muon0_pt
```

```
S.T<Cutflow>("cf").passed("initial"); // create cutflow
```

```
if (mu0.pt() < 20*GeV) return; // Cut
```


(sidenote) a4store : organize your histograms

One-line histogram & cutflow storage, initialization, and filling

```
S.T<H1>("m_ee")("ee-Mass")(500,0,100,"m_ee").fill(m);
```

- Easy to reuse histogram definitions in functions:

```
void plot(ObjectStore FS, LorentzVector v) {  
    FS.T<H1>("m")("Mass")(500,0,100,"m [GeV]").fill(v.m());  
    FS.T<H1>("pt")("p_T")(500,0,100,"pT [GeV]").fill(v.pt());  
}
```

```
plot(S("initial/muon0_"), mu0); // gives initial/muon0_pt
```

```
S.T<Cutflow>("cf").passed("initial"); // create cutflow
```

```
if (mu0.pt() < 20*GeV) return; // Cut
```

```
S.T<Cutflow>("cf").passed("20GeV");
```

```
plot(S("cut1/muon0_"), mu0);
```

- Highly efficient, **doesn't** use string comparisons for strings living in read-only memory (!)

²<https://github.com/a4/a4store/>

- Highly efficient, **doesn't** use string comparisons for strings living in read-only memory (!)
 - i.e, “text like this” uses pointer lookup in hashmap but not `std::string().c_str()`
 - cost is around 40 instructions per fill after initialization

²<https://github.com/a4/a4store/>

a4store performance

- Highly efficient, **doesn't** use string comparisons for strings living in read-only memory (!)
 - i.e, “text like this” uses pointer lookup in hashmap but not `std::string().c_str()`
 - cost is around 40 instructions per fill after initialization
- It's great for making many plots and doing systematic studies
- A version with ROOT histograms is not so hard to make²
 - (if there is interest)

²<https://github.com/a4/a4store/>

a4store performance

- Highly efficient, **doesn't** use string comparisons for strings living in read-only memory (!)
 - i.e, “text like this” uses pointer lookup in hashmap but not `std::string().c_str()`
 - cost is around 40 instructions per fill after initialization
- It's great for making many plots and doing systematic studies
- A version with ROOT histograms is not so hard to make²
 - (if there is interest)
- Disadvantage:
 - not so great for compile time

²<https://github.com/a4/a4store/>

- Now a part of the rootpy organization, so bus number > 1
 - <https://github.com/rootpy/WebOOT>
- Feel free to steal whatever is worth stealing!
- Let me know if you encounter problems on the [github issue tracker](#)
- Platform as a service seems like a neat idea - no friction for users
- I'd love to hear from you, come and find me