namespace root {}

G.Barrand, CNRS/IN2P3/LAL

A **smooth** migration/**refactoring** plan for **CERN-ROOT** to have, at least and last, around 2040, the histogram class for HEP that any **"doer"** expects.

Since now twenty years that HEP is doing C++, it is a little bit strange that this community can't expose an histogram class that encodes **"what is an histogram" only**. And this in the same way/spirit that in the STL we can find a string class that handles only what someone expects from a string class : handle a string! or that in the STL, we can find a vector class that encodes only "what is a vector".

Then where is the "hep::histogram" class ?

This is especially shocking from a **doer** point of view, knowing that in HEP, statistical analysis, due to the nature of the physics that is done here is at the core of things (due to the uncertainty principle, grounding probabilities, quantum mechanics and all that).

Common practice of a **ROOT doer**, let us look at the **TH1** :

```
class TH : <inheritance not related to "histograming only"> {
  <fields : arrays representing bin contents> // ok with that.
  void fill() { <fill arrays representing bin contents> } // ok with that.
// but :
  <a hell of other things not related to "histogramming" only>
  <in particular auto management of an histo in some "gDirectory">
  <connection to a graphics service through a global "gPad",>
  <management in a introspection system, through a ClassDef cpp macro,
   for an interpreter>
};
```

And what if a **non** ROOT doer wants to manage an histo in something else? Or plots with another graphics system? Or manipulates it with another interpreter (for exa Python)? and wants to do that without having to compile/build/link a lot of things not related to the "else services" he wants to use:

He can't !

In CERN-ROOT, the same pattern happens for a lot of things where the **"spirit code"** of a problem is not put in a standalone code. Another example: I needed spline in my graphics, but had not been able to find the **right piece of code** handling "just that" in C++ on the web. Finally I discovered that the encoding of spline formulas exists in TSpline::BuildCoeff, **but, alas!**, immersed in a lot of things not related to the "spline only" problem :

```
class TSpline : <inheritance not related to "spline only"> {
      BuildCoeff() { <great! Here the "spirit code" of defining a spline> }  //I want that!
      <a hell of other things not related to "spline" only>  // ☹
};
```

What I have done finally in my **inlib/spline** file :

```
#include <STL things: vector, cmath>
namespace inlib  { namespace spline {
  class base_poly, class cubic_poly, quintic_poly,   // Take the "spline spirit code" of TSpline
  class base_spline, cubic, quintic                  //  only, and have that in inlib::spline.
}}                                                   // (Copyright respected).
```

=> standalone, **no singletons**, **strongly OO, pure header, highly portable**, and graphics done in another class that "uses" inlib::spline : **Doers want that !**

The **grand plan** proposal :

```
class TXxx : <inheritance not related to "Xxx only"> {
  <code related to the Xxx problem only>
  <code not related to Xxx only>
};
```

→

```
namespace root [
class xxx  : <inheritance related to xxx only> {
  <code related to the xxx problem only>
}; }
```

```
class TXxx  : public root::xxx {
  <code not related to xxx only, as auto management
  and implicit relationship to various services>
};
```

```
namespace root { class xxx {};}
namespace root {class yyy {};}
namespace root {class zzz {};}
```

→ **root lib : STL style (more readable), no singletons, layered, highly portable, strongly OO!**
Apply that also to services : **I/O, graphics**, etc...

→ **Promote this lib to build apps.**

```
class TXxx {};
class TYyy {};
and the "gService" logic
```

→ **Deprecate** this way of doing and related code.

My softinex, inlib, exlib, g4tools and various apps show the way. See pages and code at :
https://gbarrand.github.io https://github.com/gbarrand

Having such a **backend "namespace root" lib** would be a great engineering goal around analysis tools (and everything in fact) for the **HSF**! This effort would be a long standing one, and only a long standing structure, at the level of HEP, would be able to handle it.

23nd International Conference on Computing in High Energy and Nuclear Physics, July 9-13 2018, Sofia