PROOF tutorial
I/O Basics

Gerardo Ganis, CERN, PH-SFT
gerardo.ganis@cern.ch

The goals of the I/O system

- Be able to save objects in a simple and generic way
- Be able to read back the objects
  - On any platform
  - Efficiently
  - With a different version of the program
- Provided by the language for basics types

```
// Write to output file
fprintf(fout, “%d %lld”, aint, alonglong);

// Read from input file
sscanf(fin, “%d %lld”, &aint, &alonglong);
```

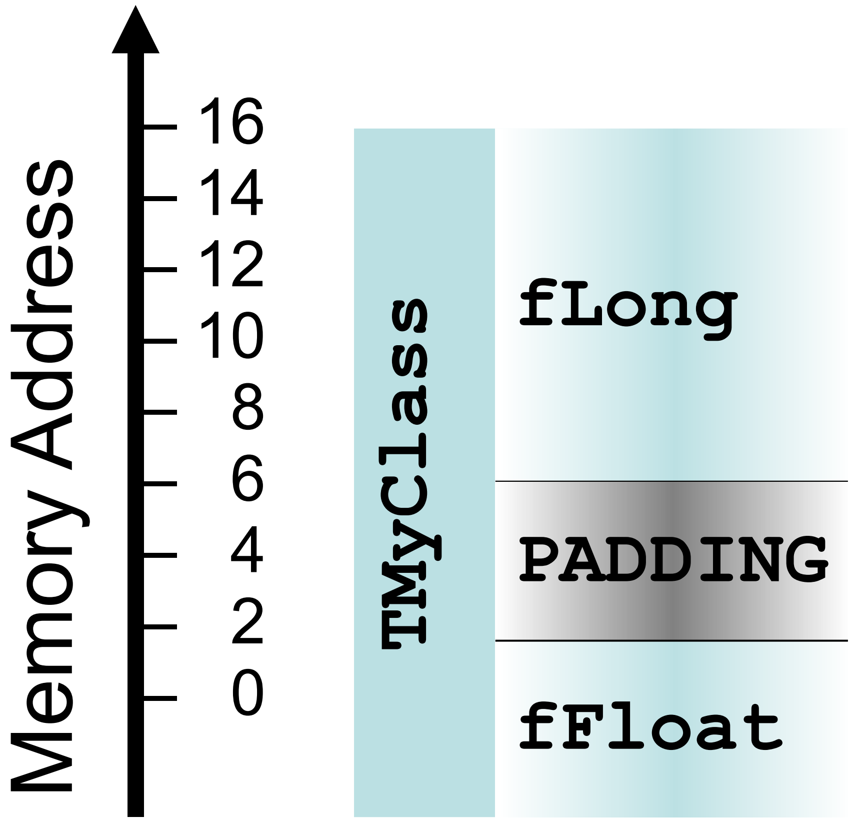
- Not available by default for new types (classes)

Streaming and Reflection

- I/O requires streamers
  - Serializing the object to store it into a file
  - Rebuild the object from the file info
- Streamers can be complicated beasts
  - Data members may be also complex types and the streamers need to take care of all of that
- Streamers need reflection, i.e. to know
  - The types of the data members
  - The base class
  - Where they are
- Reflection not (yet) available in ISO C++

Reflection

TMyClass is a class



```
class TMyClass {
   float fFloat;
   Long64_t fLong;
};
```

"fFloat", 4 bytes, is at offset 0

"fLong", 8 bytes, is at offset 8

Reflection not (yet) available in ISO C++

Dictionaries: adding reflection to C++

CINT can generate dictionaries, i.e. reflection information

Just needs the class header files

```
rootcint -f MyClassDict.cxx TMyClass.h LinkDef.h
```

Collects reflection data for types requested in Linkdef.h

Stores it in MyClassDict.cxx (dictionary file)

Compile MyClassDict.cxx, link, load: C++ with reflection!

Reflection by selection

LinkDef.h syntax:

```
#pragma link C++ class MyClass+;
#pragma link C++ typedef MyType_t;
#pragma link C++ function MyFunc(int);
#pragma link C++ enum MyEnum;
```

Dictionaries are also created by ACLiC

Can simply use ACLiC:

```
.L MyCode.cxx+
```

Will create a library MyCode_cxx.so with dictionary of all types in MyCode.cxx automatically!

TFile

ROOT stores objects in ROOT files described by TFile:

```
TFile* f = new TFile("afile.root", "NEW");
```

Options:
"READ" (default): open the file in read mode
"NEW" or "CREATE": create a new file
"RECREATE": create a new file, overwrite existing one
"UPDATE": open a file in update mode

TFile behaves like file system:

```
f->mkdir("dir");
```

TFile has a current directory:

```
f->cd("dir");
```

Saving objects in TFile

Once the dictionary is available, an object deriving from TObject can be written to the file, with default name

```
root [] f->cd()
root [] object->Write()
```

or changing the name to "newName"

```
root [] object->Write("newName")
```

Alternative way:

```
f->WriteObject(object, "name");
```

TFile / TDirectory

- A TFile object may be divided in a hierarchy of directories, like a Unix file system.
- Two I/O modes are supported
  - Key-mode (TKey): objects identified by a name (key), like files in a Unix directory
    - OK up to a few thousand objects
      - Histograms, geometries, mag fields, etc.
  - TTree-mode to store event data
    - The number of events may be millions, billions.

Name Cycles

Create snapshots regularly:

MyObject;1

MyObject;2

MyObject;3

…

MyObject

Write() does not replace but append!
but see documentation TObject::Write()

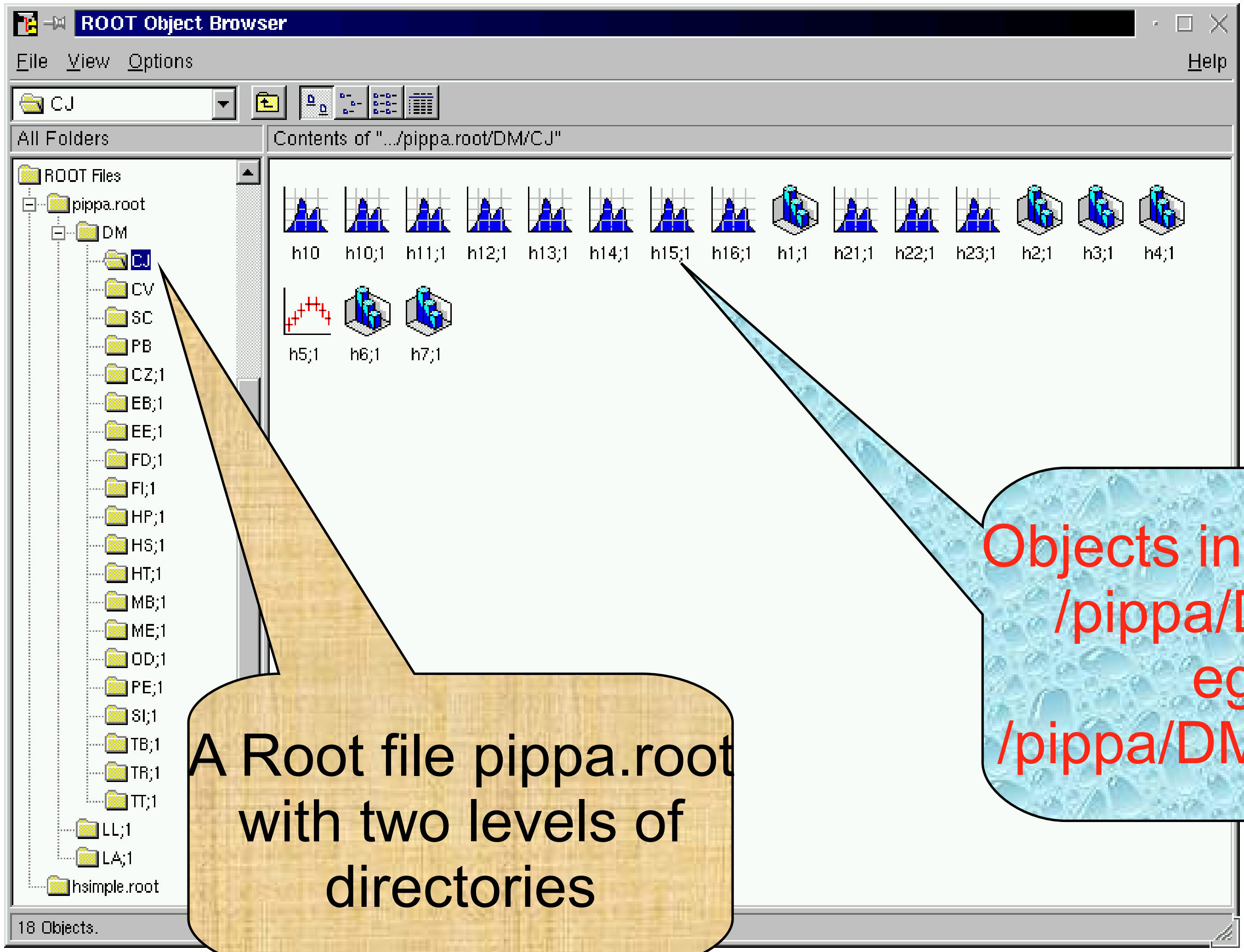
Use object->Write("name", TObject::kOverwrite) to remove old snapshots

Self-describing files

- Relevant streamer information (dictionary) for persistent classes written to the file
- ROOT files can be read by foreign readers
  - Support for Backward compatibility
  - Files created in 2001 must be readable in 2015
- Classes (data objects) for all objects in a file can be regenerated via TFile::MakeProject

```
root [] TFile f(“demo.root”);
root [] f.MakeProject(“dir”,”*”,”new++”);
```

Browsing files: TBrowser



A Root file pippa.root with two levels of directories

Objects in directory /pippa/DM/CJ eg: /pippa/DM/CJ/h15

Try to write the TGraphErrors object

- Open a file with new TFile

```
root [] TFile* f = new TFile("afb.root", “NEW”)
```

- Write the TGraphErrors object

- Check the file content before and after writing the object with TFile::ls()

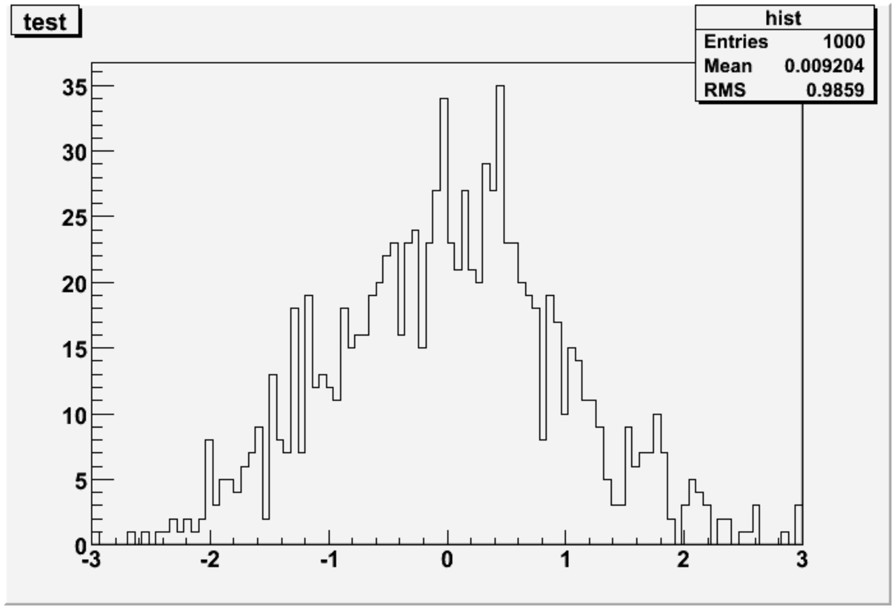
```
root [] f->ls()
```

Example of key mode

```
void keyWrite() {
   TFile f("keymode.root","new");
   TH1F h("hist","test",100,-3,3);
   h.FillRandom("gaus",1000);
   h.Write()
}
```

```
void keyRead() {
   TFile f("keymode.root");
   TH1F *h = (TH1F*)f.Get("hist");
   h.Draw();
}
```

exercises/keyMode.C



Where his my histogram?

TFile owns histograms (due to historical reasons):

```
TFile* f = new TFile("myfile.root");
TH1F* h = new TH1F("h","h",30,-3.,3.);
h->FillRandom(“gaus”);
h->Draw();
h->Write();
Canvas* c = new TCanvas();
c->Write();
delete f;
```

Histograms automatically deleted: owned by file.

Canvas still there.

The “I” of I/O

Reading is simple:

```
TFile* f = new TFile("myfile.root");
TH1F* h = 0;
f->GetObject("h", h);
h->Draw();
delete f;
```

Remember:

TFile owns histograms! File gone, histogram gone!

Ownership and TFile

Separate TFile and histograms:

```
TFile* f = new TFile("myfile.root");
TH1F* h = 0;
TH1::AddDirectory(kFALSE);
f->GetObject("h", h);
h->Draw();
delete f;
```

… and h will stay around.

Random Facts On ROOT I/O

- ROOT files are zipped
- Combine contents of TFiles with **`$ROOTSYS/bin/hadd`**
- Can even open files over the network, e.g.

  `TFile("http://myserver.com/afile.root")`

  including read-what-you-need!

HELP!

What is a TFile?

What functions does it have?

Documentation!

User's Guide, Tutorials, HowTo's:

http://root.cern.ch

Reference Guide (full class documentation):

http://root.cern.ch/root/html