Other random topics

Peter Onyisi

CLEO 101 Day 11
22 June 2005
Topics

- Ntupling made slightly easier
  - SmartMember
  - CWNFramework
- Python and PyROOT

This talk has no exercises; it merely mentions some techniques that you might consider using at some point. It does, however, come with examples.
Some possibilities for ntupling
SmartMember ntuples

- Ntuples made either through HistogramManager or directly through ROOT exist in a different universe from the objects in the data stream (NavTrack etc.)
- SmartMember ntuples are objects that *can* live in the Frame. This means you can read them in a processor with an `extract()` call, write them to pds files, and use them as a record source (your ntuple can also be your skim). You can also write them to HBOOK and ROOT files, of course.
- More information is available about SmartMember ntuples [here](#).
Make the processor and the ntuple files

```
mkproc SmartMemberExampleProc -tuplevector=Pi0Data -pi0
```

```
$C3_CVSSRC/SmartMember/mktuple Pi0Data "energy px py pz pmag shower_energy_min
shower_energy_max"
```

```
$C3_CVSSRC/SmartMember/mkstorevector SmartMemberExampleProc/Pi0Data.h
```

Edit the `SmartMemberExampleProc.cc` file, delete the small example of Pi0Data filling, and enter this in the $\pi^0$ loop:

```cpp
Pi0Data myTuple;
//fill myTuple here
myTuple.energy = pi0.energy();
myTuple.px = pi0.momentum().x();
myTuple.py = pi0.momentum().y();
myTuple.pz = pi0.momentum().z();
myTuple.pmag = pi0.momentum().mag();
myTuple.shower_energy_max = hiEnergy.attributes().energy();
myTuple.shower_energy_min = loEnergy.attributes().energy();

//copy this entry to our list
m_Pi0DataList.push_back(myTuple);
```

Compile. (Ignore all that `SMTypeTraits<>` nonsense.)
You can now run the processor and create the ntuple with the following tcl script:

```tcl
module sel EventStoreModule
eventstore in 20050520
setup_analysis
proc sel SmartMemberExampleProc

module sel RootHistogramModule
root tree out test.root {vector<Pi0Data>}

go 1000
exit
```

Try it! It kind of has to be experienced to get the full effect. (Though the result is also provided in SmartMemberExampleProc.)

The tree will be called `event`, and the names “energy,” “px,” etc. will work as expected.

If you wish to add more variables to your ntuple, just edit the header file for the ntuple (in this case `Pi0Data.h`) and add a new `SMData<xxx>` variable.
Advantages:

- SmartMember ntuple members can be ints instead of floats (unlike HistogramManager ones). They can also have compression hints (like “this variable is always 1 or 0”).
- Because you have full access to suez while making them, there are subtle tricks you can play, like filtering on the ntuple contents with another Processor and producing multiple output files with paths.
- Lets you know if you haven’t filled every variable.

Disadvantages:

- Doesn’t do column-wise ntuples. This may change.
- There’s a significant difference between storing one entry per event and storing more than one.
- A little hard to graft on to an already-existing processor.
What is a column-wise ntuple? Think of it as an ntuple where entries can be *arrays*.\(^1\)

- For example, you can store energies and angles for each shower, and energies for each \(\pi^0\) candidate, all in one ntuple entry which represents one event.
- Nadia’s direct ROOT example was a column-wise ntuple.

How do you create these?

- There is a “semiofficial” way to do this, involving writing a class for each related group of variables and storing them in the Frame à la SmartMember.
- You can do it explicitly with ROOT (or HBOOK). You have to program those interfaces directly. Also you have to explicitly declare memory locations (at least for ROOT).
- Using CWNFramework is functionally equivalent to the explicit method, but slightly more automated and less painful.\(^1\)

\(^1\)Not the original definition, but anyway.
CWNFramework consists of some scripts that autogenerate C++ files for your ntuple, and a couple of classes which know how to store a column-wise ntuple in either ROOT or HBOOK.

You define the ntuple with a file, TUPLE_DEFINITION, that consists of lines like

```
NTUPLE EXNT Example Ntuple

BLOCK GENERAL
INTEGER RUN
DOC "Run number"
INTEGER EVENT
DOC "Event number"

BLOCK DTAG
INDEX NDCAND 0 1000
DOC "Number of D candidates"
INTEGER DMODE NDCAND
DOC "Decay mode of D. Documented on DTagProd webpage."
FLOAT DMBC NDCAND
DOC "Beam-constrained D mass"
FLOAT DDELTAE NDCAND
DOC "D delta E"
```
Advantages:

- Produces column-wise ntuples. (Yes.)
- .h and .cc files never out of sync.
- Can produce HBOOK files for those who want them.
- Can “turn off” blocks of variables at runtime; e.g. you can avoid producing an MC truth block when processing data.

Disadvantages:

- Only integers and floats permitted; none of the other datatypes allowed by ROOT and HBOOK allowed.
- Objects are not in Frame.
- Ntuple does not check for you that it is completely filled, nor does it prevent you from exceeding the size of the arrays.
Python and PyROOT
What is Python?

From the Python website:

*Python is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java.*

**Interpreted:** Python does not require compilation like C++ does. The text of the Python program you write is what gets executed.

**Interactive:** You can issue commands to Python one by one at the command line instead of writing a script. Or you can have Python give you a command line after a script has executed so you can play with the variables.

**Object-oriented:** Objects are like those in C++, except much less painful to code.
How can Python be used at CLEO?

- Unfortunately, you can’t use Python in suez.
- You can, however, access ROOT, and get 99% of the functionality of the C++ interface, with a much nicer programming environment.
  - If you’re the kind who does most analysis with ntuples, this is a great way to go.
Why bother with Python?

- C++ wastes your time so the code can execute faster; Python wastes CPU time so that you can write the code faster.
  - Python manages memory for you. Memory leaks can occur, but they are on the whole not worrisome.
  - Python code is less prone to head-banging C++ stupidities like unwanted copy constructors, pointer problems, bad header files, and such.
  - Because Python does not have header files, it is much nicer to reorganize your code when you need to.
  - Python error messages make a lot more sense than C++ compiler complaints.
- PyROOT’s speed is comparable to ROOT’s “interpreted C++” on ntuples.
- Python is pleasant. For example, “associative arrays” are a basic feature of the language, so you can have `mbc_plots[‘kpi’].Draw()` naturally.
from ROOT import *

h_deltae_vs_mbc = TH2F('h_deltae_vs_mbc',
    '#DeltaE vs. m_{BC}, K#pi',
    100, 1.83, 1.89,
    100, -0.1, 0.1)

f = TFile('example.root')
t = f.Get('exnt')

for i in xrange(t.GetEntries()):
    t.GetEntry(i)
    for j in xrange(t.ndcand):
        if t.dmode[j] == 0:
            h_deltae_vs_mbc.Fill(t.dmbc[j],
                t.ddeltae[j])

h_deltae_vs_mbc.Draw()
You need release 20050417_FULL or later. If you invoke
$C3_OTHER/bin/c3python, you will get Python 2.4 with the
PyROOT interface already set up.

Run a script with

$C3_OTHER/bin/c3python script.py

if you want Python to exit as soon as the program is done, or

$C3_OTHER/bin/c3python -i script.py

for python to give you a command prompt after the program has
run (like running root with a script.C file). (To get out of Python,
type Ctrl-D.)

Run the pyrootexample.py file with the -i option to get the plot
to stay up.
Converting code

You can pretty much copy over what you would do in C++ ROOT to PyROOT:

- You always get new objects in PyROOT with `var = Class()`. All object references act like pointers, except you don’t need to specify what their type is.

- Similarly, to use an object in Python, call `var.Method()` or `var.field`. No `->` necessary.

- You can access the fields of an ntuple as fields of the TTree object in PyROOT.

- Semicolons are not necessary, but **indentation is significant**. Old Fortraners shouldn’t find this a problem.

- Numbers are converted automatically in calls to ROOT functions; pointers are handled automatically. In particular you never have to cast any pointers.
Python Tutorial
Python Library Reference
PyROOT homepage
There are Python examples in the ROOT tutorial directory
Psyco, which makes Python code execute faster “for free,” without giving up interpretation and interaction. (It’s like a just-in-time compiler.)

Note that RooFit is currently somewhat broken with the PyROOT in the version we have here. It will be fixed in the next ROOT upgrade.