Table of Contents

FelixMuellerSandbox

Registration

Max-Planck-Institut für Physik (MPP)

The registration is done locally. You will get a user account to login to the (centrally managed) desktop PCs.

Rechenzentrum Garching (RZG)

The RZG provides a web interface for registration:

https://www.rzg.mpg.de/secure/registrieren/antrag.php?inst=MPP&lang=en

Choose Dr. Stefan Kluth as institute representative and fill in your data, using

position	guest
mentor	Hubert Kroha
systems	MPP Linux Cluster
login name	same as MPP user name
shell	bash

Usually, the registration is processed and approved within 1-2 days.

ATLAS / lxplus

The ATLAS registration is available from http://atlassec.web.cern.ch/atlassec/Registration.htm

Choose the form in the section "external participation". Provide the completed form to Anja Schielke, who will take care of the signature by Siegfried Bethke and sending it to the ATLAS secretary. As the registation process takes a few days, better start the process in advance.

Setup your working environments

Up-to-date instruction how to setup the most essential software on your local machine, the Rechenzentrum Garching (RZG) and the CERN computing cluster (lxplus) can be found here:
https://twiki.cern.ch/twiki/bin/view/Main/MpiQuickStart

As we need RootCore

Getting started with ROOT

The main software utility used in high energy physics analysis is the ROOT framework:
https://root.cern.ch/drupal/ https://root.cern.ch/drupal/content/howtos

A good tutorial about the very basics has been prepared by Mike:
http://fmueller.web.cern.ch/fmueller/ROOT/Flowerdew_ROOT.tgz

ROOT is also available for Python. A PyROOT tutorial can be found here:
http://www.atlas.uni-wuppertal.de/~fleischm/lehre/ROOT2013/tutorial.html

VHbb

The data format we are using is xAOD. It is the standard format for ATLAS Run II analyses. In contrast to the ntuples in the tutorials above, xAOD do not only contain flat branches, but also complex objects and the functions to access it. In order to use them, you need a special setup, which is only available on RZG and lxplus, but not at your local machines.

Working environment for analysis

Login to RZG as described in the MPIQuickStart tutorial. Then, create your working environment.

```
# setup the ATLAS environment
setupATLASUI

# prepare current version of RootCore
mkdir -p rc/2.0.26
cd rc/2.0.26

# setup RootCore
rcSetup Base,2.0.26
```

Now, you should have setup your environment to use xAODs. When you login next time, you don't have to create the RootCore base again:

```
setupATLASUI
cd rc/2.0.26
rcSetup
```

Location of files

The signal samples are located in the ptmp directory. For convenience, place a symbolic link

```
mkdir -p ~/data/CxAOD
cd ~/data/CxAOD
ln -s /ptmp/mpp/fmueller/grid/CxAOD/r229566_substr/user.fmueller.mc14_13TeV.161805.Pythia8_AU2CTE
```

Samples for backgrounds are available as well, but some of the variables are still missing. I will add those a soon as they are needed.

Looking at xAODs in the TBrowser

If you want to examine CxAOD in the TBrowser, setup RootCore and start ROOT:

```
setupATLASUI     # on RZG worker node
cd rc/2.0.26     # your current RootCore work dir
rcSetup    # root is included in RootCore
root -l
```

And then in CINT:

```
gROOT->Macro("$ROOTCOREDIR/scripts/load_packages.C")
xAOD::Init().ignore();
f = TFile::Open("<path-to-your-xAOD-file>")
t = xAOD::MakeTransientTree( f );
TBrowser
```

Several warnings appear when making the transient tree. You can ignore those for the moment. When you get the TBrowser opened, all the variables appear as branches. There are also corresponding folders, but you don't need to worry about them. The naming policy we have adopted is to keep the original name of the branch, but then add e.g. "__Nominal". The "AuxDyn." part in the names is an xAOD thing. The variable name is at the end, e.g. "pt" or "eta".

Analysis code

The simplest approach to analyse CxAOD is using PyROOT. A basic example is given below. To find out, which functions are available for the individual objects inside the CxAOD, there are two approaches:

1) svnweb

to be added

2) python interactively

to be added

Python code

Create a working directory on RZG:

```
mkdir -p ~/analysis/CxAOD
cd ~/analysis/CxAOD
```

Create a file "CxAODExample.py" and add the code given below.

CxAODExample.py Close code

```
#!/usr/bin/env python
import ROOT
import sys
from optparse import OptionParser

def getAux(auxObject, auxName, auxType = 'float'):
    return auxObject.auxdataConst(auxType)(auxName)

def main(opts, args):

    output = ROOT.TFile(opts.output, "RECREATE")
    h_pdgId = ROOT.TH1D("h_pdgId", "ID according to Particle Data Booklet", 60, -30, 30)

    # Set up RootCore and initialize the xAOD infrastructure
    ROOT.gROOT.Macro( '$ROOTCOREDIR/scripts/load_packages.C' )
    if(not ROOT.xAOD.Init().isSuccess()): print "Failed xAOD.Init()"

    # processing each input file individually in order to avoid problems with MetaData
    # for different TTree content
    for filename in args:
        print "Processing %s" % filename
        tree = ROOT.xAOD.MakeTransientTree( ROOT.TFile(filename, "READ"), opts.treename)
        nevents = tree.GetEntries()

        print "TTree contains %i events." % nevents
        if opts.nevents > 0: nevents = min(opts.nevents, nevents)

        for i in xrange(nevents):

            # simple progress status
            if i % 100 == 0 : print "Processing event %i." % i
```

```
            if i >= nevents: break

            # event initialisation
            tree.GetEntry(i)

            # some examples how to access information from CxAOD

            # 1) event information using the function provided by xAOD
            mc_channel_number = tree.EventInfo___Nominal.mcChannelNumber()

            # 2) looping over objects
            for truth in tree.TruthParticle___Nominal:
                 pdgid = truth.PdgId()

                 # fill pdgid to hist
                 h_pdgId.Fill(pdgid)

            # 3) direct access to aux variable (when no accessor available)
            weight = getAux(tree.EventInfo___Nominal, "MCEventWeight", "float")

    output.Write()

if __name__ == "__main__":

    # parse command line input
    parser = OptionParser("usage: %prog [options] file1 [file2 file3 ...]")
    parser.add_option("-o", "--output",    dest="output",              default="output/histograms
    parser.add_option("",   "--treename",  dest="treename",            default="CollectionTree",
    parser.add_option("-n", "--nevents",   dest="nevents", type="int", default=-1,
    opts, args = parser.parse_args()

    main(opts, args)
```

Now, set the file attribute

```
chmod +x CxAODExample.py
./CxAODExample.py --nevents 100 --output pdgid.root ~/data/CxAOD/mc14_13TeV.161805.Pythia8_AU2CTE
```

If you are lucky, you should get an output file with a histogram showing the pdg IDs for 100 events.

WARNING I did not test this!

Making your code ready for batch submission

As you probably want to run over large data sets in the long term plan, we implement the use of a simple, custom made configuration manager. This allows us to use some existing scripts to submit the job on the RZG compuing cluster.

At the top, add the ConfigManager (see below).

```
from ConfigManager import ConfigManager
```

When calling the main function, replace

```
    main(opts, args)
```

by

```
    # run output of option parser through config manager
    cfg = ConfigManager(opts, args)
    cfg.print_config()
```

```
    print cfg.args

    main(cfg.opts, cfg.args)
```

Create a file "ConfigManager.py" and place it in your python directory.

ConfigManager.py Close code

```
#!/usr/bin/env python
from glob import glob

class ConfigManager(object):
    def __init__(self, opts, args):
        self.args = args
        self.opts = opts
        self.data = {}

        if len(args) == 1 and not isroot(args[0]):
            self.data = parse(args[0])

        for key in self.data:
            if opts.ensure_value(key, self.data[key]) != self.data[key]:
                setattr(opts, key, self.data[key])

        # dedicated functionality here
        if "InFiles" in self.data: # special string to replace args with config file input
            self.args = []
            for f in self.data["InFiles"].split(","):
                self.args += glob(f)

        if "OutFile" in self.data: # special string to replace opts.output with config file outpu
            self.opts.output = self.data["OutFile"]

    #def __str__(self):
        #return "channel = %s\tprocess = %s\txsec = %4.2f [pb]" % (self.chan, self.proc, self.xse

    def print_config(self):
        print "args: ", self.args
        print "opts:"
        for key in vars(self.opts): print "%20s:\t%s" % (key, getattr(self.opts, key))

def parse(filename):
    data = {}
    for l in file(filename):
        r = parseline(l)
        if r:
            key, val = r
            if isinstance(val, int): data[key] = int(val)
            if isinstance(val, float): data[key] = float(val)
            if isinstance(val, str): data[key] = val.strip("\"\'")
    return data

def parseline(line):
    s = line.replace(" ", "").replace("\n", "") # cleanup line
    if "#" in s: s = s[:s.find("#")] # remove comments
    if len(s) == 0: return None # check if comment

    assert s.count("=") != 1 or s[-1] != "!", "Syntax error in line \"%s\"" % s # check syntax
    return s.rstrip(";").split("=") # key/value pair

def isroot(filename):
    return file(filename).readline()[0:4] == "root"
```

This should be completely transparent for the standard usage of your program.

C++ code

The analysis using C++ would be analogous to the python Code. A possible example is given with the CxAODReader:

to be added

A more general tutorial for xAOD analysis using RootCore in C++ is given here:

https://twiki.cern.ch/twiki/bin/view/AtlasComputing/SoftwareTutorialxAODAnalysisInROOT

However, the analysis of CxAOD is much much simpler, as all complicated steps such as calibration and systematic variations are already taken in the predecessing steps.

-- FelixMueller - 2015-02-05

This topic: Sandbox > FelixMuellerSandbox
Topic revision: r7 - 2015-03-01 - FelixMueller