How to create nTuples with the "Brunel nTupliser" Documentation

Setup:

Before using the Brunel nTupliser, the following CMSSW dependencies must be downloaded for CMSSW.

Electron BDT weights:

Download the *EgammaAnalysis/ElectronTools* package through the following command (in the CMSSW src directory):

"*>>> git cms-addpkg EgammaAnalysis/ElectronTools*"

Navigate to *EgammaAnalysis/ElectronTools/data* directory and download the xml weight files listed in download.url by using the following command:

"*>>> wget -i download.url*"

The Brunel nTupliser is downloaded from github through the following command:

"*>>> git clone –b <CMSSW_branch> git@github.com:davidcarbonis/brunel_nTuples.git*"

Currently the branches available are "CMSSW_5_3_20" for Run 1 data/MC and "CMSSW_7_4_0" for Run 2 data/MC.

Part 1a - Users using Crab 2:

Within the *test/* subdirectory there is a configuration file called *crab.cfg*. Within the file contains the various parameters required in order to run the nTupiliser code in *test/nTupliserData_cfg.py* or *test/nTupliserMC_cfg.py* (user must choose between MC/data modes). The dataset (listed under the parameter *datasetpath*), the local working directory (*ui_working_dir*), and the output location on the Brunel Tier 2 storage element (*user_remote_dir*) will have to be set by the user.

It goes without saying that if the user wishes to do something different to the norm in the nTupilisation process, they will need to modify *nTupliserData_cfg.py/ nTupliserMC_cfg.py* before running Crab 2.

Once the crab configuration file is set up, the usual formulas before running CMS Software will need to be cast – ie. executing "*>>> cmsenv*". The user will need to have a valid CMS VO certificate (this can be checked by excuting "*>>> voms-proxy-init -voms cms*").

To submit the job to the grid, run Crab 2. A short overview is given below, but if you encounter any difficulty, the relevant documentation for using Crab 2 can be found on the CMS Twiki.

i. Execute "*>>> crab -create*" to create the job, check the availability of the dataset selected and to prepare all the jobs for submission according to the selected job splitting specified by the configuration file.
ii. Execute "*>>> crab -submit*" to submit the jobs.
iii. Execute "*>>> crab -status -c <dir-name>*" to check the progress of the submitted jobs.
iv. Once all the jobs have completed execute "*>>> crab –getoutput -c <dir-name>*" to retrieve the log files of the jobs (just the log files as the output files are copied over to the storage element associated to the T2 specified).

Continue to Part 2.

Part 1b - Users using Crab 3:

Within the *test/* subdirectory there are configuration files called Crab3Config_MC.py and Crab3Config_data.py which contain the necessary scripting to run the nTupilser code in *test/ nTupliserMC_cfg _cfg.py* or *test/ nTupliserData.py* respectively. The dataset (listed under the parameter *datasetpath*), the local working directory (*ui_working_dir*), and the output location on the Brunel Tier 2 storage element (*user_remote_dir*) will have to be set by the user.

It goes without saying that if the user wishes to do something different to the norm in the nTupilisation process, they will need to modify *nTupliserData_cfg.py/ nTupliserMC_cfg.py* before running Crab 3.

Once the crab configuration file is set up, the usual formulas before running CMS Software will need to be cast – ie. executing "*>>> cmsenv*". The user will need to have a valid CMS VO certificate (this can be checked by excuting "*>>> voms-proxy-init -voms cms*").

To submit the job to the grid, run Crab 3. A short overview is given below, but if you encounter any difficulty, the relevant documentation for using Crab 3 can be found on the CMS Twiki.

i. Execute "*>>> crab -submit -c <CRAB-config-file>*" to create and submit the jobs.

ii.   Execute "*>>> crab -status <dir-name>*" to check the progress of the submitted jobs. One can get a more detailed status report by added "—long" to the crab status command.

iii. Once all the jobs have completed execute "*>>> crab –getoutput <dir-name>*" to retrieve the log files of the jobs (just the log files as the output files are copied over to the storage element associated to the T2 specified).

Continue to Part 2.

Part 2:

Once the output logs have been retrieved one needs to create a list of all the files produced from the Crab 2 jobs using the python script found in "/*skimMacros/*". This is an important step as sometimes when jobs are re-submitted they produce extra files in the remote directory, so if one just runs over the entire contents of the directory, it can contain extra files and as such will double count. This script makes sure that one is only considering the correctly processed files. To run it the following needs to be entered into the console:

"*>>> python listCorrectFilesInSRM.py <dir-Crab2-project> <fileListFileOutputName>*"

Following this the following command must be run in order to put the file names into a format which root likes to handle (main cause I believe is that the original script was written for pion2 and wasn't updated for pion – this corrects that):

```
>>> sed –i
's|srm://dc2-grid-64.brunel.ac.uk:8446/srm/managerv2?SFN=/dpm/|root://dc2-
grid-64.brunel.ac.uk//dpm/|g'
< fileListFileOutputName >
```

These file names can now be plugged into any root application on Pion and it should be able to access them just fine. The instructions on using the skimming program that copies the datasets from the T2 to the nfs is included below – as everything runs faster locally! After compiling the exe, source the setup file and run cmsenv (as it is implicit, I'll state it explicitly, this program needs to be located in a CMSSW setup) - in that order! Afterwards, execute "*skims.exe*":

```
>>> source setup.sh
>>> cmsenv
>>> ./skims.exe < fileListFileOutputName> <outputDataSetName>
```

N.B.

All "*>>>*" are referring to the start of a new line on the console – in case this wasn't implicit enough.

Crab 2 and the skimming program have been run and tested only under CMSSW_5_3_20 by the author.

The author takes no responsibility for any unintended consequences stemming from use of these instructions. They are meant as a guide only, to provide a starting point and to give others documentation which he desperately lacked during his student days.

Code observations:

The selection of genParticles occurs between lines 1275-1298. The particle is only run over if the particle has a candidate status of "3" (1278) and is a quark or lepton or W± or Z^0 (line 1279).

The candidate status flag is nominally set to "3", however, it can also take other values:

- 0: Null Entry
- 1: Particle which hasn't decayed or fragmented. Final state as given by the generator.
- 2: Decayed or fragmented entry. For example, in the case of tbZ, where the b is produced from gluon splitting, the spare bbar would have this status.
- 3: The "hard part" of the interaction and mmediate decays of resonances. Ie. partons used in matrix event calculation including immediate resonance decays.
- 4+: Not relevant.

The hunt for t/tbar amongst the genParticles occurs between lines 1308-1419. Line 1313 ensure that the t/tbar has at least two daughters, and if this is the case, all the daughters are looped over.

Line 1320 looks for a daughter which is a b and has status "3", if this is the case, the transverse energy and x-y-z momentum are obtained (lines 1329-1332). Lines 1337-1354 looks for a W± which has status "3", and if this is the case, the transverse energy and x-y-z momentum of the hadronic daughter(s) are obtained (lines 1356-1369) and likewise for the leptonic daughters (lines 1371-1385), if there are any.

If a b and a W± with hadronic daughters has been found, lines 1387-1400 keep the top candidate in hadronic decay's 4-vector. Similarly, if a b and a W± with leptonic daughters has been found, lines 1402-1415) keep the top candidate in leptonic decay's 4-vector.