A. The *CutLang* user manual

V2.1 / May 2018 - For users with some experience in ROOT)

A.1. How to obtain and build the CutLang *interpreter:*

The *CutLang* interpreter package can be run on any standard Unix, Linux, or OSX machine which has an installation of the ROOT package. Basic knowledge of terminal operations such as text file editing and moving files around is also needed in addition to knowledge of how to use the *CutLang* interpreter. Some knowledge on ROOT macros is helpful for manipulating histograms, but is not essential for basic analysis operations.

The latest version of the *CutLang* interpreter package can be downloaded from HepForge at

```
http://cutlang.hepforge.org
```

The downloaded file should be opened with

```
tar -xzf cutlang-VXX.YY.ZZ.tgz
```

which will automatically create the `CutLang` directory. The interpreter is built using the commands

```
cd CutLang/CLA
make
```

The interpreter should be executed from the `CutLang/runs` subdirectory.

A.2. How to run a CutLang *analysis*

Typically, running an analysis with *CutLang* requires the following minimal set of files

- The text based analysis description file (examples can be found under directory `runs`, which can be used as examples to create the user's analysis description files. Default file name is `CLA.ini`, however the file name can be specified at the command line.)
- The shell script file `CLA.sh` : The script that executes the *CutLang* analysis (see Step 4 below). This script takes two mandatory arguments: i) the name of the ROOT file to be analyzed, e.g., `cms-opendata-ttbar.root`, which is to be downloaded from the relevant source or generated by the user externally; and ii) the input data format. The already existing input data formats are LVL0 (*CutLang* default format), ATLASOD (ATLAS open data), CMSOD (CMS open data), Delphes, FCC, LHCO. New input data formats can be added as to be described in Appendix A.5. The other arguments, specified below, are optional:

  `-e|--events` the number of events to be processed. By default all events (represented by 0) are processed.

  `-i|--inifile` the analysis description file to be processed. By default `CLA.ini` is processed.

  `-v|--verbose` the verbosity event count. By default after every 1000 events, the current event count is written on the screen. This option allows to change the default number of events reported, e.g. as `--verbose 5000`.

  `-h|--help` displays these explanations as reminders.

- `histoOut-NAME.root` output file produced in Step 8 where NAME is the name of the analysis description file. If no name is specified, the default value of `histoOut-CLA.root` is used. In case of multiple signal regions, each region will have its own directory inside the output file marked with `BP_i` where `i` is an index number.

- The ROOT file containing events to be analyzed (A small sample of $t\bar{t}$ events can be downloaded from http://cutlang.hepforge.org for a quick start). Multiple comma separated input files and their paths can be specified at the command line as; e.g.,

```
./CLA.sh ../roots/atlas1ttbar.root,../roots/atlas2ttbar.root ATLASOD
```

The steps below should be followed for running an analysis in *CutLang*.

0. Open a terminal
1. Go to directory `CutLang`/runs
2. Edit the analysis description file, e.g. `CLA.ini`, as needed
3. Go back to the same terminal
4. Execute the analysis description edited in Step 2 using the following command:

   ```
   ./CLA.sh [input ROOT file(s)] [data format] [-i youranalysis.ini]
            [-e number_of_events_to_process] [-v verbosity]
   ```

   An example would be

   ```
   ./CLA.sh cms-opendata-ttbar.root CMSOD -i CLA.ini -e 10000
   ```

5. *CutLang* outputs the analysis evaluation results to the terminal.
   - If the analysis description file is syntactically correct, the following message should appear on the screen: " `End of analysis initialization`" , in which case all is well and proceed to Step 6.
   - If there are any errors in the analysis description, *CutLang* notifies the user of the unknown parameter(s) as " `UFO(s)`" . Go back to Step 2, verify and correct the "`ini`" file.
6. `CLA` lists messages every 1000 processed events until it reaches the end of the ROOT file.
7. An efficiency table for the analysis is displayed on the screen.
8. `CLA` displays the message: " `saving...finished`" at the end of the analysis. The output ROOT file is saved under the same directory. The file name will be `histoOut-NAME.root`, where NAME is the name of the analysis description (`ini`) file. If no name is specified, the default value of `histoOut-CLA.root` is used. If users wish to keep output from a previous run, the output file should be renamed. Otherwise, the `CLA.sh` will overwrite the output file. In case of multiple signal regions, each region will have its own directory inside the output file marked with `BP_i` where `i` is an index number.

A.3. How to prepare a CutLang *analysis description file*

The analysis description file (e.g. `CLA.ini`) contains three sections:

- **Object thresholds:** This mandatory section contains the η and p_T threshold values for a particle to be accepted.

- **User definitions:** This is a non-mandatory section containing user definitions starting with keyword "`def`" for new composite particles and variables. These definitions can be used to create shorthand names for otherwise long expressions for later use. This section can also contain the derived objets sets, such as cleaned or tighter objects which might be used in the event selection. The object definitions should start with the with keyword "`obj`".
- **Event selection:** This section is mandatory and consists of lines starting with keyword " `cmd`", which define event operations or selection criteria. The section also consists of lines starting with " `histo`", which signify the histogram definitions.

Common rules for all sections

- All lines start with one of the `def`, `cmd`, or `histo` keywords. No space should exist before the keywords. Note that there are no keywords for the object thresholds section.
- An indefinite amount of space is allowed between the keyword and the command/description.
- Every command/description must be enclosed within double quotation marks, and there should be a space before the ending quotation mark, e.g. `"mLL : { LEP_1 LEP_0 }m "`, or `"mLL [] 70 120 "`.
- There is no upper limit for the number of lines.
- At least one space must be left before and after each term; including operands, numbers.
- All units in *CutLang* are either GeV or radians ($c = 1$, therefore mass, momentum, energy are all in GeV)
- All variable, function, and particle names are case sensitive.

Additional rules in editing the analysis description file

- "~=" and "! =" cannot be combined with any other operator or function. For example:

  " `nJET` >= 6 AND `nBJET` >= 2 " ***OK***

  " `mTopb` ~=175 " ***OK***

  " `mTopb` ~= 175 AND `nBJET` >= 2 " ***not OK***

  " `mTopb` ~= 175 AND `mTopb2` ~=175 " ***not OK***
- Any expression after a "#" is considered a comment and ignored. Consequently, lines can be skipped by commenting them out.
- Names of the user defined composite particles and variables must be unique. They cannot be redefined within the same analysis.
- "{ }" are used for stating properties of particles, e.g. mass, charge, etc. One can add as many particles as required in the term within the curly braces.
- The order of particles in a particle combination is immaterial; i.e. : `LEP_0 LEP_1` is functionally identical to `LEP_1 LEP_0`.

- All definitions in the def section should be ordered with the lower case and upper case reverse alphabetical order respectively.

A note on the trigger scope

One should note that only electron and muon triggers are implemented in this version. (The tau channel is not available in this version.) Wherever the term “lepton” or the abbreviation “LEP” is used, this refers to either an electron or a muon depending on the selected trigger. A trigger value of 0 deselects that channel, 1 treats the input file as data (i.e. no event weights are applied), and finally 2 ensures the application of all the relevant weights such as Monte Carlo weights, pileup weights, vertex weights and b-tagging weights. For a given analysis run, only a single lepton type can be triggered, while the other type has to be set to 0.

A.3.1. An example analysis description file with multiple regions

```
###### OBJECT THRESHOLDS
minpte = 15.0 # min pt of electrons
minptm = 15.0 # min pt of muons
minptj = 15.0 # min pt of jets
maxetae = 2.47 # max pseudorapidity of electrons
maxetam = 2.5 # max pseudorapidity of muons
maxetaj = 5.5 # max pseudorapidity of jets
TRGm = 0 # muon Trigger Type: 0=dont trigger, 1=1st trigger (data) 2=2nd trigger (MC)
TRGe = 2 # electron Trigger Type: 0=dont trigger, 1=1st trigger (data) 2=2nd trigger (MC)

###### USER DEFINITIONS
def "WH1 : JET_-1 JET_-1 " # W boson of the first top
def "WH2 : JET_-11 JET_-11 " # W boson of the second top
def "mWH1 : { WH1 }m " # mass of W boson of the first top
def "mWH2 : { WH2 }m " # mass of @ boson of the second top
def "mTopH1 : { WH1 JET_-2 }m " # first top quark's mass
def "mTopH2 : { WH2 JET_-4 }m " # second top quark's mass

###### EVENT SELECTION
algo __preselection__
cmd "ALL " # to count all events
cmd "nJET >= 6 " # events with 6 or more jets
cmd "MET < 100 " # fully hadronic events should have small MET
#cmd "FillHistos "
#histo "Basics "

algo __teknik1__
__preselection__
cmd "mTopH1 - mTopH2 / 4.2 ^ 2 + mWH1 - 80.4 / 2.1 ^ 2 + mWH2 - 80.4 / 2.1 ^ 2 ~= 0 "
cmd "FillHistos "
histo "mWHh1 , Hadronic W reco (GeV), 50, 50, 150, mWH1 "
histo "mWHh2 , Hadronic W reco (GeV), 50, 50, 150, mWH2 "
histo "mTopHha1 , Hadronic top reco (GeV), 70, 0, 700, mTopH1 "
histo "mTopHhb1 , Hadronic top reco (GeV), 70, 0, 700, mTopH2 "

algo __teknik2__
__preselection__
cmd "mWH1 - 80.4 / 2.1 ^ 2 + mWH2 - 80.4 / 2.1 ^ 2 ~= 0 " # 2 WHads
cmd "mTopH1 - mTopH2 / 4.2 ^ 2 ~= 0 "
cmd "FillHistos "
histo "mWHh1 , Hadronic W reco (GeV), 50, 50, 150, mWH1 "
histo "mWHh2 , Hadronic W reco (GeV), 50, 50, 150, mWH2 "
histo "mTopHha1 , Hadronic top reco (GeV), 70, 0, 700, mTopH1 "
histo "mTopHhb1 , Hadronic top reco (GeV), 70, 0, 700, mTopH2 "
```

A.4. *How to view a* CutLang *analysis output*

There are two ways to view the contents of the *CutLang* output ROOT file:

1. Open it using ROOT : `root.exe histoOut-CLA.root;` and launch a `TBrowser`
2. Run the default macro: `./showall.sh` :

   ```
   ./showall.sh [regionID] [histofileName]
   ```

   This shows the results for the region regionID. Default values are 1 and histoOut-CLA.root.

The following few lines show the typical beginning of an output ROOT file. Note that the user definitions, and the cutflow all in *CutLang* format are reproduced for the reader's convenience. Moreover, the object definition threshold values are stored in `TParameter` variables. The efficiency histograms which are always automatically booked and filled are also shown.

```
root [2] .ls
TDirectoryFile*  BP_2 BP_2
 KEY: TText CLA2defs;1
WH1 : JET_-1 JET_-1
WH2 : JET_-11 JET_-11
mWH1 : { WH1 }m
mWH2 : { WH2 }m
mTopH1 : { WH1 JET_-2 }m
mTopH2 : { WH2 JET_-4 }m
WHbR1 : {WH1 , JET_-2 }dR
WHbR2 : {WH2 , JET_-4 }dR
Wchi2 : { WH1 }m - 80.4 / 2.1 ^ 2 + { WH2 }m - 80.4 / 2.1 ^ 2
topchi2 : mTopH1 - mTopH2 / 4.2 ^ 2

 KEY: TText CLA2cuts;1
cmd1 : ALL
cmd2 : nJET >= 6
cmd3 : MET < 100
cmd4 : topchi2 + Wchi2 ~= 0
cmd5 : FillHistos
cmd6 : WHbR1 > 0.6
```

```
cmd7 : WHbR2 > 0.6
cmd8 : FillHistos

 KEY: TParameter<double> minpte;1
 KEY: TParameter<double> maxetae;1
 KEY: TParameter<double> minptm;1
 KEY: TParameter<double> maxetam;1
 KEY: TParameter<double> minptj;1
 KEY: TParameter<double> maxetaj;1
 KEY: TParameter<double> TRGe;1
 KEY: TParameter<double> TRGm;1
 KEY: TH1F eff;1 selection efficiencies
```

A.5. How to interface a new input data format to the CutLang *interpreter*

This section describes how to build the interface between a new data file format represented as a flat ntuple and the standard types used by the *CutLang* interpreter. This is one aspect of the current version of *CutLang* that requires some coding expertise. *CutLang* uses ROOT's `MakeClass` for this purpose.

- Obtain a sample ROOT ntuple file containing the new data format and load into ROOT (e.g., using `TFile f("myfile.root")`)
- Call the ROOT `MakeClass` command on the relevant tree, specifying a class name

  ```
  tree->MakeClass("NewFormatName");
  ```

- Move the resulting header file (`NewFormatName.h`) into the `analysis_core` subdirectory, and is include it in the main code `CLA.C`
- Move the resulting implementation macro (`NewFormatName.C`) into the `CutLang/CLA` directory, and include the following required headers in it:

  ```
  #include "lhco.h"
  #include <TH2.h>}
  #include <TStyle.h>}
  #include <TCanvas.h>}
  #include <signal.h>}
  #include "dbx_electron.h"
  #include "dbx_muon.h"
  #include "dbx_jet.h"
  #include "dbx_a.h"
  #include "DBXNtuple.h"
  #include "analysis_core.h"
  #include "AnalysisController.h"
  ```

- In the event loop, the input data must be transferred to the standard *CutLamg* types, e.g., the electron, muon, photon and jet particle vectors, without forgetting any available event-wide information like RunNumber, EventNumber etc. An example conversion for the `LHCO` format is:

```
      TLorentzVector alv; dbxMuon *adbxm; vector<dbxMuon> muons;
      for (unsigned int i=0; i<Muon_; i++) {
        alv.SetPtEtaPhiM(Muon_PT[i], Muon_Eta[i], Muon_Phi[i], (105.658/1E3)); // all in GeV
        adbxm= new dbxMuon(alv);
        adbxm->setCharge(Muon_Charge[i] );
        adbxm->setEtCone(Muon_ETiso[i] );
        adbxm->setPtCone(Muon_PTiso[i] );
        adbxm->setParticleIndx(i);
        muons.push_back(*adbxm);
        delete adbxm;
        }
      \end{/}
      \item Modify the end of the macro to be as follows:
      \begin{lstlisting}
      AnalysisObjects a0={muons, electrons, photons, jets, met, anevt};
      aCtrl.RunTasks(a0);
      } // end of event loop
      aCtrl.Finalize();
      } // end of Loop function
```

A.6. How to add user functions

As discussed in Section 3.7, it is possible to add (or delete) user functions to *CutLang* for computation of complex variables. This can be done using the `scripts/adduserfunction.py` script. To add a function, run

```
python adduserfunction.py <functionname>
```

This creates the function header `analysis_core/dbx_<functionname>.h` and adds the function into `analysis_core/dbxCut.cpp`. The content of an example user function `dbx_userfunc1.h` is shown below:

```
#ifndef DBX_USERFUNC1_H
#define DBX_USERFUNC1_H

class dbxCutuserfunc1 : public dbxCut {
 public:
     dbxCutuserfunc1: dbxCut("}userfunc1"){}
     dbxCutuserfunc1(std::vector<int> ts, std::vector<int> is, int v ):dbxCut("}s(name)s",ts,

      bool select(AnalysisObjects *ao){
          float result;
          result=calc(ao);
          return (Ccompare(result) );
      }
      float calc(AnalysisObjects *ao){
        float retval;

/* this is an example on how to calculate Meff in an external function.
// each particle given to this function is of type getParticleType(jj)
// each particle given to this function has index getParticleIndex(jj)

float meff=0;
for (unsigned int jj=0; jj<2; jj++) //Meff = MET + sum of all a particle type's all PTs
  switch (getParticleType(jj*2)){
    case 0: for (int ii=0; ii<ao->muos.size(); ii++)
                meff+=ao->muos[ii].lv().Pt();
                break;
    case 1: for (int ii=0; ii<ao->eles.size(); ii++)
                meff+=ao->eles[ii].lv().Pt();
                break;
    case 2: for (int ii=0; ii<ao->jets.size(); ii++)
                meff+=ao->jets[ii].lv().Pt(); //these are un-tagged jets
                break;
    case 3: for (int ii=0; ii<tagJets(ao,1).size(); ii++)
                meff+=tagJets(ao,1)[ii].lv().Pt(); //these are b-tagged jets
                break;
    case 4: for (int ii=0; ii<tagJets(ao,0).size(); ii++)
                meff+=tagJets(ao,0)[ii].lv().Pt(); //these are b-tag rejected jets
                break;
    case 7: meff+=ao->met.Mod();
                break;
    case 8: for (int ii=0; ii<ao->gams.size(); ii++)
                meff+=ao->gams[ii].lv().Pt();
                break;
}
retval=meff;
*/


        // ***********************************
        // Write your own code here
        // ***********************************

        return retval;
}
private:
};
```

If for some reason, the user function needs to be deleted, this can also be done safely with the same script using

```
python adduserfunction.py --delete <functionname>
```

B. Two example analyses In *CutLang*

In the following, we present implementations of two real life analyses written using *CutLang*. The analyses are taken from a recent study comparing public recasting tools done within the context of Les Houches PhysTeV 2017 proceedings [20] (see Section 21). The first example is an ATLAS exotic monophoton search [31] with detailed object definitions and multiple event selection regions defined by different missing transverse momentum thresholds. The second example is an ATLAS SUSY search in the jets and missing transverse momentum final state [32], which also has a detailed object selection and multiple event selection regions defined by several complex selection variables. These analyses were run using *CutLang* on signal events generated for the Les Houches study (as described in [20]), and simulated privately using Delphes. The results obtained are very close to those presented in [20], and exactly the same with those obtained from a private comparison with a recent LHADA interpreter called *lhata2tnm* (described in Section 27 of [20]). For the latter comparison, the same Delphes samples were used by *CutLang* and *lhada2tnm*.

Example 1: ATLAS exotic monophoton analysis

```
# info analysis
# experiment ATLAS
# id EXOT−2016−32
# publication Eur.Phys.J. C77 (2017) no.6, 393
# sqrtS 13.0
# lumi 36.1
# arXiv 1704.03848
# hepdata https://www.hepdata.net/record/ins1591328
# doi 10.1140/epjc/s10052−017−4965−8

######## GENERIC OBJECT THRESHOLDS
minptp = 10.0 # min pt of photons
minpte = 7.0 # min pt of electrons
minptm = 6.0 # min pt of muons
minptj = 20.0 # min pt of jets

maxetap = 2.37 # max pseudorapidity of photons
maxetae = 2.47 # max pseudorapidity of electrons
maxetam = 2.70 # max pseudorapidity of muons
maxetaj = 4.50 # max pseudorapidity of jets

######## OBJECT SELECTION
obj ”JETclean : JET ”
cmd ”{ JET_ , ELE_ }dR >= 0.2 ”

obj ”ELEclean : ELE ”
cmd ”{ ELE_ , JETclean_ }dR >= 0.4 ”

obj ”MUOclean : MUO ”
cmd ”{ MUO_ , JETclean_ }dR >= 0.4 ”
```

```
obj ”PHOtight : PHO ”
cmd ”{ PHO_ }AbsEta ][ 1.37 1.52 ”

obj ”JETsr : JETclean ”
cmd ”{ JETclean_ }Pt > 30 ”
cmd ”{ JETclean_ , PHOtight_ }dR >= 0.4 ”
cmd ”{ JETclean_ , METLV_0 }dPhi >= 0.4 ”

######## EVENT SELECTION
algo __preselection__
cmd ”ALL ” # to count all events
cmd ” nPHOtight >= 0 ” # events with 1 or more tight photons
cmd ”{ PHOtight_0 }Pt > 150 ” # select photons[0].PT > 150
cmd ”{ PHOtight_0 , METLV_0 }dPhi > 0.4 ” # select isolated photons
cmd ” MET HT ^ 0.5 / > 8.5 ” # select METoverSqrtSumET > 8.5
cmd ” nJETsr <= 1 ”
cmd ”nJETsr == 0 ? ALL : { JETsr_0 , METLV_0 }dPhi > 0.4 ” # select dphi(jetsSR.Phi, MET.Phi) > 0.4
cmd ” nMUOclean == 0 ” # select cleanmuons.size == 0
cmd ” nELEclean == 0 ” # select cleanelectrons.size == 0

# Inclusive search regions
algo __SRI1__
__preselection__
cmd ”MET > 150 ”
algo __SRI2__
__preselection__
cmd ”MET > 225 ”
algo __SRI3__
__preselection__
cmd ”MET > 300 ”

# Exclusive search regions
algo __SRE1__
__preselection__
cmd ”MET [] 150 225 ”
algo __SRE2__
__preselection__
cmd ”MET [] 225 300 ”
```

Example 2: ATLAS SUSY JetMET analysis

```
# info analysis
# experiment ATLAS
# id SUSY−2013−15
# publication Eur. Phys. J. C(2016) 76: 392
# sqrtS 13.0
# lumi 3.2
# arXiv 1605.03814
# hepdata http://hepdata.cedar.ac.uk/view/ins1304456
# doi 10.1140/epjc/s10052−016−4184−8

######## GENERIC OBJECT THRESHOLDS
minptp = 10.0 # min pt of photons
minpte = 10.0 # min pt of electrons
minptm = 10.0 # min pt of muons
minptj = 20.0 # min pt of jets

maxetap = 2.37 # max pseudorapidity of photons
maxetae = 2.47 # max pseudorapidity of electrons
maxetam = 2.70 # max pseudorapidity of muons
maxetaj = 2.80 # max pseudorapidity of jets

######## USER DEFINITIONS
def "Meff : MET + HT " #Meff is simple
def "JM0 : { JETsr_0 , METLV_0 }dPhi "
def "JM1 : { JETsr_1 , METLV_0 }dPhi "
def "JM2 : { JETsr_2 , METLV_0 }dPhi "
def "Meff4j : MET + { JETsr_0 }Pt + { JETsr_1 }Pt + { JETsr_2 }Pt + { JETsr_3 }Pt "
def "Meff5j : Meff4j + { JETsr_4 }Pt "
def "Meff6j : Meff5j + { JETsr_5 }Pt "

######## OBJECT SELECTION
obj "JETclean : JET "
cmd "{ JET_ , ELE_ }dR >= 0.2 "

obj "MUOclean : MUO "
cmd "{ MUO_ , JETclean_ }dR >= 0.4 "
cmd "{ MUO_ }IsolationRhoCorr < 0.1"

obj "ELEclean : ELE "
cmd "{ ELE_ , JETclean_ }dR >= 0.4 "

obj "ELEveryclean : ELE "
cmd "{ ELE_ , JETclean_ }dR >= 0.4 "

obj "JETsr : JETclean "
cmd "{ JETclean_ }Pt > 50 "

######## EVENT SELECTION
algo __preselection__
cmd "ALL " # to count all events
cmd "MET > 200 "
#cmd "nPHOtight >= 0 "
cmd "nMUOclean == 0 " # Reject evt if there is a muon with pT > 10
#cmd "nMUOclean == 0 ? ALL : { MUOclean_0 }Pt < 10 " # Reject evt if there is a muon with pT > 10
cmd "nELEveryclean == 0 " # Reject evt if there is a muon with pT > 10
#cmd "nELEveryclean == 0 ? ALL : { ELEveryclean_0 }Pt < 10 " # Reject evt if there is an electron with pT > 10
cmd "nJETsr > 0 "

algo __2jt__
```

```
__preselection__
cmd ”nJETsr >= 1 ”
cmd ”{ JETsr_0 }Pt > 200 ”
cmd ”nJETsr >= 2 ”
cmd ”Ex1 ( JETsr_ ) > 0.8 ”
#cmd ”nJETsr == 2 ? JM0 − JM1 < 0 ? JM0 > 0.8 : JM1 > 0.8 : ALL ”
#cmd ”JM0 − JM1 < 0 ? JM0 − JM2 < 0 ? JM0 > 0.8 : JM2 > 0.8 : JM1 − JM2 < 0 ? JM1 > 0.8 : JM2 > 0.8 ”
cmd ”{ JETsr_1 }Pt > 200 ”
cmd ”MET / HT ˆ 0.5 ( JETsr_ ) > 20 ”
cmd ”Meff ( JETsr_ ) > 2000 ”

algo __2jm__
__preselection__
cmd ” nJETsr >= 1 ”
cmd ”{ JETsr_0 }Pt > 300 ”
cmd ” nJETsr >= 2 ”
cmd ” Ex1 ( JETsr_ ) > 0.4 ”
cmd ”{ JETsr_1 }Pt > 50 ”
cmd ” MET / HT ˆ 0.5 ( JETsr_ ) > 15 ”
cmd ” Meff ( JETsr_ ) > 1600 ”

algo __2jl__
__preselection__
cmd ”nJETsr >= 1 ”
cmd ”{ JETsr_0 }Pt > 200 ”
cmd ”nJETsr >= 2 ”
cmd ”Ex1 ( JETsr_ ) > 0.8 ”
cmd ”{ JETsr_1 }Pt > 200 ”
cmd ”MET / HT ˆ 0.5 ( JETsr_ ) > 15 ” # make sure we use JETsr in HT
cmd ”Meff ( JETsr_ ) > 1200 ”

algo __4jt__
__preselection__
cmd ” nJETsr >= 4 ”
cmd ”{ JETsr_0 }Pt > 200 ”
cmd ”{ JETsr_1 }Pt > 100 ”
cmd ”{ JETsr_2 }Pt > 100 ”
cmd ”{ JETsr_3 }Pt > 100 ”
cmd ”Ex1 ( JETsr_ ) > 0.4 ”
cmd ”{ JETsr_−1 , METLV_0 }dPhi ~= 0.0 ”
cmd ”{ JETsr_−1 , METLV_0 }dPhi > 0.2 ”
#cmd ”aplanarity ( JETsr_ ) > 0.04 ”
cmd ”MET / Meff4j ( JETsr_ ) > 0.2 ”
cmd ”Meff (JETsr_ ) > 2200 ”

algo __5j__
__preselection__
cmd ” nJETsr >= 5 ”
cmd ”{ JETsr_0 }Pt > 200 ”
cmd ”{ JETsr_1 }Pt > 100 ”
cmd ”{ JETsr_2 }Pt > 100 ”
cmd "{ JETsr_3 }Pt > 100 "
cmd "{ JETsr_4 }Pt > 50 "
cmd "Ex1 ( JETsr_ ) > 0.4 "
cmd "{ JETsr_-1 , METLV_0 }dPhi ~= 0.0 "
cmd "{ JETsr_-1 , METLV_0 }dPhi > 0.2 "
#cmd "aplanarity ( JETsr_ ) > 0.04 "
cmd "MET / Meff5j ( JETsr_ ) > 0.25 "
cmd "Meff ( JETsr_ ) > 1600 "

algo __6jm__
__preselection__
cmd " nJETsr >= 6 "
cmd "{ JETsr_0 }Pt > 200 "
cmd "{ JETsr_1 }Pt > 100 "
cmd "{ JETsr_2 }Pt > 100 "
cmd "{ JETsr_3 }Pt > 100 "
cmd "{ JETsr_4 }Pt > 50 "
cmd "{ JETsr_5 }Pt > 50 "
cmd "Ex1 ( JETsr_ ) > 0.4 "
cmd "{ JETsr_-1 , METLV_0 }dPhi ~= 0.0 "
cmd "{ JETsr_-1 , METLV_0 }dPhi > 0.2 "
#cmd "aplanarity ( JETsr_ ) > 0.04 "
cmd "MET / Meff6j ( JETsr_ ) > 0.25 "
cmd "Meff ( JETsr_ ) > 1600 "

algo __6jt__
__preselection__
cmd " nJETsr >= 6 "
cmd "{ JETsr_0 }Pt > 200 "
cmd "{ JETsr_1 }Pt > 100 "
cmd "{ JETsr_2 }Pt > 100 "
cmd "{ JETsr_3 }Pt > 100 "
cmd "{ JETsr_4 }Pt > 50 "
cmd "{ JETsr_5 }Pt > 50 "
cmd "Ex1 ( JETsr_ ) > 0.4 "
cmd "{ JETsr_-1 , METLV_0 }dPhi ~= 0.0 "
cmd "{ JETsr_-1 , METLV_0 }dPhi > 0.2 "
#cmd "aplanarity ( JETsr_ ) > 0.04 "
cmd "MET / Meff6j ( JETsr_ ) > 0.2 "
cmd "Meff ( JETsr_ ) > 2000 "
```

Note that in the second example, the region defined by the algorithm called `2jt` contains an external user function called `Ex1`. This external function finds the smallest polar angular distance between MET and the first three jets. When there are more jets, only the first three are taken into account, while when there are two jets, only those available two jets are considered. The same function can also be implemented using the ternary functions and comparison operators all available in *CutLang*. The two commented out lines just below the `Ex1` function show how to do that same computation using the ternary functions and comparison operators.