Table of Contents

n.n Configurable Analysis

Complete:
Detailed Review status

Newsbox
ConfigurableAnalysis works now in 1.6.12 and 2.1.9. This guide covers the overall structure and usage of this tool . The concept and structure will not change between the versions, however some functionalities differ.

Disclaimer

Everything on this twiki might become obsolete at some point of the evolution of the ConfigurableAnalysis. Please do report to the authors missing instructions or broken functionalities.

Goals of this page:

This page is intended to familiarize you with writing a Configurable Analysis. In particular, you will learn:

- Structure of a generic analysis
- Structure of ConfigurableAnalysis
- Writing your own ConfigurableAnalysis
- Interface to user defined Ntupler

Contents

Introduction

The ConfigurableAnalysis allows you to perform the simple task of selecting events with a plotting device interleaved to monitor control distribution during selection. The usage is quite simple and the user should not have to write any code. The tool has been design such that most of the analysis selection can be done by configuration file.

Impatient User

Who should still read through the whole twiki before reporting on a problem they encounter.

- Setup your environment
- Pick up a full example configuration file
- Run.

Structure of an Analysis

This workbook page is intended to get you up and running with ConfigurableAnalysis. Its purpose is to serve as a primer to doing a physics analysis and what goes into creating an analysis. Users already familiar with the basic aspects physics analysis can skip ahead to [[#OwnAnalysis] [Writing your own Analysis.]]

In any physics analysis there are a series of steps which occur in the course of making a measurement. ConfigurableAnalysis serves as a platform to automate these steps for you without sacrificing flexibility. It puts the responsibility of running the analysis in the hands of the user, and performs all of the tedious tasks automatically.

Briefly an analysis can consists of:

- Event selection
- Calculation of some interesting quantity
- A fit of data to an theoretical model
- Visualization of data

Event Selection

Event selection consists of the user specifying some requirement of the data. For instance, "I want only events who's transferse momentum is greater than 5 GeV/c and who's Eta between pi/4 and pi/2" These are what are known as cuts. They are normally given as boolean expressions:

```
myCut = "p_t > 500 & eta > pi/4 & eta < pi/2"
```

Using these criterion we can select events which are interesting to us. These selection cuts are usually based on the signature of whatever physical process that the user is studying. It is the user's responsibility to know the physical process he/she wants to study, as well as what variables it is appropriate to cut on.

Calculation of a quantity

As an analysis develops there may be a need to examine some quantity which is not directly observed in the detector. This is where the user specifies some quantity that he/she wants to calculate on an event by event basis. (An example of this will follow). ConfigurableAnalysis can also handle these types of arguments. These variables have all of the other properties of observed variables. This means that the user can cut on them if he/she wishes.

Fitting

In the current version of ConfigurableAnalysis, there is no implementation of a fitter. Once a user has developed a set of cuts to highlight his/her signal and has some calculated quantities, they may want to fit data to a model in order to make a measurement, such as a branching ratio, or cross section. This is where fitting comes in, the fitter takes the data, and a user supplied function and fits the function to the data, returning the parameters which fit the data.

Visualization of Data

One of the most important things in an analysis is actually being able to look at the data! ConfigurableAnalysis makes booking histograms trivial. It supports all of the main root histogram types, and creating groups of histograms is very simple. In the future we will have many examples of pre-defined histograms so that a new user can simply pick out which histograms he/she wants from a list and insert them into the Configuration code.

With this in mind, we can begin writing an analysis!

Set up your Environment

1.6.12

The instructions to install the analysis in 1.6 is not so trivial because the code is not part of the release. Some of the functionality of the ConfigurableAnalysis are not available in 1.6 (VariableComputer)

Refer to SWGuidePATRecipes to install the latest PAT software if needed.

```
cmsrel CMSSW_1_6_12
cd CMSSW_1_6_12/src
cmsenv

addpkg PhysicsTools/PatUtils
addpkg PhysicsTools/RecoUtils
addpkg PhysicsTools/UtilAlgos

cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/CachingVariable.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/ConfigurableHisto.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/EventSelector.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/InputTagDistributor.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/NTupler.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/Plotter.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/Selections.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/StringCutEventSelector.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/TwoObjectCalculator.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/VariableHelper.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/interface/UpdaterService.h

cvs co -r U00-00-00 PhysicsTools/UtilAlgos/plugins/ConfigurableAnalysis.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/plugins/NTuplingDevice.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/plugins/PlottingDevice.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/plugins/VariableEventSelector.h
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/plugins/modules.cc

cvs co -r U00-00-00 PhysicsTools/UtilAlgos/src/CachingVariable.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/src/InputTagDistributor.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/src/ToolsPluginsFactory.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/src/UpdaterService.cc
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/src/VariableHelper.cc

cvs co -r U00-00-00 PhysicsTools/UtilAlgos/data/configurableAnalysis.cfi
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/data/configurableAnalysis.cff
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/data/ntuplingDevice.cfi
cvs co -r U00-00-00 PhysicsTools/UtilAlgos/data/plottingDevice.cfi

cvs co -r V00-01-00 PhysicsTools/RecoUtils/plugins
cvs co -r V03-03-00 PhysicsTools/PatUtils/plugins

scramv1 build
```

2.1.9

Most of the code for ConfigurableAnalysis has entered the 2.1.X release cycle. However, some concrete implementation on reco and pat objects are not.

```
cmsrel CMSSW_2_1_9
cd CMSSW_2_1_9/src
cmsenv

#pat specific tags check on the twiki
```

```
cvs co -r V04-06-03 PhysicsTools/PatAlgos

#get the base for these packages
addpkg PhysicsTools/PatUtils

cvs co -r V00-01-02 PhysicsTools/RecoUtils
cvs co -r V03-03-01 PhysicsTools/PatUtils/plugins
cvs co -r V07-03-19 PhysicsTools/UtilAlgos

scramv1 build
```

Structure of ConfigurableAnalysis

The overall structure of the ConfigurableAnalysis is a list of event filters which defines selections. There is a plotting device is interleaved between the filtering step. An Ntupler empty interface is defined for the analysis user or group to define on their own and use it as a plugin.

Definition of variables

One of the main features of the ConfigurableAnalysis design is the VariableHelper class which holds a pool of variables computed from the Event. These variables can be computed and then cut on like any other parameter. They allow the end user to create his/her own analysis from scratch without needing to worry about writing lots of C++ code.

The VariableHelper class is accessed using the VariableHelperService which needs to be configured as follows

```
service = VariableHelperService {
              string printValuesForEachEventCategory = "VariableHelperServiceCategory"
}
```

The following is a listing of existing variables available for use in configurable analysis. For the benefit of the collaboration of people using the tools described in this Twiki, we ask the user to kindly report on any new variable(s) defined. This will allow central documentation and integration of the analysis code. It will also allow users to avoid "reinventing the wheel."

plugin name	functionnality	parameters
ExpressionVariable	string based variable for one object in a collection	InputTag src uint32 index string expr
CandidateExpressionVariable	concrete string variable for object deriving from reco::Candidate	
JetExpressionVariable	concrete string variable for pat::Jet	
METExpressionVariable	concrete string variable for pat::MET	
MuonExpressionVariable	concrete string variable for pat::Muon	
ElectronExpressionVariable	concrete string variable for pat::Electron	
TauExpressionVariable	concrete string variable for pat::Tau	
PhotonExpressionVariable	concrete string variable for pat::Photon	
CosDphiVariable<A,B>	calculate cos(delta Phi(A,B))	InputTag srcLhs uint32 indexLhs InputTag srcRhs uint32 indexRhs
JetMuonCosDphiVariable	concrete cosDphi for pat::Jet/pat::Muon	
JetMETCosDphiVariable	concrete cosDphi for pat::Jet/pat::MET	
JetJetCosDphiVariable	concrete cosDphi for pat::Jet/pat::Jet	
Power	calculate var^power	string var double power
ProcessIdSplitter	Discretised value of the CSA07 process Id	double lumi string weightLabel uint32 maxID
VarSplitter		

	The value of the variable is discretised according to configuration	string var bool useUnderFlow bool useOverFlow vdouble slots vstring labels string labelsFormat
DoubleVar	get a simple double value from the event	InputTag src
BoolVar	get a simple bool value from the event	InputTag src
DoubleVVar	gte a simple double in a vector of double from the event	InputTag src uint32 index
BoolVVar	gte a simple bool in a vector of bool from the event	InputTag src uint32 index
HLTBitVariable	return 0 or 1 depending on the status of the HLT bit specified by the name of the variable PSet	InputTag src

User-Defined Variables

Simple Variable

If ever an new variable has to calculated from the Event and this variable is not stored as a simple edm::product, it is possible to overload the class and define a Variable like

```
class myVariable : public CachingVariable {
 public:
  myVariable(CachingVariableFactoryArg arg) :
    CachingVariable("myVariable",arg.n,arg.iConfig){
      addDescriptionLine("I am a user defined variable doing...");
      addDescriptionLine("nothing interesting.");
      //retrieve configuration from iConfig
      arg.iConfig.getParameter<double>("value");
      //registration of the variable
      arg.m[arg.n]=this;
  }
  //concrete calculation of the variable
  CachingVariable::evalType eval(const edm::Event & iEvent) const {
     float someComputedValue;
     return std::make_pair(true,someComputedValue);
  }
}
```

And declare (the header files are left behing for clarity) the user variable to the plugin factor in a plugin library using

```
DEFINE_EDM_PLUGIN(CachingVariableFactory, myVariable, "myVariable");
```

In the case where many different quantities are computed at the same time and involving complicated calculation, having them in separate Variable will cause the computation to be done twice. To avoid this, the class VariableComputer (like a helper class) can calculate and evaluate mutliple Variable.

An example of overloading this class is given by VariableComputerTest

```
class VariableComputerTest : public VariableComputer {
 public:
  VariableComputerTest(CachingVariable::CachingVariableFactoryArg arg) ;
  ~VariableComputerTest(){};
```

```
  void compute(const edm::Event & iEvent) const;
};

VariableComputerTest::VariableComputerTest(CachingVariable::CachingVariableFactoryArg arg) : Vari
  declare("var1");
  declare("var2");
  declare("var3");
}

void VariableComputerTest::compute(const edm::Event & iEvent) const{
  //does some mumbo jumbo with the event.
  // computes a bunch of doubles
  double var1 = 3;
  double var2 = 4;

  //set the  variables  value (which do as if they had been cached)
  assign("var1",var1);
  assign("var2",var2);
  doesNotCompute("var3");
}
```

Eventually declared to the proper object plugin factor.

```
DEFINE_EDM_PLUGIN(VariableComputerFactory, VariableComputerTest, "VariableComputerTest");
```

This example, configured as follows

```
PSet computer={
             string method="ComputedVariable"
             string computer="VariableComputerTest"
             string separator="_"
}
```

will create three variable with label "computer_var1", "computer_var2" and "computer_var3". "computer_var3" (for the example) will be told to not be computable.

The **string separator** can be ommitted and is taken to be "_" by default.

Specific Variable Documentation

Some of the variable configurations demand more details than what is reported in the table above. If this is also the case for a user submitted variable, please kindly include the information which you would like included regarding the documentation of your variable and the ConfigurableAnalysis documentation people will add the explanation here.

ExpressionVariable

Here are some precisions about the configuration of any concrete type of the ExpressionVariable template class (CandidateExpressionVariable, MuonExpressionVariable, JetExpressionVariable,...).

The **string method** parameter has to specify one the available plugin names.

The **InputTag src** defines what is the collection of object used.

The **uint32 index** p parameter specifies the rank of the object to be considered. If the **uint32 index** parameter is not specified, the parameter **vuint32 indexes** has to be specified and the name of the PSet has to contain "_N'.

The **string expr** parameter defines (based on a parser) what expression has to be evaluated on the object and returned (semi-complicated expression can be used).If the **string expr** is not specified, the parameter **vstring vars** has to be specified and the name of the PSet has to contain "_V". The **vstring vars** has to be a vector of string under the format "name:expression"; "_N" will be replace by "name" and "expression" will be evaluated.

Starting in 2.1, there are additional functionality. The **string order** parameter will give the variable which defines the order of the vector of objects. The parameter is optional, but not untracked. Do not leave it empty if you don't want to use it, just remove it from configuration.

The **string selection** parameter determine which of the object in the collection should actually be considered. The parameter is optional, but not untracked. Do not leave it empty if you don't want to use it, just remove it from configuration.

Configuration Snippet

```
module configurableAnalysis = ConfigurableAnalysis
{
   PSet InputTags={}
   PSet Variables={
                PSet met_V={
                 string method = "METExpressionVariable"
                 InputTag src=selectedLayer1METs
                 uint32 index=0
                 vstring vars = { "Et:et", "Phi:phi", "X:px", "Y:py"}
                }
                PSet electron_N_V={
                 string method = "ElectronExpressionVariable"
                 InputTag src=selectedLayer1Electrons
                 vstring vars = { "Pt:pt" , "Eta:eta", "Phi:phi" , "CIso:caloIso" }
                 vuint32 indexes = { 0, 1 }
                }
                PSet jet_N_V={
                 string method = "JetExpressionVariable"
                 InputTag src=selectedLayer1Jets
                 vstring vars = { "Pt:pt" , "Et:et", "Eta:eta", "Phi:phi" }
                 vuint32 indexes = { 0, 1, 2 }
                }
                PSet jet2PtSplit={
                 string method="VarSplitter"
                 string var="jet2Pt"
                 bool useUnderFlow=false
                 bool useOverFlow=true
                 vdouble slots={0, 10, 40, 50, 100, 200, 500, 1000}
                 string labelsFormat="Jet_{2} p_{T} [%2.f , %2.f]"
                }
                PSet HLT_MuIso9={
                 InputTag src=TriggerResults::HLT
                }
   }#Variables

   PSet Selections={...}
   vstring flows={...}

   bool workAsASelector = true

   PSet Plotter={}
   PSet Ntupler={}
}
```

Which configure the variables met1Et, met1Eta, met1Phi, met1X, met1Y and electron1Pt, electron1Eta, electron1Phi, electron1CIso, electron2Pt, electron2Eta, electron2Phi, electron2CIso and jet1Pt, jet1Et, jet1Eta,

jet1Phi, jet2Pt, jet2Et, jet2Eta, jet2Phi, jet3Pt, jet3Et, jet3Eta, jet3Phi with their obvious meaning. The variable jet2PtSplit is a special variable which is able to split histograms in bins of the second jet pT (see histogramming in next section). The variable HLT_IsoMu9 is 0 or 1 depending on the status of the corresponding HLT bit named "HLT_IsoMu9".

Histogramming

The CMS.HamburgWikiAnalysisHistPlotter class is a generic class which can be overloaded by the user to re-define the CMS.HamburgWikiAnalysisHistPlotter interface. The CMS.HamburgWikiAnalysisHistPlotter are distributed by the PlotterFactory and therefore must implement the constructor with ParameterSet and the pure virtual methods setDir, fill and complete.

VariableHamburgWikiAnalysisHistPlotter is one already defined HamburgWikiAnalysisHistPlotter which uses the variable already defined and is the recommended HamburgWikiAnalysisHistPlotter to use. However, nothing prevents experienced user to redefine their own HamburgWikiAnalysisHistPlotter.

```
class Plotter {
public:
   Plotter(){}
   Plotter(edm::ParameterSet iConfig){}
   virtual ~Plotter(){}

   virtual void setDir(std::string dir) =0;
   virtual void fill(std::string subDir,const edm::Event& iEvent) =0;
   virtual void complete() =0;
};
```

The user defined CMS.HamburgWikiAnalysisHistPlotter class has then to be declared as a plugins using:

```
#include "CMS.PhysicsTools/UtilAlgos/interface/Plotter.h"
DEFINE_EDM_PLUGIN(PlotterFactory, UserDefinedPlotter, "UserDefinedPlotter");
```

The user defined CMS.HamburgWikiAnalysisHistPlotter is then summoned in the configuration file

```
string ComponentName = "UserDefinedPlotter"
```

as explained later on.

Configuration Snipet

Here is the configuration snipet for the VariableHamburgWikiAnalysisHistPlotter which is a concrete implementation of the HamburgWikiAnalysisHistPlotter interface, using only the variables defined in the PSet Variables={...}.

```
module configurableAnalysis = ConfigurableAnalysis
{
   PSet InputTags={}
   PSet Variables={...}

   PSet Selections={...}
   vstring flows={...}

   bool workAsASelector = true

   PSet Plotter={
            string ComponentName = "VariablePlotter"
            PSet TH1s={
                PSet hEl1pT={
                        vstring splitters={ "jet2PtSplit" }
```

```
                        string title="Electron_{1} p_{T}"
                        PSet xAxis={
                                string var="electron1Pt"
                                string Label="leading electron p_{T} [GeV]"
                                uint32 nBins=600
                                double Min=0
                                double Max=1200
                        }
                    }
                    PSet hEl2pT={
                            string title="Electron_{2} p_{T}"
                            PSet xAxis={
                                    string var="electron2Pt"
                                    string Label="second electron p_{T} [GeV]"
                                    vdouble vBins={0,10,30,50,100,600,1200}
                            }
                    }
                }#TH1s

                PSet TH2s={
                    PSet hEl1PtvsEl2Pt={
                            string title="Electron_{1} p_{T} versus Electron_{2} p_{T}"
                            string weight="jet1Eta"
                            PSet xAxis={
                                    string var="electron1Pt"
                                    string Label="leading electron p_{T} [GeV]"
                                    uint32 nBins=600
                                    double Min=0
                                    double Max=1200
                            }
                            PSet yAxis={
                                    string var="electron2Pt"
                                    string Label="second electron p_{T} [GeV]"
                                    vdouble vBins={0,10,30,50,100,600,1200}
                            }
                    }
                }#TH2s

                PSet TProfiles={}
    }
    PSet Ntupler={}
}
```

Which defines the histograms

- "hEl1pT" of the leading electron pT, and the sub-sequent "hEl1pT_jet2PtSplit_*" histograms split according to the second jet Pt.
- "hEl2pT" of the second electron pT with variable sized binning.
- "hEl1PtvsEl2Pt" 2D plot of leading electron pT versus second electron pT, with variable sized binning on the y axis and the entries are weighted according to "jet1Eta" (as an example).

TProfile are defined with the same interface as the TH2s since both x and y axis have to be defined.

TH1, TH2, TProfile can be split using

```
vstring splitter = {...}
```

Filtering

Another main feature of the ConfigurablAnalysis is the event filtering.

Filter are simple class which holds a pointer to an EventSelector class. The Filter class is used internally and only the EventSelector class should be specified or over loaded by the user. The EventSelector class (originally from [[][Susy Pat]]) has a simple interface

```
class EventSelector {
public:
  virtual bool select (const edm::Event&) const = 0;
};
```

which has to be overloaded by a concrete class. Here is a list of concrete classes already defined

plugin name	functionnality	parameters
HLTEventSelector	retrieve edm::TriggerResults and select events base on HLT bit	InputTag triggerResults vstring pathNames
StringCutEventSelector	string base selector on n first objects in a collection. Will not fail if not enough object in the collection. nFirst=0 will let the cut be applied to **all** object in the collection	InputTag src string cut uint32 nFirst
JetEventSelector	concrete pat::Jet selector	InputTag src string cut uint32 nFirst
MuonEventSelector	concrete pat::Muon selector	InputTag src string cut uint32 nFirst
METEventSelector	concrete pat::MET seletor	InputTag src string cut uint32 nFirst
ElectronEventSelector	concrete pat::Electron selector	InputTag src string cut uint32 nFirst
PhotonEventSelector	concrete pat::Photon selector	InputTag src string cut uint32 nFirst
CandidateEventSelector	concrete selector for any class deriving from reco::Candidate	InputTag src string cut" uint32 nFirst
StringCutsEventSelector	string base selector on a collection of objects. "-" string will make no cut on the object.	InputTag src vstring cut
JetSEventSelector	concrete selector on pat::Jets	InputTag src vstring cut
MuonSEventSelector	concrete selector on pat::Muons	InputTag src vstring cut
METSEventSelector	concrete selector on pat::MET	InputTag src vstring cut
ElectronSEventSelector	concrete pat::Electron selector	InputTag src vstring cut
PhotonSEventSelector	concrete pat::Photon selector	InputTag src vstring cut
CandidateSEventSelector	concrete selector for any class deriving from reco::Candidate	InputTag src vstring cut
StringCutsEventSelector<A,false>	string base selector on a collection of objects. "-" string will make no cut on the object. it is the best way to implement a **veto**	InputTag src vstring cut

JetSEventVetoSelector	concrete selector on pat::Jets	InputTag src vstring cut
MuonSEventVetoSelector	concrete veto pat::Muons selector	InputTag src vstring cut
METSEventVetoSelector	concrete veto pat::MET selector	InputTag src vstring cut
ElectronEventVetoSelector	concrete veto pat::Electron selector	InputTag src vstring cut
PhotonEventVetoSelector	concrete vetor pat::Photon selector	InputTag src vstring cut
CandidateSEventVetoSelector	concrete veto selector for object deriving from reco::Candidate	InputTag src vstring cut
VariableEventSelector	event selector on variable provide by VariableHelper. not mentionning min (max) parameter will void the cut on the minimum (maximum)	string var double min double max

Configuration Snipet

```
module configurableAnalysis = ConfigurableAnalysis
{
   PSet InputTags={}
   PSet Variables={}

   PSet Selections={
      PSet filters={
                        PSet triggers={
                        string selector="HLTEventSelector"
                        InputTag triggerResults=TriggerResults::HLT
                        vstring pathNames = { "HLT1MuonIso", "HLT2MuonIso" }
                        }
                        PSet leadingMuon={
                        string selector="MuonSEventSelector"
                        InputTag src=selectedLayer1Muons
                        vstring cut = { "pt > 30.0 & abs(eta) < 3.0" }
                        }
                        PSet leadingElectron={
                        string selector="ElectronSEventSelector"
                        InputTag src=selectedLayer1Electrons
                        vstring cut = { "pt > 30.0 & abs(eta) < 3.0" }
                        }

                        PSet jets={
                        string selector="JetSEventSelector"
                        InputTag src=selectedLayer1Jets
                        vstring cut = { "et > 150.0 & abs(eta) < 1.0", "-", "et > 50.0 & abs(eta)
                        }
      }#filters
      PSet selections={
                        PSet triggerSelection={
                                vstring filterOrder = { "triggers" }
                                bool ntuplize=false
                                bool makeContentPlots=false
                                bool makeFinalPlots=false
                                bool makeCumulativePlots=false
                                bool makeAllButOnePlots=false
                                bool makeSummaryTable=false
                        }
                        PSet preSelection={
                                vstring filterOrder = { "leadingMuon" }
                                bool ntuplize=false
                                bool makeContentPlots=false
                                bool makeFinalPlots=false
```

```
                                bool makeCumulativePlots=false
                                bool makeAllButOnePlots=false
                                bool makeSummaryTable=false
                        }
                        PSet finalSelection={
                                vstring filterOrder = { "leadingMuon_OR_leadingElectron" , "jets"
                                bool ntuplize=false
                                bool makeContentPlots=true
                                bool makeFinalPlots=true
                                bool makeCumulativePlots=true
                                bool makeAllButOnePlots=true
                                bool makeSummaryTable=true
                                uint32 nMonitor=1000
                                string detailledPrintoutCategory="finalSelectionCategory"
                        }
        }#selections
   }#Selection
   vstring flows={ "finalSelection" }

   bool workAsASelector = true

   PSet Plotter={}
   PSet Ntupler={}
}
```

Which defines three actual filters to accept events with

- the HLT bit to be an OR of "HLT1MuonIso" and "HLT2MuonIso".
- the leading muon (first in the collection) to be within 3.0 of pseudo rapidity and with more than 30 GeV of transverse momentum.
- at least three jets, the leading jet has more than 150 GeV of transverse energy, and be within a pseudo rapidity of 1. There is no requirement on the second leading jet, and the thir jet must have more than 50 GeV of transverse energy and be within a pseudo rapidity of 3.

The "finalSelection" actually apply the three filters from "leadingMuon_OR_leadingElectron" (or of filters "leadingMuon" and "leadingElectron"), "jets" (filter) and "triggerSelection" (selection). For this selection, the ConfigurableAnalysis will fill plots for every events before any cut (**makeContentPlots=true**), plots for events passing all the filters (**makeFinalPlots=true**), plots after each filter (**makeCumulativePlots=true**), plots for events which pass all the filters except one of them (**makeAllButOnePlots=true**).

The module will print a summary table (**makeSummaryTable=true**) every 1000 events read (**nMonitor=1000**), and after all events are read. If **nMonitor** is set to 0, then the printout does not happen for every events, only at the end.

If **string detailledPrintoutCategory** is specified, then a detailled printout of which filter passes which event is send to the specified category through the message logger.

Because in the old-style configuration file, there are no easy way to replace the many InputTag specified in the configuration file. In order to be able to replace all input tags at once, one should use the **PSet InputTags** (which uses internally the InputTagDistributorService) as follows

```
PSet InputTags={
               InputTag jets=selectedLayer1Jets
               InputTag electrons=selectedLayer1Electrons
               InputTag muons=selectedLayer1Muons
               InputTag mets=selectedLayer1METs
}
```

And replace InputTag parameter by a string parameter to specify the alias name rather than the actual InputTag in most of the tools described in this twiki. For example

```
PSet jet_N_V={
                  string method = "JetExpressionVariable"
                  string src="jets"
                  vstring vars = { "Pt:pt" , "Et:et", "Eta:eta", "Phi:phi" }
                  vuint32 indexes = { 0, 1, 2 }
}
```

Instead of

```
PSet jet_N_V={
                  string method = "JetExpressionVariable"
                  InputTag src=selectedLayer1Jets
                  vstring vars = { "Pt:pt" , "Et:et", "Eta:eta", "Phi:phi" }
                  vuint32 indexes = { 0, 1, 2 }
}
```

Usage in User Code

If the user has to write or overload any of the tools described here and wish to have the InputTagDistributor functionality, just do

```
edm::InputTag src_=edm::Service<InputTagDistributorService>()->retrieve("src",iConfig);
```

where src is the name of the parameter, and iConfig is the parameter set which contains either **InputTag src** or **string src**.

Ntupler

No documentation is given one this twiki since no Ntupler is centrally maintain by the physics analysis group. The user is welcome to link here twiki of specific NTupler they have created.

- A Variable based Ntupler
- A string based Ntupler (check this one out, it is very functionnal !)
- string based ntuple documentation (not officially supported)

Complete Configuration Example

Configuration in 1.6.12

Get the three following files in the src directory

- testFromAOD.cfg: testFromAOD.cfg
- configurableAnalysis.cfi: configurableAnalysis.cfi
- configurableAnalysis.cff: configurableAnalysis.cff

```
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookConfigurableAnalysis/configurableAnalysis.cfi -O
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookConfigurableAnalysis/configurableAnalysis.cff -O
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookConfigurableAnalysis/testFromAOD.cfg -O testFrom
```

And run testFromAOD.cfg to run the PAT on the fly and a test version of the ConfigurableAnalysis

Configuration in 2.1.9

Get the three following files in the src directory

- configurableAnalysis_cff.py.txt: configurableAnalysis_cff.py
- configurableAnalysis_cfi.py.txt: configurableAnalysis_cfi.py
- testFromAOD_cfg.py.txt: testFromAOD_cfg.py

```
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookConfigurableAnalysis/configurableAnalysis_cfi.py
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookConfigurableAnalysis/configurableAnalysis_cff.py
wget https://twiki.cern.ch/twiki/pub/CMS/WorkBookConfigurableAnalysis/testFromAOD_cfg.py.txt -O t
```

And run testFromAOD_cfg.py to run the PAT on the fly and a test version of the ConfigurableAnalysis

Information Sources

Review status

Reviewer/Editor and Date (copy from screen)	Comments
your name	What you did (e.g., created page, reviewed, edited, added/answered questions or comments, etc.) and/or what you think needs to be done next (e.g., "needs review by ...", "still needs content on subject x", etc.)
JeanrochVlimant - 28 Sep 2008	almost finalized documentation
DavidBjergaard - 22 Sep 2008	Re-ordered things to flow better, adjusted aesthetics, explained components of ExpressionVariable PSets (needs to be cross-checked by Jean-Roch
JeanrochVlimant - 21 Sep 2008	Fill in documentation for histogramming, variable definition and filtering
DavidBjergaard - 21 Sep 2008	Outlined skeleton and began to flesh out analysis sections
KatiLassilaPerini - 17 Sep 2008	created the template page

Responsible: ResponsibleIndividual
Last reviewed by: YourName - 06 Dec 2008

c

This topic: CMSPublic > WorkBookConfigurableAnalysis
Topic revision: r24 - 2009-05-27 - RogerWolf