# Table of Contents

# Example of Dynamically Generated Analysis Code

Here below you can find the Analysis Code, automatically generated from this example *steering file*

The generated code is at once ready to be run with Athena or ARA, locally or on the Grid, with automatically generated `jobOption` files and Panda and Ganga launchers.

---

```
1. #!python
2.
3. ###############################################################################
4. #                                                             #
5. #   Code automatically generated with the package                      #
6. #                                                             #
7. #                ATLASWatchMan                                #
8. #                                                             #
9. #   Copyright (C) 2008-2009 by                                #
10. #                                                            #
11. #      Riccardo-Maria BIANCHI                                #
12. #             and                                            #
13. #       Renaud BRUNELIERE                                    #
14. #                                                            #
15. #      (Freiburg University, Germany)                        #
16. #                                                            #
17. #   For comments and questions:   rbianchi@cernNOSPAMPLEASE.ch,
    bruneli@cernNOSPAMPLEASE.ch      #
18. #                                                            #
19. #   ----------                                               #
20. #   This program is free software; you can redistribute it and/or modify  #
21. #   it under the terms of the GNU General Public License as published by  #
22. #   the Free Software Foundation; either version 2 of the License, or     #
23. #   (at your option) any later version.                      #
24. #                                                            #
25. #   This program is distributed in the hope that it will be useful,       #
26. #   but WITHOUT ANY WARRANTY; without even the implied warranty of        #
27. #   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         #
28. #   GNU General Public License for more details.                         #
29. #                                                            #
30. #   You should have received a copy of the GNU General Public License     #
31. #   along with this program; it should be in the COPYING file.            #
32. #   If not, write to the                                     #
33. #   Free Software Foundation, Inc.,                          #
34. #   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.             #
35. ###############################################################################
36.
37.
38.
39. #@note: Based on an idea of Sascha CARON
40. #@author: Riccardo-Maria BIANCHI <rbianchi@cern.ch>, Renaud BRUNELIERE
    <bruneli@cern.ch>
41. import AthenaCommon.SystemOfUnits as Units
42. from AthenaPython.PyAthena import StatusCode
43. import AthenaPython.PyAthena as PyAthena
44. import FourMomUtils.Bindings #to use the 'P4Helpers' namespace via PyAthena
```

```
45. import SUSYTools.Bindings
46. import SUSYTools.SUSYDefaultOptsLib
47. import SUSYTools.TriggerToolsLib
48. import SUSYTools.SUSYObjSelection
49. import SUSYTools.SUSYOverlapRemoval
50. import SUSYTools.AthenaObjWrapper
51. import ATLASWatchMan.Bindings
52. import math
53. import ROOT
54. import sys
55. class ATLASWatchMan_AnalysisAlg(PyAthena.Alg):
56.     """Class generated by the ATLASWatchMan package,
57.    containing the analysis code."""
58.
59.
60.
61.     ###########################################################
62.     def __init__(self, name='ATLASWatchMan_AnalysisAlg', **kw):
63.         ## init the base class
64.         kw['name'] = name
65.         super(ATLASWatchMan_AnalysisAlg,self).__init__(**kw)
66.
67.         self.name = kw['name']
68.         self.printLevel = kw.get('PrintLevel',5)
69.
70.
71.         ## provide default values for properties
72.         self.runARA                = kw.get('runARA', False) #By default we run it on Grid with Athena
73.         self.doSkimmingGlobal      = kw.get('doSkimmingGlobal', True) #By default we want global
    skimming for all datasets: only events passing all the cuts for at least one channel are kept.
74.         self.doEventSelectionEvenIfNotSkimming = kw.get( 'doEventSelectionEvenIfNotSkimming',
     False) #By default we do not want to perform eventSelection while not skimming
75.         self.runAtlfast1           = kw.get('runAtlfast1', False) #By default do not run on Atlfast1 file
76.         self.isTriggerFilter       = kw.get('isTriggerFilter', False) #By def. we don't check if the event
    passed given triggers
77.         self.hsvcInitialized       = False
78.
79.         self.channelsList1lep = []
80.         self.channelsList3j1lepMediumCutsMTSmaller100GeV = []
81.         self.channelsList3j1lepMediumCutsMTBigger100GeV = []
82.         ## adding lists and dicts to store channels info
83.         self.channelsPassed = {}
84.         self.channelsList = []
85.         self.channelsPassed1lep = {}
86.         self.channelsList1lep = []
87.         self.channelsPassed4j0lepCSC = {}
88.         self.channelsList4j0lepCSC = []
89.         self.channelsPassed3j1lepMediumCutsMTSmaller100GeV = {}
90.         self.channelsList3j1lepMediumCutsMTSmaller100GeV = []
91.         self.channelsPassed3j1lepMediumCutsMTBigger100GeV = {}
92.         self.channelsList3j1lepMediumCutsMTBigger100GeV = []
93.
94.
95.         ## set tree variables
```

```
96.         ## The variables will be linked to ROOT variables.
97.         self.treeVar = {
98.         'jet': {'4mom': ROOT.std.vector(ROOT.TLorentzVector)(), 'Channels':
    ROOT.std.vector(ROOT.std.vector(ROOT.std.string))()},
99.         'electron':{'4mom': ROOT.std.vector(ROOT.TLorentzVector)(),'Charge': ROOT.std.vector(int)(),
    'Channels': ROOT.std.vector(ROOT.std.vector(ROOT.std.string))()},
100.        'muon':{'4mom': ROOT.std.vector(ROOT.TLorentzVector)(),'Charge': ROOT.std.vector(int)(),
    'Channels': ROOT.std.vector(ROOT.std.vector(ROOT.std.string))()},
101.        'met':{'Et': ROOT.std.vector(float)(), 'Phi': ROOT.std.vector(float)(), 'Channels':
    ROOT.std.vector(ROOT.std.vector(ROOT.std.string))()},
102.        'eventNumber': ROOT.std.vector(int)(),
103.        'runNumber': ROOT.std.vector(int)(),
104.        'weightMC': ROOT.std.vector(float)(),
105.        'channels': ROOT.std.vector(ROOT.std.string)(),
106.        'channelsTopology': ROOT.std.vector(ROOT.std.string)(),
107.        }
108.
109.        #extra branches user-defined
110.        self.extraTreeVar = {
111.        'sphCSC': { 'Values': ROOT.std.vector(float)(), 'Channels': ROOT.std.vector(ROOT.std.string)()
    },
112.        'TM_CSC': { 'Values': ROOT.std.vector(float)(), 'Channels':
    ROOT.std.vector(ROOT.std.string)() },
113.        'meff4j': { 'Values': ROOT.std.vector(float)(), 'Channels': ROOT.std.vector(ROOT.std.string)() },
114.        }
115.
116.        # Building the dict containing the formulae for user-defined extra branches
117.        self.extraTreeVarFormula = {
118.        'sphCSC': 'sphericity',
119.        'TM_CSC': 'transverseMass',
120.        'meff4j': 'meff',
121.        }
122.        self.extraTreeVarFormulaInverse = {
123.        'sphericity': 'sphCSC',
124.        'transverseMass': 'TM_CSC',
125.        'meff': 'meff4j',
126.        }
127.        self.extraBranches = {
128.        'jetenergy_FCAL2': ROOT.std.vector(float)(),
129.        'jetenergy_FCAL1': ROOT.std.vector(float)(),
130.        'jetenergy_FCAL0': ROOT.std.vector(float)(),
131.        'electronAuthor': ROOT.std.vector(int)(),
132.        'METSigL': ROOT.std.vector(float)(),
133.        'truthPdgid': ROOT.std.vector(int)(),
134.        }
135.
136.        ## set histos
137.        self.histoVar = {}
138.
139.        ## set default selection cuts and collection names
140.        self.cuts = ROOT.SUSYTools.SUSYDefinitionsCSC()
141.        if self.runAtlfast1:
142.            self.cuts.electron.applyOnlyAcceptanceCuts=1
143.            self.cuts.photon.applyOnlyAcceptanceCuts=1
```

```
144.            self.cuts.muon.applyOnlyAcceptanceCuts=1
145.            self.cuts.tau.applyOnlyAcceptanceCuts=1
146.            pass
147.        self.defObjs = SUSYTools.SUSYDefaultOptsLib.SUSYDefaultObjs()
148.        self.defObjs1lep = SUSYTools.SUSYDefaultOptsLib.SUSYDefaultObjs()
149.        self.defObjs4j0lepCSC = SUSYTools.SUSYDefaultOptsLib.SUSYDefaultObjs()
150.        self.defObjs3j1lepMediumCutsMTSmaller100GeV =
    SUSYTools.SUSYDefaultOptsLib.SUSYDefaultObjs()
151.        self.defObjs3j1lepMediumCutsMTBigger100GeV =
    SUSYTools.SUSYDefaultOptsLib.SUSYDefaultObjs()
152.        self.collections = self.defObjs.collections
153.        import ATLASWatchMan.ATLASWatchMan_GeneratedUserCutsAndChannelsLib
154.        self.cuts2 = ATLASWatchMan.ATLASWatchMan_GeneratedUserCutsAndChannelsLib
155.
156.        ## modifying default object-definition/overlap-removal/event-selection cuts for some channels
157.
158.        ## get handles to general tools
159.        self.triggerTools   = SUSYTools.TriggerToolsLib.TriggerTools() #This gives some warnings in
    the output
160.        #    initializing the objects selection with default cuts
161.        self.objSelection   =
    SUSYTools.SUSYObjSelection.SUSYObjSelection('SUSYObjSelection',self.printLevel,self.cuts)
162.        #    initializing the overlap removal function with default cuts
163.        self.overlapRemoval =
    SUSYTools.SUSYOverlapRemoval.SUSYOverlapRemoval('SUSYOverlapRemoval',self.printLevel,self.cuts)
164.
165.
166.        ## adding extra user-defined Collections to collections)
167.        self.collections['muon']['select'] = True #Updating a collection
168.        self.collections['jet']['select'] = True #Updating a collection
169.        self.collections['track'] = {'type': 'Rec::TrackParticleContainer', 'name': 'TrackParticleCandidate',
    'select': False}  #Adding a new collection
170.        self.collections['electron']['select'] = True #Updating a collection
171.        self.collections['METSig'] = {'type': 'MissingETSig', 'name': 'METSig', 'select': False}  #Adding
    a new collection
172.        self.collections['truth']['select'] = False #Updating a collection
173.        self.collections['truth_jet']['select'] = True #Updating a collection
174.        self.collections['gen']['select'] = False #Updating a collection
175.
176.        ## set properties
177.        if self.defObjs.checkTrigger == True: self.isTriggerFilter = True
178.
179.        ## User has defined some channels with custom object-definition/overalp-removal cuts
180.        # So in order to use the user-defined cuts we do:
181.        #    1) taking default cuts
182.        #    2) modifying the default cuts with user-defined custom cuts fot this channel
183.        #    3) initializing the objects selection with user-defined custom cuts for this channel
184.        #    4) initializing the overlap removal function with user-defined custom cuts for this channel
185.
186.        self.customCutsChannels = []
187.        # --- Channel: 1lep. - Overwriting default cuts with user-defined custom cuts ---
188.        self.customCutsChannels.append('1lep')
189.        self.cuts1lep = ROOT.SUSYTools.SUSYDefinitionsCSC()
190.        self.cuts1lep.muon.ptMin = 5000.0
```

```
191.         self.cuts1lep.electron.ptMin = 5000.0
192.         if self.runAtlfast1:
193.             self.cuts1lep.electron.applyOnlyAcceptanceCuts=1
194.             self.cuts1lep.photon.applyOnlyAcceptanceCuts=1
195.             self.cuts1lep.muon.applyOnlyAcceptanceCuts=1
196.             self.cuts1lep.tau.applyOnlyAcceptanceCuts=1
197.             pass
198.         self.objSelection1lep =
    SUSYTools.SUSYObjSelection.SUSYObjSelection('SUSYObjSelection',self.printLevel,self.cuts1lep)
199.         self.overlapRemoval1lep =
    SUSYTools.SUSYOverlapRemoval.SUSYOverlapRemoval('SUSYOverlapRemoval',self.printLevel,self.cuts1
200.         if self.defObjs1lep.checkTrigger == True: self.isTriggerFilter1lep = True
201.         # --- Channel: 4j0lepCSC. - Overwriting default cuts with user-defined custom cuts ---
202.         self.customCutsChannels.append('4j0lepCSC')
203.         self.cuts4j0lepCSC = ROOT.SUSYTools.SUSYDefinitionsCSC()
204.         self.cuts4j0lepCSC.muon.ptMin = 20000.0
205.         self.cuts4j0lepCSC.electron.ptMin = 20000.0
206.         self.cuts4j0lepCSC.photon.applyOverlapRemoval = False
207.         self.cuts4j0lepCSC.tau.applyOverlapRemoval = False
208.         if self.runAtlfast1:
209.             self.cuts4j0lepCSC.electron.applyOnlyAcceptanceCuts=1
210.             self.cuts4j0lepCSC.photon.applyOnlyAcceptanceCuts=1
211.             self.cuts4j0lepCSC.muon.applyOnlyAcceptanceCuts=1
212.             self.cuts4j0lepCSC.tau.applyOnlyAcceptanceCuts=1
213.             pass
214.         self.objSelection4j0lepCSC =
    SUSYTools.SUSYObjSelection.SUSYObjSelection('SUSYObjSelection',self.printLevel,self.cuts4j0lepCSC)
215.         self.overlapRemoval4j0lepCSC =
    SUSYTools.SUSYOverlapRemoval.SUSYOverlapRemoval('SUSYOverlapRemoval',self.printLevel,self.cuts4
216.         if self.defObjs4j0lepCSC.checkTrigger == True: self.isTriggerFilter4j0lepCSC = True
217.         # --- Channel: 3j1lepMediumCutsMTSmaller100GeV. - Overwriting default cuts with
    user-defined custom cuts ---
218.         self.customCutsChannels.append('3j1lepMediumCutsMTSmaller100GeV')
219.         self.cuts3j1lepMediumCutsMTSmaller100GeV = ROOT.SUSYTools.SUSYDefinitionsCSC()
220.         self.cuts3j1lepMediumCutsMTSmaller100GeV.muon.ptMin = 20000.0
221.         self.cuts3j1lepMediumCutsMTSmaller100GeV.electron.ptMin = 20000.0
222.         self.cuts3j1lepMediumCutsMTSmaller100GeV.photon.applyOverlapRemoval = False
223.         self.cuts3j1lepMediumCutsMTSmaller100GeV.tau.applyOverlapRemoval = False
224.         if self.runAtlfast1:
225.             self.cuts3j1lepMediumCutsMTSmaller100GeV.electron.applyOnlyAcceptanceCuts=1
226.             self.cuts3j1lepMediumCutsMTSmaller100GeV.photon.applyOnlyAcceptanceCuts=1
227.             self.cuts3j1lepMediumCutsMTSmaller100GeV.muon.applyOnlyAcceptanceCuts=1
228.             self.cuts3j1lepMediumCutsMTSmaller100GeV.tau.applyOnlyAcceptanceCuts=1
229.             pass
230.         self.objSelection3j1lepMediumCutsMTSmaller100GeV =
    SUSYTools.SUSYObjSelection.SUSYObjSelection('SUSYObjSelection',self.printLevel,self.cuts3j1lepMediu
231.         self.overlapRemoval3j1lepMediumCutsMTSmaller100GeV =
    SUSYTools.SUSYOverlapRemoval.SUSYOverlapRemoval('SUSYOverlapRemoval',self.printLevel,self.cuts3
232.         if self.defObjs3j1lepMediumCutsMTSmaller100GeV.checkTrigger == True:
    self.isTriggerFilter3j1lepMediumCutsMTSmaller100GeV = True
233.         # --- Channel: 3j1lepMediumCutsMTBigger100GeV. - Overwriting default cuts with
    user-defined custom cuts ---
234.         self.customCutsChannels.append('3j1lepMediumCutsMTBigger100GeV')
235.         self.cuts3j1lepMediumCutsMTBigger100GeV = ROOT.SUSYTools.SUSYDefinitionsCSC()
```

```
236.        self.cuts3j1lepMediumCutsMTBigger100GeV.muon.ptMin = 20000.0
237.        self.cuts3j1lepMediumCutsMTBigger100GeV.electron.ptMin = 20000.0
238.        self.cuts3j1lepMediumCutsMTBigger100GeV.photon.applyOverlapRemoval = False
239.        self.cuts3j1lepMediumCutsMTBigger100GeV.tau.applyOverlapRemoval = False
240.        if self.runAtlfast1:
241.            self.cuts3j1lepMediumCutsMTBigger100GeV.electron.applyOnlyAcceptanceCuts=1
242.            self.cuts3j1lepMediumCutsMTBigger100GeV.photon.applyOnlyAcceptanceCuts=1
243.            self.cuts3j1lepMediumCutsMTBigger100GeV.muon.applyOnlyAcceptanceCuts=1
244.            self.cuts3j1lepMediumCutsMTBigger100GeV.tau.applyOnlyAcceptanceCuts=1
245.            pass
246.        self.objSelection3j1lepMediumCutsMTBigger100GeV =
    SUSYTools.SUSYObjSelection.SUSYObjSelection('SUSYObjSelection',self.printLevel,self.cuts3j1lepMediu
247.        self.overlapRemoval3j1lepMediumCutsMTBigger100GeV =
    SUSYTools.SUSYOverlapRemoval.SUSYOverlapRemoval('SUSYOverlapRemoval',self.printLevel,self.cuts3
248.        if self.defObjs3j1lepMediumCutsMTBigger100GeV.checkTrigger == True:
    self.isTriggerFilter3j1lepMediumCutsMTBigger100GeV = True
249.        return
250.
251.
252.
253.
    ##########################################################################################
254.    def initialize(self):
255.        '''It initializes the Athena services.
256.       Note! This is called automatically by the Athena jobOption,
257.       but it has to be called explicitly by the ARA jobOption.'''
258.
259.        if self.printLevel < 5: print self.name,'.initialize(): starting...'
260.
261.        if not self.runARA: #the following is used only by Athena
262.            ## We instantiate the handle to the StoreGate
263.            self.sg = PyAthena.py_svc('StoreGateSvc')
264.            if not self.sg:
265.                self.msg.error('Could not retrieve StoreGateSvc')
266.                return StatusCode.Failure
267.
268.        self.hsvc = PyAthena.py_svc('THistSvc/THistSvc')
269.        if not self.hsvc:
270.            self.msg.error('Could not retrieve THistSvc/THistSvc')
271.            return StatusCode.Failure
272.
273.        self.bookHists(self.treeVar, self.extraTreeVar, self.extraBranches, 'D3PDTree')
274.
275.        if self.printLevel < 3: print self.name,'.initialize(): ...done'
276.        return StatusCode.Success
277.
278.
279.
280.
    ##########################################################################################
281.    def bookHists(self, treeVar, extraTreeVar, extraBranches, treeName):
282.        '''Booking output TTree branches and histograms'''
283.        if self.printLevel < 5: print self.name,'.bookHists(): booking histograms and ttrees/branches for
    tree', treeName, '...'
```

```
284.
285.        if not self.hsvcInitialized:
286.            self.hsvcInitialized = True
287.            if self.runARA: self.hsvc = {}
288.
289.            #Build a histo to store number of generated events before any cuts. In order to normalize
    samples.
290.            self.hsvc['/rec/NEvents'] = ROOT.TH1F('NEvents','number of events before any cut',1,0.,1.)
291.            #Build a histo to store sum of MC weights number of generated events before any cuts. In
    order to normalize samples.
292.            self.hsvc['/rec/sumWeightsMC'] = ROOT.TH1F('sumWeightsMC','sum of MC weights of
    events before any cut',1,-1.,1.)
293.
294.        # Build a TTree which stores plot variables
295.        self.hsvc['/rec/'+treeName] = ROOT.TTree(treeName,treeName) #
296.
297.        for part in treeVar: #looping over particles to build branches
298.            if self.printLevel < 2: print 'part: %s, type: %s' % (part, str(type(treeVar[part])) )
299.            #I declare the branches, with name and type
300.            if type(treeVar[part]) == dict:
301.                for var in treeVar[part]:
302.                    if self.printLevel < 2: print 'part: %s, var: %s' % (part, var)
303.                    treeVar[part][var].reserve(10) #???vedi un po'
304.                    self.hsvc['/rec/'+treeName].Branch(part+var,treeVar[part][var])
305.                    pass
306.                pass
307.            else:
308.                self.hsvc['/rec/'+treeName].Branch(part,treeVar[part])
309.            pass
310.        for part in extraTreeVar: #looping over extra user-defined branches
311.            if self.printLevel < 3: print 'Looping on user-defined extra branches'
312.            if self.printLevel < 2: print 'part: %s, type: %s' % (part, str(type(extraTreeVar[part])) )
313.            if type(extraTreeVar[part]) == dict:
314.                for var in extraTreeVar[part]:
315.                    self.hsvc['/rec/'+treeName].Branch(part+var,extraTreeVar[part][var])
316.            else:
317.                self.hsvc['/rec/'+treeName].Branch(part,extraTreeVar[part])
318.        #looping over 'extraBranches', built from 'dumpContainers' in the steering file
319.        for br in extraBranches:
320.            self.hsvc['/rec/'+treeName].Branch(br,extraBranches[br])
321.        pass
322.
323.        if self.printLevel < 3: print self.name,'.bookHists(): ...done'
324.        return StatusCode.Success
325.
326.
327.
328.
    ##############################################################################################
329.    def execute(self):
330.        '''This retrieves StoreGate containers and then it call analyze().
331.        It is used only by Athena, not ARA.'''
332.
333.        if self.printLevel < 5: print self.name,'.execute: running...'
```

```
334.
335.        ## Read storegate containers
336.        for collection in self.collections:
337.            collectionName = self.collections[collection]['name']
338.            if self.runAtlfast1 and 'atlfast1' in self.collections[collection]: collectionName =
    self.collections[collection]['atlfast1']
339.            collectionType = self.collections[collection]['type']
340.            if self.printLevel < 3: print self.name,('.execute(): retrieve [%s/%s]' % (collectionType,
    collectionName))
341.            container = self.sg.retrieve(collectionType, collectionName)
342.            if not container:
343.                if self.printLevel < 3:
344.                    print self.name,('.execute(): Could not retrieve [%s] at [%s]' % (collectionType,
    collectionName))
345.                return StatusCode.Recoverable
346.            ## I store an handle to the retrieved container inside the dict
347.            self.collections[collection]['container'] = container
348.            pass
349.
350.        self.analyze()
351.
352.        if self.printLevel < 5: print self.name,'.execute(): ...done'
353.        return StatusCode.Success
354.
355.
356.
357.
    ##############################################################################################
358.    def analyze(self):
359.        '''This function is the main loop run on every event'''
360.
361.        if self.printLevel < 5: print self.name,'.analyze(): running...'
362.        ## runNumber, eventNumber
363.        eventID = self.collections['evinfo']['container'].event_ID()
364.        self.runNumber   = eventID.run_number()
365.        self.eventNumber = eventID.event_number()
366.        if self.printLevel < 3: print 'running on run: %s, ev: %s' % (self.runNumber,self.eventNumber)
367.        #MC weight stored in the AOD/DPD. It is +1/-1 for MC@NLO generator.
368.        self.weightMC = self.collections['gen']['container'][0].weights()[0]
369.
370.
371.        ## I fill the NEvents histo every event #
372.        self.hsvc['/rec/NEvents'].Fill(0.5)
373.        ## I fill the sumWeightsMC histo every event #
374.        self.hsvc['/rec/sumWeightsMC'].Fill(0.5, self.weightMC)
375.
376.        ## Check trigger if required
377.        if self.isTriggerFilter and 'evinfo' in self.collections:
378.            if self.printLevel < 3: print self.name,'.analyze(): check triggers'
379.            if not self.triggerTools.checkTrigger(self.collections['evinfo']['container'],self.trigger):
380.                ## I skip the events not satisfying the chosen triggers
381.                if self.printLevel < 2: print 'Skipping this event because not satisfying the chosen triggers...'
382.                return StatusCode.Success
383.            pass
```

```
384.
385.
386.        ## -- 1 -- Selecting objects
387.        try: candidates = self.objSelection.SelectObjects(self.collections)
388.        except: print; print '** ERROR ** Perhaps you set select=True to a collection with no support
    for object selection in SUSYTools/src/SUSYDefinitions.cxx'; print; print; sys.exit()
389.        self.candidatesList = []
390.        self.candidatesList.append( (candidates,'DEFAULT') )
391.        try: candidates1lep = self.objSelection1lep.SelectObjects(self.collections)
392.        except: print; print '** ERROR ** Perhaps you set select=True to a collection with no support
    for object selection in SUSYTools/src/SUSYDefinitions.cxx'; print; print; sys.exit()
393.        self.candidatesList.append( (candidates1lep, '1lep') )
394.        try: candidates4j0lepCSC = self.objSelection4j0lepCSC.SelectObjects(self.collections)
395.        except: print; print '** ERROR ** Perhaps you set select=True to a collection with no support
    for object selection in SUSYTools/src/SUSYDefinitions.cxx'; print; print; sys.exit()
396.        self.candidatesList.append( (candidates4j0lepCSC, '4j0lepCSC') )
397.        try: candidates3j1lepMediumCutsMTSmaller100GeV =
    self.objSelection3j1lepMediumCutsMTSmaller100GeV.SelectObjects(self.collections)
398.        except: print; print '** ERROR ** Perhaps you set select=True to a collection with no support
    for object selection in SUSYTools/src/SUSYDefinitions.cxx'; print; print; sys.exit()
399.        self.candidatesList.append( (candidates3j1lepMediumCutsMTSmaller100GeV,
    '3j1lepMediumCutsMTSmaller100GeV') )
400.        try: candidates3j1lepMediumCutsMTBigger100GeV =
    self.objSelection3j1lepMediumCutsMTBigger100GeV.SelectObjects(self.collections)
401.        except: print; print '** ERROR ** Perhaps you set select=True to a collection with no support
    for object selection in SUSYTools/src/SUSYDefinitions.cxx'; print; print; sys.exit()
402.        self.candidatesList.append( (candidates3j1lepMediumCutsMTBigger100GeV,
    '3j1lepMediumCutsMTBigger100GeV') )
403.
404.
405.        ## -- 2 -- Removing overlap
406.        self.overlapRemoval.RemoveOverlap(candidates)
407.        self.overlapRemoval.RemoveOverlap(candidates1lep)
408.        self.overlapRemoval.RemoveOverlap(candidates4j0lepCSC)
409.        self.overlapRemoval.RemoveOverlap(candidates3j1lepMediumCutsMTSmaller100GeV)
410.        self.overlapRemoval.RemoveOverlap(candidates3j1lepMediumCutsMTBigger100GeV)
411.
412.        ## -- 3 -- Adding 'met' by hand to 'candidates'. It is not a 'particle' and it's not taken into account
    in SelectObjects() or OverlapRemoval
413.        if 'met' in self.collections:
414.            candidates['met'] = self.collections['met']['container']
415.            candidates1lep['met'] = self.collections['met']['container']
416.            candidates4j0lepCSC['met'] = self.collections['met']['container']
417.            candidates3j1lepMediumCutsMTSmaller100GeV['met'] = self.collections['met']['container']
418.            candidates3j1lepMediumCutsMTBigger100GeV['met'] = self.collections['met']['container']
419.            candidates['truth_met'] = self.collections['truth_met']['container']
420.            candidates1lep['truth_met'] = self.collections['truth_met']['container']
421.            candidates4j0lepCSC['truth_met'] = self.collections['truth_met']['container']
422.            candidates3j1lepMediumCutsMTSmaller100GeV['truth_met'] =
    self.collections['truth_met']['container']
423.            candidates3j1lepMediumCutsMTBigger100GeV['truth_met'] =
    self.collections['truth_met']['container']
424.
425.
```

```
426.        ## -- 4 -- Calculating user-defined variables
427.
    self.calculateUserVariables(candidates,candidates1lep,candidates4j0lepCSC,candidates3j1lepMediumCutsMT
428.
429.
430.        ## -- 5 -- Selecting event
431.        if not
    self.selectEvent(candidates,candidates1lep,candidates4j0lepCSC,candidates3j1lepMediumCutsMTSmaller100
    and self.doSkimmingGlobal: #The selectEvent func is accessed anyway and then left if it's the case
432.            ## I am here if the event did not pass the selection or if I don't want the selection of events
433.            if self.printLevel <2: print 'Skipped this event...'; print; print
434.            return StatusCode.Success
435.
436.
437.        ## -- 6 -- Merging candidates
438.        candidatesMerged = self.MergeCandidates(self.candidatesList)
439.
440.
441.        ## -- 7 -- Filling histograms and ttrees/branches
442.        if True in self.channelsPassed.values() or not self.doSkimmingGlobal:
443.            self.fillHists(self.treeVar, self.extraTreeVar, self.extraBranches, candidatesMerged, ",
    'D3PDTree', oneTree = True)
444.
445.        if self.printLevel < 3: print self.name,'.analyze(): ...done'
446.        return StatusCode.Success
447.
448.
449.
450.
    ##########################################################################################
451.     def calculateUserVariables(self,
    candidates,candidates1lep,candidates4j0lepCSC,candidates3j1lepMediumCutsMTSmaller100GeV,candidates3
452.        'Calculating user-defined variable. Moreover, if a channel makes use of leptons, we build the
    -lepton- container, with leptons (electrons and muons) ordered by descending pT.'
453.
454.        #---------------------------------
455.        def buildingLeptonContainer(candidates):
456.
457.            '''Building the 'leptons' container.'''
458.            ## -----------------------------------
459.            ## Creating and filling a list of 'lepton' dicts. Each dict represents a lepton (el or mu)
460.            leptons = []
461.            if len(candidates['muon']):
462.                [ leptons.append({ 'Pt': m.pt(), 'Eta': m.eta(), 'Phi': m.phi(), 'Type': 'm' }) for m in
    candidates['muon'] ]
463.            if len(candidates['electron']):
464.                [ leptons.append({ 'Pt': e.pt(), 'Eta': e.eta(), 'Phi': e.phi(), 'Type': 'e' }) for e in
    candidates['electron'] ]
465.
466.            if self.printLevel <= 1: print "leptons: ", leptons
467.
468.            ## Ordering the list of leptons according to the pt
469.            import operator #a module of the standard Python library
470.            leptons.sort(key=operator.itemgetter('Pt'))
```

```
471.            #   this sorts the list in place, according to the key 'Pt'
472.            #   Suggestion: use 'newleps = sorted(key...)' if you want to avoid to overwrite the 'leptons' list
473.
474.            ## Reverting the order, according to descending pt
475.            leptons.reverse()
476.            if self.printLevel <= 1: print "leptons: ", leptons
477.
478.            return leptons
479.        #--------------------------------
480.        self.leptons1lep = buildingLeptonContainer(candidates1lep)
481.        self.leptons3j1lepMediumCutsMTSmaller100GeV =
    buildingLeptonContainer(candidates3j1lepMediumCutsMTSmaller100GeV)
482.        self.leptons3j1lepMediumCutsMTBigger100GeV =
    buildingLeptonContainer(candidates3j1lepMediumCutsMTBigger100GeV)
483.
484.        #--------------------------------
485.        def meff(candidates):
486.
487.            meff = 0.
488.            for i,jet in enumerate(candidates['jet']):
489.                meff += candidates['jet'][i].pt()
490.            meff += candidates['met'].et()
491.            return meff
492.        #--------------------------------
493.
494.        #--------------------------------
495.        def sphericity(candidates):
496.
497.            '''Sphericity Calculation formula'''
498.            ## --------------------------------
499.            # To compute the sphericity we use the px,py,pz of the jets and leptons (not MET by default)
500.            # For the cut on leptons pt we use the value of the jetPtCut (20GeV in CSC)
501.
502.            vx = ROOT.std.vector(float)()
503.            vy = ROOT.std.vector(float)()
504.            vz = ROOT.std.vector(float)()
505.
506.            def fill_v(cand):
507.                '''fill vectors with px,py,pz'''
508.                vx.push_back(cand.px())
509.                vy.push_back(cand.py())
510.                vz.push_back(cand.pz())
511.
512.            [ fill_v(candidates[cand][i]) for cand in candidates if (cand = 'electron' or cand = 'muon'
    or cand == 'jet') for i,c in enumerate(candidates[cand]) ]
513.
514.            sphericity = ROOT.SUSYTools.Sphericity(vx,vy,vz,True) #%s)
515.
516.            return sphericity
517.        #--------------------------------
518.        #--------------------------------
519.        def transverseMass(candidates):
520.
521.            '''Calculating the Transverse Mass (1 lepton CSC definition)'''
```

```
522.            ## ------------------------------------------------------------
523.            if not 'buildingLeptonContainer' in vars(): print '\n\n\n***ERROR***\n\nNo func to build
    lepton container.\n\n\n\n';sys.exit()
524.            leptons = buildingLeptonContainer(candidates)
525.            if len(leptons) == 0: return -999.
526.            lep = leptons[0]
527.            deltaPhiLepMet = PyAthena.P4Helpers.deltaPhi( lep['Phi'], candidates['met'].phi() )
528.            transverseMass1Lep = ROOT.SUSYTools.TransverseMass( lep['Pt'], candidates['met'].et() ,
    deltaPhiLepMet )
529.            return transverseMass1Lep
530.        #--------------------------------
531.
532.        self.extraTreeVarChannelsDict = {}
533.
534.
535.        self.extraTreeVarChannelsDict['sphericity'] = {}
536.        self.sphericity1lep = sphericity(candidates1lep)
537.        self.extraTreeVarChannelsDict['sphericity']['1lep'] = self.sphericity1lep
538.        self.sphericity4j0lepCSC = sphericity(candidates4j0lepCSC)
539.        self.extraTreeVarChannelsDict['sphericity']['4j0lepCSC'] = self.sphericity4j0lepCSC
540.        self.sphericity3j1lepMediumCutsMTSmaller100GeV =
    sphericity(candidates3j1lepMediumCutsMTSmaller100GeV)
541.        self.extraTreeVarChannelsDict['sphericity']['3j1lepMediumCutsMTSmaller100GeV'] =
    self.sphericity3j1lepMediumCutsMTSmaller100GeV
542.        self.sphericity3j1lepMediumCutsMTBigger100GeV =
    sphericity(candidates3j1lepMediumCutsMTBigger100GeV)
543.        self.extraTreeVarChannelsDict['sphericity']['3j1lepMediumCutsMTBigger100GeV'] =
    self.sphericity3j1lepMediumCutsMTBigger100GeV
544.        self.sphericity = sphericity(candidates)
545.        self.extraTreeVarChannelsDict['sphericity']['DEFAULT'] = self.sphericity
546.
547.        self.extraTreeVarChannelsDict['meff'] = {}
548.        self.meff1lep = meff(candidates1lep)
549.        self.extraTreeVarChannelsDict['meff']['1lep'] = self.meff1lep
550.        self.meff4j0lepCSC = meff(candidates4j0lepCSC)
551.        self.extraTreeVarChannelsDict['meff']['4j0lepCSC'] = self.meff4j0lepCSC
552.        self.meff3j1lepMediumCutsMTSmaller100GeV =
    meff(candidates3j1lepMediumCutsMTSmaller100GeV)
553.        self.extraTreeVarChannelsDict['meff']['3j1lepMediumCutsMTSmaller100GeV'] =
    self.meff3j1lepMediumCutsMTSmaller100GeV
554.        self.meff3j1lepMediumCutsMTBigger100GeV =
    meff(candidates3j1lepMediumCutsMTBigger100GeV)
555.        self.extraTreeVarChannelsDict['meff']['3j1lepMediumCutsMTBigger100GeV'] =
    self.meff3j1lepMediumCutsMTBigger100GeV
556.        self.meff = meff(candidates)
557.        self.extraTreeVarChannelsDict['meff']['DEFAULT'] = self.meff
558.
559.        self.extraTreeVarChannelsDict['transverseMass'] = {}
560.        self.transverseMass1lep = transverseMass(candidates1lep)
561.        self.extraTreeVarChannelsDict['transverseMass']['1lep'] = self.transverseMass1lep
562.        self.transverseMass4j0lepCSC = transverseMass(candidates4j0lepCSC)
563.        self.extraTreeVarChannelsDict['transverseMass']['4j0lepCSC'] = self.transverseMass4j0lepCSC
564.        self.transverseMass3j1lepMediumCutsMTSmaller100GeV =
    transverseMass(candidates3j1lepMediumCutsMTSmaller100GeV)
```

```
565.        self.extraTreeVarChannelsDict['transverseMass']['3j1lepMediumCutsMTSmaller100GeV'] =
    self.transverseMass3j1lepMediumCutsMTSmaller100GeV
566.        self.transverseMass3j1lepMediumCutsMTBigger100GeV =
    transverseMass(candidates3j1lepMediumCutsMTBigger100GeV)
567.        self.extraTreeVarChannelsDict['transverseMass']['3j1lepMediumCutsMTBigger100GeV'] =
    self.transverseMass3j1lepMediumCutsMTBigger100GeV
568.        self.transverseMass = transverseMass(candidates)
569.        self.extraTreeVarChannelsDict['transverseMass']['DEFAULT'] = self.transverseMass
570.        if self.printLevel <1: print 'self.extraTreeVarChannelsDict = ', self.extraTreeVarChannelsDict
571.
572.
573.
574.
    ##########################################################################################
575.      def selectEvent(self,
    defcandidates,candidates1lep,candidates4j0lepCSC,candidates3j1lepMediumCutsMTSmaller100GeV,candidat
576.        if self.printLevel < 5: print self.name,'.selectEvent(): selecting event...'
577.
578.
579.        ## clearing the dict and the list for this event
580.        self.channelsPassed.clear()
581.        self.channelsList = []
582.        self.channelsPassed1lep.clear()
583.        self.channelsList1lep = []
584.        self.channelsPassed4j0lepCSC.clear()
585.        self.channelsList4j0lepCSC = []
586.        self.channelsPassed3j1lepMediumCutsMTSmaller100GeV.clear()
587.        self.channelsList3j1lepMediumCutsMTSmaller100GeV = []
588.        self.channelsPassed3j1lepMediumCutsMTBigger100GeV.clear()
589.        self.channelsList3j1lepMediumCutsMTBigger100GeV = []
590.
591.
592.        if not self.doSkimmingGlobal:# and not self.doEventSelectionEvenIfNotSkimming:
593.            self.channelsList.append('999')
594.            self.channelsPassed['999'] = True
595.            self.channelsList1lep.append('999')
596.            self.channelsPassed1lep['999'] = True
597.            self.channelsList4j0lepCSC.append('999')
598.            self.channelsPassed4j0lepCSC['999'] = True
599.            self.channelsList3j1lepMediumCutsMTSmaller100GeV.append('999')
600.            self.channelsPassed3j1lepMediumCutsMTSmaller100GeV['999'] = True
601.            self.channelsList3j1lepMediumCutsMTBigger100GeV.append('999')
602.            self.channelsPassed3j1lepMediumCutsMTBigger100GeV['999'] = True
603.            if not self.doEventSelectionEvenIfNotSkimming:
604.                return False
605.            pass
606.
607.        #*********************************************************
608.        #   cuts for channel: 1lep of type 'l'
609.        #*********************************************************
610.
611.        cutPassed = True
612.        if self.printLevel < 5: print self.name,'.selectEvent(): ...evaluating cuts for channel: 1lep'
613.
```

```
614.        #'candidates' used by this channel:
615.        candidates = candidates1lep
616.
617.        #This channel contains leptons. 'leptons' used by this channel:
618.        leptons = self.leptons1lep
619.
620.        #'variables' used by the cuts of this channel:
621.        sphericity = self.sphericity1lep
622.        transverseMass = self.transverseMass1lep
623.        meff = self.meff1lep
624.
625.        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
626.        channelsDict = self.cuts2.channels['1lep']['cuts'][1]
627.        #'value' is an handle to the 'value' key of the dict of this cut. It is used in formulae
628.        value = [20000.0]
629.        if cutPassed:
630.
631.            ## ------------------------------------
632.            ## Cut on number of leptons - Inclusive
633.            ## ------------------------------------
634.            if len(leptons) < len( value ):
635.                if self.printLevel < 3: print self.name,'.selectEvent(): event does not pass number of leptons
    cut'
636.                cutPassed = False
637.                pass
638.            if cutPassed:
639.                ## -----------------
640.                ## Cut on lepton Pts
641.                ## -----------------
642.                for i,lepPtCut in enumerate( value ):
643.                    if leptons[i]['Pt'] < lepPtCut:
644.                        if self.printLevel < 3:
645.                            print self.name, '.selectEvent(): lepton'+str(i)+leptons[i]['Type']+' with '+str(
    leptons[i]['Pt'] )+' pt does not pass '+str(lepPtCut)+' cut'
646.                        cutPassed = False
647.                        pass
648.                pass
649.
650.        ##channel label
651.        if cutPassed:
652.            self.channelsPassed1lep['l'] = True
653.            self.channelsList1lep.append('1lep')
654.            self.channelsPassed['l'] = True
655.            self.channelsList.append('1lep')
656.        #************************************************************
657.
658.
659.
660.        #************************************************************
661.        #   cuts for channel: TestOnlyJets of type 'jjj'
662.        #************************************************************
663.
664.        cutPassed = True
```

```
665.        if self.printLevel < 5: print self.name,'.selectEvent(): ...evaluating cuts for channel: TestOnlyJets'
666.
667.        #'candidates' used by this channel:
668.        candidates = defcandidates
669.
670.
671.        #'variables' used by the cuts of this channel:
672.        sphericity = self.sphericity
673.        transverseMass = self.transverseMass
674.        meff = self.meff
675.        # 'compare' is a handle to the value you put in the definitions of 1 cut for this channel. You can
    use it in your formulae.
676.        # rangeList, rangeMin and rangeMax are handle to the values you put in the definition of cut 1
    for this channel. You can use them in your formulae.
677.        rangeList = [20000.0, 50000.0]
678.        rangeMin = 20000.0
679.        rangeMax = 50000.0
680.
681.        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
682.        channelsDict = self.cuts2.channels['TestOnlyJets']['cuts'][1]
683.        if cutPassed:
684.
685.            ## ------------------------------------------------------------------------
686.            ## electronPtVeto - Reject events with an electron specifying a Pt value or range
687.            ## ------------------------------------------------------------------------
688.            if len(candidates['electron']) > 0:
689.                if 'value' in channelsDict or 'range' in channelsDict:
690.                    for el in candidates['electron']:
691.                        if 'value' in channelsDict and 'range' in channelsDict:
692.                            print '*** WARNING * You have used both value and range key in your channel
    definition!!!'
693.                        ## -------------------------
694.                        ## if a 'value' was provided
695.                        ## -------------------------
696.                        if 'value' in channelsDict:
697.                            if el.pt() <= value:
698.                                cutPassed = False
699.                                break
700.                        ## -------------------------
701.                        ## if a 'range' was provided
702.                        ## -------------------------
703.                        elif 'range' in channelsDict:
704.                            ## if 'compare' was provided
705.                            if 'compare' in channelsDict:
706.                                if rangeMin <= el.pt() and rangeMax >= el.pt():
707.                                    cutPassed = False
708.                                    break
709.                            ## otherwise the default '<>' is taken
710.                            else:
711.                                if rangeMin <= el.pt() and rangeMax >= el.pt():
712.                                    cutPassed = False
713.                                    break
714.                else:
```

```
715.                 cutPassed = False
716.                 if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the
    ElectronPt Veto'
717.
718.         ##channel label
719.         if cutPassed:
720.             self.channelsPassed['jjj'] = True
721.             self.channelsList.append('TestOnlyJets')
722.         #*************************************************************
723.
724.
725.
726.         #*************************************************************
727.         #   cuts for channel: 4j0lepCSC of type 'jjjj'
728.         #*************************************************************
729.
730.         cutPassed = True
731.         if self.printLevel < 5: print self.name,'.selectEvent(): ...evaluating cuts for channel: 4j0lepCSC'
732.
733.         #'candidates' used by this channel:
734.         candidates = candidates4j0lepCSC
735.
736.
737.         #'variables' used by the cuts of this channel:
738.         meff = self.meff4j0lepCSC
739.         sphericity = self.sphericity4j0lepCSC
740.         transverseMass = self.transverseMass4j0lepCSC
741.
742.         #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
743.         channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][1]
744.         #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
745.         value = [100000.0, 50000.0, 50000.0, 50000.0]
746.         if cutPassed:
747.
748.             ## ---------------------------------
749.             ## Cut on number of jets - Inclusive
750.             ## ---------------------------------
751.             if len(candidates['jet']) < len( value ):
752.                 if self.printLevel < 3: print self.name,'.selectEvent(): event does not pass number of jets cut'
753.                 cutPassed = False
754.                 pass
755.             if cutPassed:
756.                 ## -------------
757.                 ## Cut on jet pt
758.                 ## -------------
759.                 for iJet,jetPtCut in enumerate( value ):
760.                     if candidates['jet'][iJet].pt() < jetPtCut:
761.                         if self.printLevel < 3:
762.                             print self.name, '.selectEvent(): jet '+str(iJet)+' with '+str( candidates['jet'][iJet].pt()
    )+' pt does not pass '+str(jetPtCut)+' cut'
763.                         cutPassed = False
764.                         pass
765.                 pass
```

```
766.
767.        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
     dict keys added by the user
768.        channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][2]
769.        #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
770.        value = 100000.0
771.        if cutPassed:
772.
773.            ## ----------------
774.            ## Cut on missingEt
775.            ## ----------------
776.            if candidates['met'].et() <= value:
777.                cutPassed = False
778.                if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the missingEt
     Cut of ', value
779.
780.        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
     dict keys added by the user
781.        channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][3]
782.        #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
783.        value = 0.2
784.        if cutPassed:
785.
786.            ## --------------------------------
787.            ## Cut on missingEt > Fraction*Meff
788.            ## --------------------------------
789.            fracMeff = value
790.            if candidates['met'].et() <= fracMeff*meff:
791.                cutPassed = False
792.                if self.printLevel < 3: print self.name, ".selectEvent(): the event with MET=",
     candidates['met'].et()," and meff=", meff, " did not pass the MetFracMeffCut Cut of: ", value
793.                pass
794.
795.        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
     dict keys added by the user
796.        channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][4]
797.        #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
798.        value = 0.2
799.        if cutPassed:
800.
801.            ## -----------------
802.            ## Cut on Sphericity
803.            ## -----------------
804.            # in order to use this cut you must include the "sphericity" formula in your steering file
805.            ##
806.            ## safety check
807.            ## ------------
808.            try:
809.                if sphericity: pass
810.            except:
811.                print "\n * ERROR * \n\n'sphericity' variable seems not to be defined!!"
812.                print "\nIn order to cut on Sphericity you have to calculate it."
813.                print "In your steering-file, please add a branch filled with Sphericity, for example with:"
814.                print "\nuserD3PDBranchesToFill = {'sphericityCSC' : {'label': 'sphCSC', 'type': 'float',
```

```
    'formula': 'sphericity'},}\n\n"
815.             sys.exit()
816.
817.         ## ---------------------
818.         ## Actual Sphericity Cut
819.         ## ---------------------
820.         if sphericity <= value:
821.             cutPassed = False
822.             if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the Sphericity
    Cut of', value
823.
824.     #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
825.     channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][5]
826.     #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
827.     value = [0.20000000000000001, 0.20000000000000001, 0.20000000000000001]
828.     if cutPassed:
829.
830.         ## ------------------------
831.         ## Cut on deltaPhi(jet,met)
832.         ## ------------------------
833.         try:
834.             for iJet,jetDPhiCut in enumerate( value ): #Note! 'jetsDPhiCuts' is a list,defining the cut jet
    by jet
835.                 deltaPhi = PyAthena.P4Helpers.deltaPhi( candidates['jet'][iJet], candidates['met'].phi() )
836.                 if math.fabs(deltaPhi) < jetDPhiCut:
837.                     if self.printLevel < 3:
838.                         print self.name, '.selectEvent(): jet '+str(iJet)+' with deltaPhi = '+str(deltaPhi)+' does
    not pass '+str(jetDPhiCut)+' cut'
839.                         pass
840.                     cutPassed = False
841.                     pass
842.                 pass
843.         except IndexError:
844.             print"\n\n * ERROR ***\nCheck your steering file.\n[Hint: check if lenght of
    jet_metDPhiCut 'value' list is <= jetPtCuts one]\n\n\n"
845.             sys.exit()
846.
847.     #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
848.     channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][6]
849.     if cutPassed:
850.
851.         ## ------------------------------------------------------------------------
852.         ## electronPtVeto - Reject events with an electron specifying a Pt value or range
853.         ## ------------------------------------------------------------------------
854.         if len(candidates['electron']) > 0:
855.             if 'value' in channelsDict or 'range' in channelsDict:
856.                 for el in candidates['electron']:
857.                     if 'value' in channelsDict and 'range' in channelsDict:
858.                         print '*** WARNING * You have used both value and range key in your channel
    definition!!!'
859.                     ## -------------------------
860.                     ## if a 'value' was provided
```

```
                    ## -------------------------
                    if 'value' in channelsDict:
                        if el.pt() <= value:
                            cutPassed = False
                            break
                    ## -------------------------
                    ## if a 'range' was provided
                    ## -------------------------
                    elif 'range' in channelsDict:
                        ## if 'compare' was provided
                        if 'compare' in channelsDict:
                            if rangeMin <= el.pt() and rangeMax >= el.pt():
                                cutPassed = False
                                break
                        ## otherwise the default '<>' is taken
                        else:
                            if rangeMin <= el.pt() and rangeMax >= el.pt():
                                cutPassed = False
                                break
                else:
                    cutPassed = False
                    if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the ElectronPt Veto'

        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
        channelsDict = self.cuts2.channels['4j0lepCSC']['cuts'][7]
        if cutPassed:

            ## ------------------------------------------------------------------
            ## muonPtVeto - Reject events with a muon specifying a Pt value or range
            ## ------------------------------------------------------------------
            if len(candidates['muon']) > 0:
                if 'value' in channelsDict or 'range' in channelsDict:
                    for el in candidates['muon']:
                        if 'value' in channelsDict and 'range' in channelsDict:
                            print '*** WARNING * You have used both value and range key in your channel definition!!!'
                        ## -------------------------
                        ## if a 'value' was provided
                        ## -------------------------
                        if 'value' in channelsDict:
                            if el.pt() <= value:
                                cutPassed = False
                                break #if we find at least one, we exit the loop
                        ## -------------------------
                        ## if a 'range' was provided
                        ## -------------------------
                        elif 'range' in channelsDict:
                            ## if 'compare' was provided
                            if 'compare' in channelsDict:
                                if rangeMin <= el.pt() and rangeMax >= el.pt():
                                    cutPassed = False
                                    break
```

```
912.                     ## otherwise the default '<>' is taken
913.                     else:
914.                         if rangeMin <= el.pt() and rangeMax >= el.pt():
915.                             cutPassed = False
916.                             break
917.             else:
918.                 cutPassed = False
919.                 if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the MuonPt
    Veto'
920.
921.         if cutPassed:
922.             ## MeffCut: cutting on variable meff taking self.meff
923.             if meff <= 800000.0:
924.                 cutPassed = False
925.                 if self.printLevel < 3: print self.name, '.selectEvent(): event did not pass the meff Cut'
926.
927.         ##channel label
928.         if cutPassed:
929.             self.channelsPassed4j0lepCSC['jjjj'] = True
930.             self.channelsList4j0lepCSC.append('4j0lepCSC')
931.             self.channelsPassed['jjjj'] = True
932.             self.channelsList.append('4j0lepCSC')
933.         #************************************************************
934.
935.
936.
937.         #************************************************************
938.         #   cuts for channel: 3j1lepMediumCutsMTSmaller100GeV of type 'ljjj'
939.         #************************************************************
940.
941.         cutPassed = True
942.         if self.printLevel < 5: print self.name,'.selectEvent(): ...evaluating cuts for channel:
    3j1lepMediumCutsMTSmaller100GeV'
943.
944.         #'candidates' used by this channel:
945.         candidates = candidates3j1lepMediumCutsMTSmaller100GeV
946.
947.         #This channel contains leptons. 'leptons' used by this channel:
948.         leptons = self.leptons3j1lepMediumCutsMTSmaller100GeV
949.
950.         #'variables' used by the cuts of this channel:
951.         sphericity = self.sphericity3j1lepMediumCutsMTSmaller100GeV
952.         transverseMass = self.transverseMass3j1lepMediumCutsMTSmaller100GeV
953.         meff = self.meff3j1lepMediumCutsMTSmaller100GeV
954.
955.         #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
956.         channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][1]
957.         #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
958.         value = [20000.0]
959.         if cutPassed:
960.
961.             ## ----------------------------------
962.             ## Cut on number of leptons - Exclusive
```

```
963.          ## ------------------------------------
964.          if not len(leptons) == len( value ):
965.             cutPassed = False
966.             if self.printLevel < 3: print self.name,'.selectEvent(): event does not pass number of leptons
      cut'
967.          if cutPassed:
968.             ## -----------------
969.             ## Cut on lepton Pts
970.             ## -----------------
971.             for i,lepPtCut in enumerate( value ):
972.                if leptons[i]['Pt'] < lepPtCut:
973.                   if self.printLevel < 3:
974.                      print self.name, '.selectEvent(): lepton'+str(i)+leptons[i]['Type']+' with '+str(
      leptons[i]['Pt'] )+' pt does not pass '+str(lepPtCut)+' cut'
975.                   cutPassed = False
976.                   pass
977.                pass
978.
979.       # rangeList, rangeMin and rangeMax are handle to the values you put in the definition of cut 2
      for this channel. You can use them in your formulae.
980.       rangeList = [10000.0, 20000.0]
981.       rangeMin = 10000.0
982.       rangeMax = 20000.0
983.
984.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
      dict keys added by the user
985.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][2]
986.       if cutPassed:
987.
988.          ## ------------------------------------------------------------------------
989.          ## electronPtVeto - Reject events with an electron specifying a Pt value or range
990.          ## ------------------------------------------------------------------------
991.          if len(candidates['electron']) > 0:
992.             if 'value' in channelsDict or 'range' in channelsDict:
993.                for el in candidates['electron']:
994.                   if 'value' in channelsDict and 'range' in channelsDict:
995.                      print '*** WARNING * You have used both value and range key in your channel
      definition!!!'
996.                   ## -------------------------
997.                   ## if a 'value' was provided
998.                   ## -------------------------
999.                   if 'value' in channelsDict:
1000.                     if el.pt() <= value:
1001.                        cutPassed = False
1002.                        break
1003.                  ## -------------------------
1004.                  ## if a 'range' was provided
1005.                  ## -------------------------
1006.                  elif 'range' in channelsDict:
1007.                     ## if 'compare' was provided
1008.                     if 'compare' in channelsDict:
1009.                        if rangeMin <= el.pt() and rangeMax >= el.pt():
1010.                           cutPassed = False
1011.                           break
```

```
                    ## otherwise the default '<>' is taken
                    else:
                        if rangeMin <= el.pt() and rangeMax >= el.pt():
                            cutPassed = False
                            break
            else:
                cutPassed = False
                if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the ElectronPt Veto'
    # rangeList, rangeMin and rangeMax are handle to the values you put in the definition of cut 3 for this channel. You can use them in your formulae.
    rangeList = [10000.0, 20000.0]
    rangeMin = 10000.0
    rangeMax = 20000.0

    #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
    channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][3]
    if cutPassed:

        ## ------------------------------------------------------------------
        ## muonPtVeto - Reject events with a muon specifying a Pt value or range
        ## ------------------------------------------------------------------
        if len(candidates['muon']) > 0:
            if 'value' in channelsDict or 'range' in channelsDict:
                for el in candidates['muon']:
                    if 'value' in channelsDict and 'range' in channelsDict:
                        print '*** WARNING * You have used both value and range key in your channel definition!!!'
                    ## -------------------------
                    ## if a 'value' was provided
                    ## -------------------------
                    if 'value' in channelsDict:
                        if el.pt() <= value:
                            cutPassed = False
                            break #if we find at least one, we exit the loop
                    ## -------------------------
                    ## if a 'range' was provided
                    ## -------------------------
                    elif 'range' in channelsDict:
                        ## if 'compare' was provided
                        if 'compare' in channelsDict:
                            if rangeMin <= el.pt() and rangeMax >= el.pt():
                                cutPassed = False
                                break
                        ## otherwise the default '<>' is taken
                        else:
                            if rangeMin <= el.pt() and rangeMax >= el.pt():
                                cutPassed = False
                                break
            else:
                cutPassed = False
                if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the MuonPt Veto'
```

```
1061.
1062.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
1063.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][4]
1064.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1065.       value = [100000.0, 40000.0, 40000.0]
1066.       if cutPassed:
1067.
1068.          ## ---------------------------------
1069.          ## Cut on number of jets - Inclusive
1070.          ## ---------------------------------
1071.          if len(candidates['jet']) < len( value ):
1072.             if self.printLevel < 3: print self.name,'.selectEvent(): event does not pass number of jets cut'
1073.             cutPassed = False
1074.             pass
1075.          if cutPassed:
1076.             ## -------------
1077.             ## Cut on jet pt
1078.             ## -------------
1079.             for iJet,jetPtCut in enumerate( value ):
1080.                if candidates['jet'][iJet].pt() < jetPtCut:
1081.                   if self.printLevel < 3:
1082.                      print self.name, '.selectEvent(): jet '+str(iJet)+' with '+str( candidates['jet'][iJet].pt() )+' pt does not pass '+str(jetPtCut)+' cut'
1083.                   cutPassed = False
1084.                   pass
1085.             pass
1086.
1087.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
1088.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][5]
1089.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1090.       value = 80000.0
1091.       if cutPassed:
1092.
1093.          ## ----------------
1094.          ## Cut on missingEt
1095.          ## ----------------
1096.          if candidates['met'].et() <= value:
1097.             cutPassed = False
1098.             if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the missingEt Cut of ', value
1099.
1100.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
1101.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][6]
1102.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1103.       value = 0.25
1104.       if cutPassed:
1105.
1106.          ## ---------------------------------
1107.          ## Cut on missingEt > Fraction*Meff
1108.          ## ---------------------------------
1109.          fracMeff = value
```

```
1110.           if candidates['met'].et() <= fracMeff*meff:
1111.               cutPassed = False
1112.               if self.printLevel < 3: print self.name, ".selectEvent(): the event with MET=",
    candidates['met'].et()," and meff=", meff, " did not pass the MetFracMeffCut Cut of: ", value
1113.               pass
1114.
1115.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
1116.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][7]
1117.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1118.       value = 0.2
1119.       if cutPassed:
1120.
1121.           ## -----------------
1122.           ## Cut on Sphericity
1123.           ## -----------------
1124.           # in order to use this cut you must include the "sphericity" formula in your steering file
1125.           ##
1126.           ## safety check
1127.           ## ------------
1128.           try:
1129.               if sphericity: pass
1130.           except:
1131.               print "\n * ERROR * \n\n'sphericity' variable seems not to be defined!!"
1132.               print "\nIn order to cut on Sphericity you have to calculate it."
1133.               print "In your steering-file, please add a branch filled with Sphericity, for example with:"
1134.               print "\nuserD3PDBranchesToFill = {'sphericityCSC' : {'label': 'sphCSC', 'type': 'float',
    'formula': 'sphericity'},}\n\n"
1135.               sys.exit()
1136.
1137.           ## ---------------------
1138.           ## Actual Sphericity Cut
1139.           ## ---------------------
1140.           if sphericity <= value:
1141.               cutPassed = False
1142.               if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the Sphericity
    Cut of', value
1143.
1144.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
1145.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][8]
1146.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1147.       value = [0.20000000000000001, 0.20000000000000001, 0.20000000000000001]
1148.       if cutPassed:
1149.
1150.           ## ------------------------
1151.           ## Cut on deltaPhi(jet,met)
1152.           ## ------------------------
1153.           try:
1154.               for iJet,jetDPhiCut in enumerate( value ): #Note! 'jetsDPhiCuts' is a list,defining the cut jet
    by jet
1155.                   deltaPhi = PyAthena.P4Helpers.deltaPhi( candidates['jet'][iJet], candidates['met'].phi() )
1156.                   if math.fabs(deltaPhi) < jetDPhiCut:
1157.                       if self.printLevel < 3:
```

```
1158.                         print self.name, '.selectEvent(): jet '+str(iJet)+' with deltaPhi = '+str(deltaPhi)+' does not pass '+str(jetDPhiCut)+' cut'
1159.                         pass
1160.                     cutPassed = False
1161.                     pass
1162.                 pass
1163.         except IndexError:
1164.             print"\n\n * ERROR ***\nCheck your steering file.\n[Hint: check if lenght of jet_metDPhiCut 'value' list is <= jetPtCuts one]\n\n\n"
1165.             sys.exit()
1166.     # 'compare' is a handle to the value you put in the definitions of 9 cut for this channel. You can use it in your formulae.
1167.
1168.     #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
1169.     channelsDict = self.cuts2.channels['3j1lepMediumCutsMTSmaller100GeV']['cuts'][9]
1170.     #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1171.     value = 100000.0
1172.     if cutPassed:
1173.
1174.         ## ---------------------------
1175.         # Cut on Transverse Mass (CSC)
1176.         ## ---------------------------
1177.         # Please notice: in order to use this cut you must include the "transverseMass" formula in your steering file
1178.         #
1179.         ## safety check
1180.         ## ------------
1181.         #try:
1182.         #    if transverseMass: pass
1183.         #except:
1184.         #    print "\n * ERROR * \n\n'transverseMass' variable seems not to be defined!!"
1185.         #    print "\nIn order to cut on Transverse Mass you have to calculate it."
1186.         #    print "In your steering-file, please add a branch filled with Sphericity, for example with:"
1187.         #    print "\nuserD3PDBranchesToFill = {'TM_CSC' : {'label': 'TM_CSC', 'type': 'float', 'formula': 'transverseMass'},}\n\n"
1188.         #    sys.exit()
1189.
1190.         ## ------------------------------
1191.         ## Actual CSC Transverse Mass Cut
1192.         ## ------------------------------
1193.         #
1194.         # check if we have only one lepton
1195.         # ------------------------------
1196.         if not len(leptons) == 1:
1197.             print "\n * ERROR ***\n\nMore than one lepton for a channel where transverseMassCalculation is used!! Check your steering file!"
1198.             sys.exit()
1199.
1200.         lep = leptons[0]
1201.         deltaPhiLepMet = PyAthena.P4Helpers.deltaPhi( lep['Phi'], candidates['met'].phi() )
1202.         transverseMass = ROOT.SUSYTools.TransverseMass( lep['Pt'], candidates['met'].et() , deltaPhiLepMet )
1203.
```

```
1204.           # cut on Transverse Mass
1205.           # ----------------------
1206.           #if 'compare' in channelsDict:
1207.           if transverseMass > value:
1208.              cutPassed = False
1209.              if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the Transverse
    Mass Cut of', value
1210.           #else:
1211.           #    if transverseMass <= value:
1212.           #        cutPassed = False
1213.           #        if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the
    Transverse Mass Cut of', value
1214.
1215.        ##channel label
1216.        if cutPassed:
1217.           self.channelsPassed3j1lepMediumCutsMTSmaller100GeV['ljjj'] = True
1218.
    self.channelsList3j1lepMediumCutsMTSmaller100GeV.append('3j1lepMediumCutsMTSmaller100GeV')
1219.           self.channelsPassed['ljjj'] = True
1220.           self.channelsList.append('3j1lepMediumCutsMTSmaller100GeV')
1221.        #************************************************************
1222.
1223.
1224.
1225.        #************************************************************
1226.        #   cuts for channel: 3j1lepMediumCutsMTBigger100GeV of type 'ljjj'
1227.        #************************************************************
1228.
1229.        cutPassed = True
1230.        if self.printLevel < 5: print self.name,'.selectEvent(): ...evaluating cuts for channel:
    3j1lepMediumCutsMTBigger100GeV'
1231.
1232.        #'candidates' used by this channel:
1233.        candidates = candidates3j1lepMediumCutsMTBigger100GeV
1234.
1235.        #This channel contains leptons. 'leptons' used by this channel:
1236.        leptons = self.leptons3j1lepMediumCutsMTBigger100GeV
1237.
1238.        #'variables' used by the cuts of this channel:
1239.        sphericity = self.sphericity3j1lepMediumCutsMTBigger100GeV
1240.        transverseMass = self.transverseMass3j1lepMediumCutsMTBigger100GeV
1241.        meff = self.meff3j1lepMediumCutsMTBigger100GeV
1242.
1243.        #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
1244.        channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][1]
1245.        #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1246.        value = [20000.0]
1247.        if cutPassed:
1248.
1249.           ## ------------------------------------
1250.           ## Cut on number of leptons - Exclusive
1251.           ## ------------------------------------
1252.           if not len(leptons) == len( value ):
```

```
1253.             cutPassed = False
1254.             if self.printLevel < 3: print self.name,'.selectEvent(): event does not pass number of leptons cut'
1255.         if cutPassed:
1256.             ## -----------------
1257.             ## Cut on lepton Pts
1258.             ## -----------------
1259.             for i,lepPtCut in enumerate( value ):
1260.                 if leptons[i]['Pt'] < lepPtCut:
1261.                     if self.printLevel < 3:
1262.                         print self.name, '.selectEvent(): lepton'+str(i)+leptons[i]['Type']+' with '+str( leptons[i]['Pt'] )+' pt does not pass '+str(lepPtCut)+' cut'
1263.                     cutPassed = False
1264.                     pass
1265.                 pass
1266.
1267.     # rangeList, rangeMin and rangeMax are handle to the values you put in the definition of cut 2 for this channel. You can use them in your formulae.
1268.     rangeList = [10000.0, 20000.0]
1269.     rangeMin = 10000.0
1270.     rangeMax = 20000.0
1271.
1272.     #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
1273.     channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][2]
1274.     if cutPassed:
1275.
1276.         ## ------------------------------------------------------------------------
1277.         ## electronPtVeto - Reject events with an electron specifying a Pt value or range
1278.         ## ------------------------------------------------------------------------
1279.         if len(candidates['electron']) > 0:
1280.             if 'value' in channelsDict or 'range' in channelsDict:
1281.                 for el in candidates['electron']:
1282.                     if 'value' in channelsDict and 'range' in channelsDict:
1283.                         print '*** WARNING * You have used both value and range key in your channel definition!!!'
1284.                     ## -------------------------
1285.                     ## if a 'value' was provided
1286.                     ## -------------------------
1287.                     if 'value' in channelsDict:
1288.                         if el.pt() <= value:
1289.                             cutPassed = False
1290.                             break
1291.                     ## -------------------------
1292.                     ## if a 'range' was provided
1293.                     ## -------------------------
1294.                     elif 'range' in channelsDict:
1295.                         ## if 'compare' was provided
1296.                         if 'compare' in channelsDict:
1297.                             if rangeMin <= el.pt() and rangeMax >= el.pt():
1298.                                 cutPassed = False
1299.                                 break
1300.                         ## otherwise the default '<>' is taken
1301.                         else:
```

```
1302.                         if rangeMin <= el.pt() and rangeMax >= el.pt():
1303.                             cutPassed = False
1304.                             break
1305.             else:
1306.                 cutPassed = False
1307.                 if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the ElectronPt Veto'
1308.         # rangeList, rangeMin and rangeMax are handle to the values you put in the definition of cut 3 for this channel. You can use them in your formulae.
1309.         rangeList = [10000.0, 20000.0]
1310.         rangeMin = 10000.0
1311.         rangeMax = 20000.0
1312.
1313.         #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
1314.         channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][3]
1315.         if cutPassed:
1316.
1317.             ## ------------------------------------------------------------------
1318.             ## muonPtVeto - Reject events with a muon specifying a Pt value or range
1319.             ## ------------------------------------------------------------------
1320.             if len(candidates['muon']) > 0:
1321.                 if 'value' in channelsDict or 'range' in channelsDict:
1322.                     for el in candidates['muon']:
1323.                         if 'value' in channelsDict and 'range' in channelsDict:
1324.                             print '*** WARNING * You have used both value and range key in your channel definition!!!'
1325.                         ## -------------------------
1326.                         ## if a 'value' was provided
1327.                         ## -------------------------
1328.                         if 'value' in channelsDict:
1329.                             if el.pt() <= value:
1330.                                 cutPassed = False
1331.                                 break #if we find at least one, we exit the loop
1332.                         ## -------------------------
1333.                         ## if a 'range' was provided
1334.                         ## -------------------------
1335.                         elif 'range' in channelsDict:
1336.                             ## if 'compare' was provided
1337.                             if 'compare' in channelsDict:
1338.                                 if rangeMin <= el.pt() and rangeMax >= el.pt():
1339.                                     cutPassed = False
1340.                                     break
1341.                             ## otherwise the default '<>' is taken
1342.                             else:
1343.                                 if rangeMin <= el.pt() and rangeMax >= el.pt():
1344.                                     cutPassed = False
1345.                                     break
1346.                 else:
1347.                     cutPassed = False
1348.                     if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the MuonPt Veto'
1349.
1350.         #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
```

```
    dict keys added by the user
1351.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][4]
1352.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1353.       value = [100000.0, 40000.0, 40000.0]
1354.       if cutPassed:
1355.
1356.          ## ---------------------------------
1357.          ## Cut on number of jets - Inclusive
1358.          ## ---------------------------------
1359.          if len(candidates['jet']) < len( value ):
1360.             if self.printLevel < 3: print self.name,'.selectEvent(): event does not pass number of jets cut'
1361.             cutPassed = False
1362.             pass
1363.          if cutPassed:
1364.             ## -------------
1365.             ## Cut on jet pt
1366.             ## -------------
1367.             for iJet,jetPtCut in enumerate( value ):
1368.                if candidates['jet'][iJet].pt() < jetPtCut:
1369.                   if self.printLevel < 3:
1370.                      print self.name, '.selectEvent(): jet '+str(iJet)+' with '+str( candidates['jet'][iJet].pt()
    )+' pt does not pass '+str(jetPtCut)+' cut'
1371.                   cutPassed = False
1372.                   pass
1373.             pass
1374.
1375.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
1376.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][5]
1377.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1378.       value = 80000.0
1379.       if cutPassed:
1380.
1381.          ## ----------------
1382.          ## Cut on missingEt
1383.          ## ----------------
1384.          if candidates['met'].et() <= value:
1385.             cutPassed = False
1386.             if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the missingEt
    Cut of ', value
1387.
1388.       #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
    dict keys added by the user
1389.       channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][6]
1390.       #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1391.       value = 0.25
1392.       if cutPassed:
1393.
1394.          ## -------------------------------
1395.          ## Cut on missingEt > Fraction*Meff
1396.          ## -------------------------------
1397.          fracMeff = value
1398.          if candidates['met'].et() <= fracMeff*meff:
1399.             cutPassed = False
```

```
1400.                 if self.printLevel < 3: print self.name, ".selectEvent(): the event with MET=",
      candidates['met'].et()," and meff=", meff, " did not pass the MetFracMeffCut Cut of: ", value
1401.                 pass
1402.
1403.         #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
      dict keys added by the user
1404.         channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][7]
1405.         #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1406.         value = 0.2
1407.         if cutPassed:
1408.
1409.             ## -----------------
1410.             ## Cut on Sphericity
1411.             ## -----------------
1412.             # in order to use this cut you must include the "sphericity" formula in your steering file
1413.             ##
1414.             ## safety check
1415.             ## ------------
1416.             try:
1417.                 if sphericity: pass
1418.             except:
1419.                 print "\n * ERROR * \n\n'sphericity' variable seems not to be defined!!"
1420.                 print "\nIn order to cut on Sphericity you have to calculate it."
1421.                 print "In your steering-file, please add a branch filled with Sphericity, for example with:"
1422.                 print "\nuserD3PDBranchesToFill = {'sphericityCSC' : {'label': 'sphCSC', 'type': 'float',
      'formula': 'sphericity'},}\n\n"
1423.                 sys.exit()
1424.
1425.             ## ---------------------
1426.             ## Actual Sphericity Cut
1427.             ## ---------------------
1428.             if sphericity <= value:
1429.                 cutPassed = False
1430.                 if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the Sphericity
      Cut of', value
1431.
1432.         #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access
      dict keys added by the user
1433.         channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][8]
1434.         #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
1435.         value = [0.20000000000000001, 0.20000000000000001, 0.20000000000000001]
1436.         if cutPassed:
1437.
1438.             ## ------------------------
1439.             ## Cut on deltaPhi(jet,met)
1440.             ## ------------------------
1441.             try:
1442.                 for iJet,jetDPhiCut in enumerate( value ): #Note! 'jetsDPhiCuts' is a list,defining the cut jet
      by jet
1443.                     deltaPhi = PyAthena.P4Helpers.deltaPhi( candidates['jet'][iJet], candidates['met'].phi() )
1444.                     if math.fabs(deltaPhi) < jetDPhiCut:
1445.                         if self.printLevel < 3:
1446.                             print self.name, '.selectEvent(): jet '+str(iJet)+' with deltaPhi = '+str(deltaPhi)+' does
      not pass '+str(jetDPhiCut)+' cut'
```

```
                    pass
                cutPassed = False
                pass
            pass
        except IndexError:
            print"\n\n * ERROR ***\nCheck your steering file.\n[Hint: check if lenght of jet_metDPhiCut 'value' list is <= jetPtCuts one]\n\n\n"
            sys.exit()
    # 'compare' is a handle to the value you put in the definitions of 9 cut for this channel. You can use it in your formulae.

    #'channelsDict' is an handle to the dict relative to this cut, it can be used in formulae to access dict keys added by the user
    channelsDict = self.cuts2.channels['3j1lepMediumCutsMTBigger100GeV']['cuts'][9]
    #'value' is an handle to the 'value' key of the dict of this cut. It's used in formulae
    value = 100000.0
    if cutPassed:

        ## --------------------------
        # Cut on Transverse Mass (CSC)
        ## --------------------------
        # Please notice: in order to use this cut you must include the "transverseMass" formula in your steering file
        #
        ## safety check
        ## ------------
        #try:
        #    if transverseMass: pass
        #except:
        #    print "\n * ERROR * \n\n'transverseMass' variable seems not to be defined!!"
        #    print "\nIn order to cut on Transverse Mass you have to calculate it."
        #    print "In your steering-file, please add a branch filled with Sphericity, for example with:"
        #    print "\nuserD3PDBranchesToFill = {'TM_CSC' : {'label': 'TM_CSC', 'type': 'float', 'formula': 'transverseMass'},}\n\n"
        #    sys.exit()

        ## ------------------------------
        ## Actual CSC Transverse Mass Cut
        ## ------------------------------
        #
        # check if we have only one lepton
        # --------------------------------
        if not len(leptons) == 1:
            print "\n * ERROR ***\n\nMore than one lepton for a channel where transverseMassCalculation is used!! Check your steering file!"
            sys.exit()

        lep = leptons[0]
        deltaPhiLepMet = PyAthena.P4Helpers.deltaPhi( lep['Phi'], candidates['met'].phi() )
        transverseMass = ROOT.SUSYTools.TransverseMass( lep['Pt'], candidates['met'].et() , deltaPhiLepMet )

        # cut on Transverse Mass
        # ----------------------
```

```
1494.           #if 'compare' in channelsDict:
1495.           if transverseMass <= value:
1496.               cutPassed = False
1497.               if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the Transverse Mass Cut of', value
1498.           #else:
1499.           #    if transverseMass <= value:
1500.           #        cutPassed = False
1501.           #        if self.printLevel < 3: print self.name, '.selectEvent(): the event did not pass the Transverse Mass Cut of', value
1502.
1503.       ##channel label
1504.       if cutPassed:
1505.           self.channelsPassed3j1lepMediumCutsMTBigger100GeV['ljjj'] = True
1506.
self.channelsList3j1lepMediumCutsMTBigger100GeV.append('3j1lepMediumCutsMTBigger100GeV')
1507.           self.channelsPassed['ljjj'] = True
1508.           self.channelsList.append('3j1lepMediumCutsMTBigger100GeV')
1509.       #*********************************************************
1510.
1511.
1512.
1513.
1514.
1515.       #If doSkimmingGlobal==False and an event passed at least one channel, the default value '999' is removed.
1516.       if '999' in self.channelsList and len(self.channelsList)>1: self.channelsList.remove('999')
1517.
1518.       if self.printLevel <=1: print self.name,'.selectEvent(): channelsList, channelsPassed: ', self.channelsList, self.channelsPassed
1519.       if self.printLevel <=1: print self.name,'.selectEvent(): channelsList1lep, channelsPassed1lep: ', self.channelsList1lep, self.channelsPassed1lep
1520.       if self.printLevel <=1: print self.name,'.selectEvent(): channelsList4j0lepCSC, channelsPassed4j0lepCSC: ', self.channelsList4j0lepCSC, self.channelsPassed4j0lepCSC
1521.       if self.printLevel <=1: print self.name,'.selectEvent():
channelsList3j1lepMediumCutsMTSmaller100GeV,
channelsPassed3j1lepMediumCutsMTSmaller100GeV: ',
self.channelsList3j1lepMediumCutsMTSmaller100GeV,
self.channelsPassed3j1lepMediumCutsMTSmaller100GeV
1522.       if self.printLevel <=1: print self.name,'.selectEvent():
channelsList3j1lepMediumCutsMTBigger100GeV,
channelsPassed3j1lepMediumCutsMTBigger100GeV: ',
self.channelsList3j1lepMediumCutsMTBigger100GeV,
self.channelsPassed3j1lepMediumCutsMTBigger100GeV
1523.       if self.printLevel < 4: print self.name,'.selectEvent(): ...done'
1524.       if True in self.channelsPassed.values(): return True
1525.       elif True in self.channelsPassed1lep.values(): return True
1526.       elif True in self.channelsPassed4j0lepCSC.values(): return True
1527.       elif True in self.channelsPassed3j1lepMediumCutsMTSmaller100GeV.values(): return True
1528.       elif True in self.channelsPassed3j1lepMediumCutsMTBigger100GeV.values(): return True
1529.       else: return False
1530.
1531.
1532.
```

```
    ##################################################################################################
1533.   def MergeCandidates(self, candidates):
1534.     '''Merging candidates from different channels with different selection/overlap cuts. Every particle
    is flagged with channels whose cuts satisfies.'''
1535.     partDict = {}
1536.     for tupla in candidates:
1537.       candidate = tupla[0]
1538.       chan = tupla[1]
1539.       if self.printLevel<2: print 'Merging Candidates():Merging - candidate: ', candidate
1540.       if self.printLevel<2: print'channel: ', chan
1541.       #looping over particles
1542.       for cand in candidate:
1543.         if self.printLevel<2: print 'cand: ', cand
1544.         if not cand in partDict:
1545.           partDict[cand] = {} #create a new key for the dictionary
1546.         if cand = 'met' or cand = 'truth_met':
1547.           if self.printLevel<1: print 'candidate[cand], partDict[cand]: ', candidate[cand],
    partDict[cand]
1548.           # Check if a particle had already been written down
1549.           #with 'met', added 'by hand', this check works because the objects pointing to the same
    'met'
1550.           #  object have the very same id(). For other particle I have to do a trick.
1551.           if candidate[cand] in partDict[cand]:
1552.             #if Yes, update only the Channels list
1553.             if self.printLevel<1: print 'obj already present in dict as key, I update channels'
1554.             partDict[cand][candidate[cand]].append( chan )
1555.           else:
1556.             #If No, add a new entry for that particle
1557.             if self.printLevel<1: print 'obj not present in dict as key, I add a new entry'
1558.             partDict[cand][ candidate[cand] ] = []
1559.             partDict[cand][ candidate[cand] ].append( chan )
1560.         else:
1561.           for obj in candidate[cand]:
1562.             if self.printLevel<1: print 'obj, candidate[cand], partDict[cand]: ', obj, candidate[cand],
    partDict[cand]
1563.             # Check if a particle had already been written down
1564.             # Those candidates, even if they point to the same object, they have different id()
1565.             #  so a mere comparison between them does not work. I found a trick: I compare repr()
    of the objs.
1566.             listRepr = [repr(xx) for xx in partDict[cand].keys()]
1567.             if self.printLevel < 1: print 'listRepr: ', listRepr
1568.             if repr(obj) in listRepr:
1569.               for stuff in partDict[cand]:
1570.                 if self.printLevel<1: print 'repr(obj), repr(stuff): ', repr(obj), repr(stuff)
1571.                 if repr(obj) == repr(stuff):
1572.                   #if Yes, update only the Channels list
1573.                   if self.printLevel<1: print 'obj already present in dict as key, I update channels'
1574.                   partDict[cand][stuff].append( chan )
1575.             else:
1576.               #If No, add a new entry for that particle
1577.               if self.printLevel<1: print 'obj not present in dict as key, I add a new entry'
1578.               partDict[cand][obj] = []
1579.               partDict[cand][obj].append( chan )
1580.           if self.printLevel<1:
```

```
1581.                     for elem in partDict[cand]:
1582.                         print 'elem, id(elem), repr(elem), type(elem): ', elem, id(elem), repr(elem),
      type(elem)
1583.         #Preparing lists
1584.         partLists = {}
1585.         for cand in partDict:
1586.             if self.printLevel<2: print 'MergingCandidates():Preparing Lists - cand: ', cand
1587.             partLists[cand] = []
1588.             for obj in partDict[cand]:
1589.                 if self.printLevel<2: print 'cand, obj, partDict[cand][job], : ', cand, obj, partDict[cand][obj]
1590.                 if cand = 'met' or cand = 'truth_met':
1591.                     partLists[cand].append( { 'obj': obj, 'pt': obj.et(), 'channels': partDict[cand][obj] } )
1592.                 else:
1593.                     partLists[cand].append( { 'obj': obj, 'pt': obj.pt(), 'channels': partDict[cand][obj] } )
1594.             #Sorting objects for candidates on the pT
1595.             import operator #a module of the standard Python library
1596.             partLists[cand].sort(key=operator.itemgetter('pt'))
1597.             #   this sorts the list in place, according to the key 'Pt'
1598.             #   Suggestion: use 'newleps = sorted(key...)' if you want to avoid to overwrite the 'leptons' list
1599.             #Reverting the order, according to descending pt
1600.             partLists[cand].reverse()
1601.
1602.         if self.printLevel<1:
1603.             print 'partDict: ', partDict
1604.             print 'partLists: ', partLists
1605.
1606.         return partLists
1607.
1608.
1609.
1610.
      ##########################################################################################
1611.     def fillHists(self, treeVar, extraTreeVar, extraBranches, candidates, chan, treeName, **kws):
1612.         #Notice: 'chan' is not used in this version. It's only used when we want more TTree's in the
      output D3PD file.
1613.         oneTree = kws.get('oneTree', False) #True when passing candidatesMerged. Flase otherwise.
1614.         if self.printLevel < 5: print self.name,'.fillHists(): filling histograms and ttree/branches...'
1615.         if self.printLevel < 1: print 'candidates{}: ',candidates
1616.         if self.printLevel<2: print 'looping over treeVar...'
1617.         if self.printLevel<1: print'treeVar: ', treeVar
1618.         for part in treeVar:
1619.             if type(treeVar[part]) == dict:
1620.                 for var in treeVar[part]:
1621.                     if not var == 'Channels':
1622.                         treeVar[part][var].clear()
1623.                         if oneTree: treeVar[part]['Channels'].clear()
1624.                         if self.printLevel<2: print 'part:', part
1625.                         if part = 'met' or part = 'truth_met': #it has only one element #??? Do like with
      other no-array objects
1626.                             if oneTree: value =
      SUSYTools.AthenaObjWrapper.getCandidateVal(candidates[part][0]['obj'],var) #[0] because met has
      only one element
1627.                             else: value = SUSYTools.AthenaObjWrapper.getCandidateVal(candidates[part],var)
1628.                             treeVar[part][var].push_back(value)
```

```
                    if oneTree:
                        vecChan = ROOT.std.vector(ROOT.std.string)()
                        for ch in candidates[part][0]['channels']: #[0] because met has only one element
                            if self.printLevel<1: print 'push_back(ch), ch= ', ch
                            vecChan.push_back(ch)
                        if self.printLevel<1:
                            for i in range(vecChan.size()): print 'vecChan[%s]: ' % i, vecChan[i]
                        treeVar[part]['Channels'].push_back(vecChan)
                    else:
                        for icand, candidate in enumerate(candidates[part]):
                            if oneTree: value = SUSYTools.AthenaObjWrapper.getCandidateVal(candidate['obj'],var)
                            else: value = SUSYTools.AthenaObjWrapper.getCandidateVal(candidate,var)
                            if self.printLevel <=1: print 'part,var,value: ',part,var,value
                            treeVar[part][var].push_back(value)
                            if oneTree:
                                vecChan = ROOT.std.vector(ROOT.std.string)()
                                for ch in candidates[part][icand]['channels']:
                                    if self.printLevel<1: print 'push_back(ch), ch= ', ch
                                    vecChan.push_back(ch)
                                if self.printLevel<1:
                                    for i in range(vecChan.size()): print 'vecChan[%s]: ' % i, vecChan[i]
                                treeVar[part]['Channels'].push_back(vecChan)
                            pass
                        pass
                    pass
                pass
            pass
        elif part = 'channels' or part = 'channelsTopology':
            if self.printLevel <=2: print self.name,'.fillHists(): filling channels branch...'
            if self.printLevel <2 and not chan in self.customCutsChannels : print 'channelsList, channelsPassed: ', self.channelsList, self.channelsPassed
            elif self.printLevel <2 and chan in self.customCutsChannels: print 'channelsList'+chan,' channelsPassed'+chan,': ', getattr(self, 'channelsList'+chan), getattr(self, 'channelsPassed'+chan)
            treeVar[part].clear()
            if part=='channels': [ treeVar[part].push_back(value) for value in getattr(self,'channelsList'+chan) ]
            if part=='channelsTopology': [ treeVar[part].push_back(value) for value in getattr(self,'channelsPassed'+chan) ]
        else: #For example for 'eventNumber' or 'weightMC'
            if self.printLevel <=1: print 'part, value: ',part,getattr(self,part)
            treeVar[part].clear()
            try: #is it an iterable? e.g.list
                it = iter( getattr(self,part) )
                it = True
            except:
                it = False
            if it: #it's an iterable
                if self.printLevel <2: print 'iterable'
                [ treeVar[part].push_back(value) for value in getattr(self,part) ]
            else: #otherwise, if it's not an iterable
                if self.printLevel <2: print 'not-iterable'
                treeVar[part].push_back( getattr(self,part) )
                pass
```

```
1678.             pass
1679.
1680.       if self.printLevel<2: print 'looping over extraTreeVar...'
1681.       if self.printLevel<1: print'extraTreeVar: ', extraTreeVar
1682.
1683.       for part in extraTreeVar:
1684.          extraTreeVar[part]['Values'].clear()
1685.          if oneTree: extraTreeVar[part]['Channels'].clear()
1686.          if self.printLevel <2: print 'part, value: ',part, getattr(self,self.extraTreeVarFormula[part])
1687.          try: #is it an iterable? e.g.list
1688.             it = iter( getattr(self,extraTreeVarFormula[part]) )
1689.             it = True
1690.          except: it = False
1691.          if it: #it's an iterable
1692.             if self.printLevel <2: print 'iterable'
1693.             if oneTree:
1694.                pass # to be implemented if needed.
1695.             else:
1696.                [ extraTreeVar[part]['Values'].push_back(value) for value in
    getattr(self,self.extraTreeVarFormula[part]+chan) ]
1697.                extraTreeVar[part]['Channels'].push_back(chan)
1698.          else: #otherwise, if it's not an iterable
1699.             if self.printLevel <2: print 'not-iterable'
1700.             if oneTree:
1701.                for channel in self.extraTreeVarChannelsDict[ self.extraTreeVarFormula[part] ]:
1702.                   extraTreeVar[part]['Values'].push_back(
    self.extraTreeVarChannelsDict[self.extraTreeVarFormula[part]][channel] )
1703.                   extraTreeVar[part]['Channels'].push_back( channel )
1704.             else:
1705.                if self.printLevel<1: print part, extraTreeVar[part]
1706.                extraTreeVar[part]['Values'].push_back(
    getattr(self,self.extraTreeVarFormula[part]+chan) )
1707.                extraTreeVar[part]['Channels'].push_back( chan )
1708.
1709.       #Filling branches from SteeringFile.dumpContainers.
1710.       #Those objects are not related to any object selection and thus they will not be flagged with
    channel names
1711.       if 'jetenergy_FCAL2' in extraBranches:
1712.          extraBranches['jetenergy_FCAL2'].clear()
1713.          if 'jet' in candidates:
1714.             [
    extraBranches['jetenergy_FCAL2'].push_back(cand['obj'].getShape("energy_FCAL2",True)) for cand
    in candidates['jet'] ]
1715.          elif 'jet' in self.collections:
1716.             try: #is it an iterable? e.g.list
1717.                it = iter( self.collections['jet']['container'] )
1718.                it = True
1719.             except:
1720.                it = False
1721.             if it: #it's an iterable
1722.                [ extraBranches['jetenergy_FCAL2'].push_back( cand.getShape("energy_FCAL2",True) )
    for cand in self.collections['jet']['container'] ]
1723.             else: #it is not an iterable
1724.                extraBranches['jetenergy_FCAL2'].push_back(
```

```
self.collections['jet']['container'].getShape("energy_FCAL2",True) )
1725.                 pass
1726.             pass
1727.         pass
1728.     if 'jetenergy_FCAL1' in extraBranches:
1729.         extraBranches['jetenergy_FCAL1'].clear()
1730.         if 'jet' in candidates:
1731.             [
extraBranches['jetenergy_FCAL1'].push_back(cand['obj'].getShape("energy_FCAL1",True)) for cand
in candidates['jet'] ]
1732.         elif 'jet' in self.collections:
1733.             try: #is it an iterable? e.g.list
1734.                 it = iter( self.collections['jet']['container'] )
1735.                 it = True
1736.             except:
1737.                 it = False
1738.             if it: #it's an iterable
1739.                 [ extraBranches['jetenergy_FCAL1'].push_back( cand.getShape("energy_FCAL1",True) )
for cand in self.collections['jet']['container'] ]
1740.             else: #it is not an iterable
1741.                 extraBranches['jetenergy_FCAL1'].push_back(
self.collections['jet']['container'].getShape("energy_FCAL1",True) )
1742.                 pass
1743.             pass
1744.         pass
1745.     if 'jetenergy_FCAL0' in extraBranches:
1746.         extraBranches['jetenergy_FCAL0'].clear()
1747.         if 'jet' in candidates:
1748.             [
extraBranches['jetenergy_FCAL0'].push_back(cand['obj'].getShape("energy_FCAL0",True)) for cand
in candidates['jet'] ]
1749.         elif 'jet' in self.collections:
1750.             try: #is it an iterable? e.g.list
1751.                 it = iter( self.collections['jet']['container'] )
1752.                 it = True
1753.             except:
1754.                 it = False
1755.             if it: #it's an iterable
1756.                 [ extraBranches['jetenergy_FCAL0'].push_back( cand.getShape("energy_FCAL0",True) )
for cand in self.collections['jet']['container'] ]
1757.             else: #it is not an iterable
1758.                 extraBranches['jetenergy_FCAL0'].push_back(
self.collections['jet']['container'].getShape("energy_FCAL0",True) )
1759.                 pass
1760.             pass
1761.         pass
1762.     if 'electronAuthor' in extraBranches:
1763.         extraBranches['electronAuthor'].clear()
1764.         if 'electron' in candidates:
1765.             [ extraBranches['electronAuthor'].push_back(cand['obj'].author()) for cand in
candidates['electron'] ]
1766.         elif 'electron' in self.collections:
1767.             try: #is it an iterable? e.g.list
1768.                 it = iter( self.collections['electron']['container'] )
```

```
1769.                 it = True
1770.             except:
1771.                 it = False
1772.             if it: #it's an iterable
1773.                 [ extraBranches['electronAuthor'].push_back( cand.author() ) for cand in
      self.collections['electron']['container'] ]
1774.             else: #it is not an iterable
1775.                 extraBranches['electronAuthor'].push_back(
      self.collections['electron']['container'].author() )
1776.                 pass
1777.             pass
1778.         pass
1779.     if 'METSigL' in extraBranches:
1780.         extraBranches['METSigL'].clear()
1781.         if 'METSig' in candidates:
1782.             [ extraBranches['METSigL'].push_back(cand['obj'].sigL()) for cand in candidates['METSig']
      ]
1783.         elif 'METSig' in self.collections:
1784.             try: #is it an iterable? e.g.list
1785.                 it = iter( self.collections['METSig']['container'] )
1786.                 it = True
1787.             except:
1788.                 it = False
1789.             if it: #it's an iterable
1790.                 [ extraBranches['METSigL'].push_back( cand.sigL() ) for cand in
      self.collections['METSig']['container'] ]
1791.             else: #it is not an iterable
1792.                 extraBranches['METSigL'].push_back( self.collections['METSig']['container'].sigL() )
1793.                 pass
1794.             pass
1795.         pass
1796.     if 'truthPdgid' in extraBranches:
1797.         extraBranches['truthPdgid'].clear()
1798.         if 'truth' in candidates:
1799.             [ extraBranches['truthPdgid'].push_back(cand['obj'].pdgId()) for cand in candidates['truth'] ]
1800.         elif 'truth' in self.collections:
1801.             try: #is it an iterable? e.g.list
1802.                 it = iter( self.collections['truth']['container'] )
1803.                 it = True
1804.             except:
1805.                 it = False
1806.             if it: #it's an iterable
1807.                 [ extraBranches['truthPdgid'].push_back( cand.pdgId() ) for cand in
      self.collections['truth']['container'] ]
1808.             else: #it is not an iterable
1809.                 extraBranches['truthPdgid'].push_back( self.collections['truth']['container'].pdgId() )
1810.                 pass
1811.             pass
1812.         pass
1813.
1814.     #Write to the TTree
1815.     self.hsvc['/rec/'+treeName].Fill()
1816.
1817.     if self.printLevel < 3: print self.name,'.fillHists(): ...done'
```

```
1818.         return StatusCode.Success
1819.
1820.
1821.
1822.
        ##################################################################################################
1823.     def finalize(self):
1824.         if self.printLevel < 5: print self.name,'.finalize(): finalizing...'
1825.         return StatusCode.Success
1826.
```

---

-- RiccardoMariaBianchi - 11 Mar 2009

---

This topic: Main > ATLASWatchManAutomaticallyGeneratedAnalysisCodeExample
Topic revision: r3 - 2009-03-23 - RiccardoMariaBianchi