Table of Contents

PAT Cross Cleaning

Contents

Introduction

This page gives an overview of the subject of event cleaning (cleaning). It will answer the following questions:

- What is cleaning? Why is it important?
- How does PAT deal with the issue of object cleaning?
- How do I retrieve the result of PAT cleaning?
- How do I configure PAT for cleaning?

Furthermore, one can find a short tutorial and exercises at the end of this introduction to PAT cleaning.

REMARK: The PAT v1 version of the cleaning is at SWGuidePATv1Cleaning

What is event cleaning?

The high-level analysis objects (muons, electrons, etc.) are provided by dedicated reconstruction groups, the *Physics Object Groups* (POG). Each object basically belongs to a separate POG. As a consequence, it may happen that some low-level detector information (*e.g.*, energy deposit in the calorimeters) is used to reconstruct several candidates. As an example, clustered energy in the electromagnetic calorimeter (ECAL) may lead to the reconstruction of a photon. If a track points to the cluster, an isolated electron might also be formed. Finally, the energy deposit might also appear as a jet candidate in the event.

It is important to note that the resolution of these ambiguities is **analysis-dependent**:

- In the above example an analysis interested in QCD multi-jet events will only look at the jets and not care about disambiguation. There will be little interest in the rare cases in multi-jet events where a jet might just consist of an isolated electron.
- For an analysis of the jet multiplicity in top anti-top events in the semi-leptonic decay channel, with an electron in the final state, it is vital not to count the isolated electron as an additional jet. And so the corresponding jet needs to be removed from the collection of jets.
- Finally, analyses that rely on missing energy measured from all objects in the events (most notably SUSY searches) always have to ensure that there is no remaining ambiguity across any collection.

The ambiguities are, therefore, not a flaw in the reconstruction, but additional flexibility provided to the "user".

It is worth noting that ultimately cleaning leads to the idea of particle flow, which aims at an unambiguous identification and interpretation of each single particle in the event.

Cross-cleaning and object duplication

What was just described is sometimes referred to as *cross-cleaning*, since it checks for ambiguities *across* different collections of candidates (photons, electrons and jets in the above example). There is another type of ambiguity when objects of the same collection actually overlap: this is sometimes referred to as *duplication*. It happens when information from several sub-detectors is combined to form a single candidate, as is the case for electrons and muons:

object	component	component	reco algorithm
muon	TRACKING system	MUON system	SWGuideMuons
electron	TRACKING system	ECAL	SWGuideEgamma

- E.g. more then one track in the tracking system might point to a super cluster in the ECAL thus leading to more then one electron candidate.
- Or the reconstructed track segments in the muon system might appear as a so called *Standalone Muon* or as part of a refitted *Global Muon* due to the reconstruction algorithms used in the muon POG.

These ambiguities are generally taken care of by the corresponding POGs, and a sensible default behaviour is provided. Nevertheless a specific analysis might need to refine certain identification flags, indicated as *SelectionType* in the reco::Muon structure.

What are the collections of concern?

Typical object collections of concern for event cleaning are given in the table below:

collection of interest	to be cleaned from	relevance
jets	photons	analyses with jets and isolated (prompt) photons
jets	electrons	analyses with jets and isolated electrons
jets	muons	analyses with jets and muons
jets	taus	analyses with jets and taus
photon	electron	analyses of (prompt) photons

The example that will be discussed below is the cleaning of jets from isolated electrons, which is most common ambiguity in analyses relying on isolated leptons.

How to get the code

The cleaning of objects happens at the end of the PAT workflow, after the creation of PAT candidates and their pre-selection. The corresponding python files can be found in

- CMS.PhysicsTools/PatAlgos/python/cleaningLayer1

Find more details about the config file

Generally speaking, the cleaning consists in a series of stages (CMSSW modules), each of which produces one collection of clean items starting from an inclusive *input collection* and other *test collections* that have been already been cleaned in a previous stage. The ordering of modules matters!

Each cleaning module performs the following steps:

1. Discard from its input collection any items already present in one or more collections of clean objects of the same type (e.g. to make two exclusive lists of jets, one that doesn't include the ones overlapping with electrons and one that includes only those).
   Note: this has not yet been implemented.
2. Apply a generic preselection cut to the input objects.
3. Check for overlaps with one or more input collections. Depending on how each overlap checking test is configured, items that have overlaps can be kept or discarded. The algorithm saves in the clean objects pointers to the items with which they overlap, so that they can be retrieved in the later steps of the analysis (as long as they're still in the root file, otherwise one can only know the number of overlapping items for each test).
4. One generic selection cut is finally applied to the objects before saving them in the event.

As an example, let us have a look at the PAT electron cleaner electronCleaner_cfi.py:

```
cleanPatElectrons = cms.EDProducer("PATElectronCleaner",
    ## pat electron input source
    src = cms.InputTag("selectedPatElectrons"),
    # preselection (any string-based cut for pat::Electron)
    preselection = cms.string(''),
    # overlap checking configurables
    checkOverlaps = cms.PSet(
        muons = cms.PSet(
           src       = cms.InputTag("cleanPatMuons"),
           algorithm = cms.string("byDeltaR"),
           preselection        = cms.string(""),  # don't preselect the muons
           deltaR              = cms.double(0.3),
           checkRecoComponents = cms.bool(False), # don't check if they share some AOD object ref
           pairCut             = cms.string(""),
           requireNoOverlaps = cms.bool(False), # overlaps don't cause the electron to be discare
        )
    ),
    # finalCut (any string-based cut for pat::Electron)
    finalCut = cms.string(''),
)
```

In this example, electrons are checked against clean muons as follows:

- all electrons that passed previous PAT steps are checked (*selectedPatElectrons*);
- they are checked against all muons that passed the cleaning step;
- there is no additional pre-selection cut on electrons or muons;
- there is no check on their reco. components (could be, *e.g.*, super-clusters in the case of electron/photon cleaning)
- there is overlap if the ΔR distance between the electron and the muon is below 0.3;
- electrons overlapping will have a reference to the muon (they are not discarded);
- all electrons are saved in the event (no final cut).

Default configuration

The default configuration of the cleaning is as follows:

1. **Muons:** all muons are considered clean; no preselection or final selection cut is applied, and no overlaps are checked for.
2. **Electrons:** clean muons that overlap by ΔR (<0.3) and saved in the pat electrons but electrons are not discarded. No preselection or final selection cut is applied.
3. **Photons:** photons are checked for overlap against the clean electrons by supercluster seed, **and photons that overlap are discarded**. No preselection or final selection cut is applied.
4. **Taus:** by default **only taus that pass the discriminator by isolation are accepted**; overlaps by ΔR (<0.3) with electrons and muons are saved as references in the photons, but no cuts are applied.

5. **Jets:** overlaps with all the above collections are saved (&DeltaR<0.5). In addition, another overlap test is performed against tracker isolated electrons ($\Delta R<0.3$; electron trackIso<3, $p_t>10$ GeV) and saved with label `"tkIsoElectrons"` (this is identical to the old flag in PAT v1). No preselection or final selection is applied. All jets are kept.

Note: there is no MET cleaning, it doesn't make any sense, there is one and only one MET in the event.

N.B. The cleaning configuration is not necessarily optimal or even sensible.

Checking and handling overlaps

The result of the cleaning step can be checked directly from the output objects, unless the overlapping objects were discarded (*i.e.*, `requireNoOvelaps` was set to `True`, which is not the default behaviour, except for photons). The following `pat::Object` methods are available:

- **`hasOverlaps(coll)`** checks if there was any overlap with the collection named `coll`. The name corresponds to the name used in the configuration of the cleaner, in the `checkOverlaps` PSet (in the example above, it would be `muons`).
  Example: `myObject.hasOverlap("electrons")`
- **`overlapLabels()`** returns the full list of overlap checks (*i.e.*, names to be used in the other methods) that found at least one overlap (useful for inspection or debugging).
- **`overlaps(coll)`** returns a list of items with which overlap was found in collection `coll`, again corresponding to the name used in the configuration of the cleaner. The return value is a CandidatePtrVector, a vector of EDM pointers to Candidate objects. The following methods are then available:
  - `size()` to get the number of overlapping objects;
  - other basic Particle methods, like kinematics (`pt()`, `eta()`, etc.). This works only if the corresponding collection is still in the root file.
  - Use `dynamic_cast` to convert the pointers to a specific PAT Object type to access variables which are not in the base Candidate class
  - Check if two edm::Ptrs point to the same object, by comparing them with '=='

Here is a full example use of the cleaning result's retrieval:

```
const reco::CandidatePtrVector & elecs = myJet.overlaps("electrons");
std::cout << "This jet overlaps with " << elecs.size() <<  " electrons." << std::endl;
for (size_t i = 0; i < elecs.size(); ++i) {
     std::cout << "  electron " <<  i << " pt = " <<  elecs[i]->pt() <<  std::endl;
     // try to convert in a pat::Electron
     const pat::Electron *elec = dynamic_cast<const pat::Electron *>(&*elecs[i]);
     if (elec) {
           std::cout << "  electron " <<  i << " electron id " <<  ele->electronID("eidRobustTigh
     }
}
```

Example changes to the configuration

Some simple example of configuration changes can be done in the PAT cleaning:

- **The muon preselection cut can be changed**, *e.g.*, to apply some standard muon ID, p_t or isolation cut:

  ```
  process.cleanPatMuons.preselection = "isGood('GlobalMuonPromptTight')"
  ```

  As a result, only muons passing this pre-selection will be stored in the `cleanPatMuons` collection that is used in the later cleaning stages.

- **The electron preselection cut can be changed**, *e.g.*, to apply some standard electron ID, p_t or isolation cut.

```
process.cleanPatElectrons.preselection = "(electronID('eidRobustLoose') > 0) && (trackIso
```

- You can choose to **accept also photons that share a supercluster seed with an electrons**:

```
process.cleanPatPhotons.checkOverlaps.electrons.requireNoOvelaps = False
```

  Such photons will have a reference to the electron they overlap with.
- You can choose to **swap the order in which electrons and photons are cleaned**:

```
# take away electrons and put them after the photons
process.cleanPatObjects.remove(process.cleanPatElectrons)
process.cleanPatObjects.replace(process.cleanPatPhotons, process.cleanPatPhotons * process
# don't remove electrons from photons
process.cleanPatPhotons.checkOverlaps = cms.PSet()
# remove photons from electrons
process.cleanPatElectrons.checkOverlaps = cms.PSet(
     photons = cms.PSet(
           src       = cms.InputTag("cleanPatPhotons"),
           algorithm = cms.string("bySuperClusterSeed"),
           requireNoOvelaps = cms.bool(True), # discard electrons that overlap!
     ),
)
```

- You can add to the sequence to **run a copy of a cleaner module with different parameters**:

```
# make a copy of e.g. the electron cleaner
process.myCleanPatElectrons = process.cleanPatElectrons.clone()
# modify some configuration (e.g. the preselection)
process.myCleanPatElectrons.preselection = 'pt > 5'
# add it next to the electron cleaner
process.cleanPatObjects.replace(process.cleanPatElectrons, process.cleanPatElectrons + pro
# (optional) add it to the cleanPatSummary, to see how many items pass
process.cleanPatSummary.candidates.append( cms.InputTag("myCleanPatElectrons") )
```

Advanced configuration

Preselection and Final Cut

Preselection and final selection cuts use the standard physics tools cut parser, and can access all the variables and methods of the specific object (*e.g.*, all `pat::Electron` methods in the `PATElectronCleaner`).

In the final selection cut, the overlaps have been already added to the item, so one can use them as well.

Example: you can select taus that overlap with exactly one muon and not with any electron through:

```
process.cleanPatTaus.finalCut = "!hasOverlaps("electrons") &&  (overlaps("muons").size() == 1)"
```

Overlap checking

The overlap checking is used to mark or select items that have overlaps with others.

Different algorithms can be used; currently there is one generic overlap algorithm that checks overlaps by ΔR, optionally checking also for shared references to AOD objects (*e.g.*, tracks). There is also a more specific algorithm that selects objects sharing a supercluster seed; this algorithm can also serve as an example for developing similar specific algorithms.

The configuration of the overlap checking has the following structure:

```
checkOverlaps = cms.PSet(
       someCheck = cms.PSet(
               src       = cms.InputTag("collection to check overlap against"),
               algorithm = cms.string("the algorithm to use to search the overlap"),
               ... parameters specific to the algorithm, e.g. a deltaR cut ...
               requireNoOvelaps   = cms.bool(True if objects with overlaps must be discarded in t
        ),
       otherCheck = cms.PSet(
               ...
        ),
        ...
)
```

The typo in `requireNoOvelaps` has not been fixed in the code yet. Please use this syntax, even if it hurts your eyes!

Default Overlap by deltaR

The default overlap algorithm, based on ΔR distance, is named **`byDeltaR`** and does the following checks:

1. A **preselection** on the list of items to check overlaps against (*e.g.*. to flag only jets that overlap with isolated electrons). This is controlled by the parameter **`preselection`** , which is a generic cut string and has access to all variables of PAT objects.
2. A **ΔR** matching, with cone size equal to the **`deltaR`** parameter
3. If requested, it uses the Candidate Overlap Checker to check if the two items share a reference to the same RECO object (*e.g.*, if a PAT Muon and a PAT GenericParticle use the same `reco::Track`). This is controlled by the parameter **`checkRecoComponents`** .
4. It can apply a combined cut on the variables of the two objects of the pair, e.g. `0.5 < ele.pt/mu.pt < 1.5` . You can refer to the two members of the pair by their type ("ele", "mu", "tau", "gam" for Photon, "jet", "met", "part" for GenericParticle and "pf" for PFParticle). If the two particles are of the same type, you need to specify the number (e.g. "ele1", "mu2"); particle 1 is the from the collection you're cleaning, particle 2 is the one from the other collection.
   You can also access the ΔR (as `deltaR`) and the total 4-momentum (as `totalP4`).
   If the total collection you're checking the overlap against is not of PAT objects, you can generically refer to its items as "cand2" (because "cand1" is the item you are cleaning)

Overlap checker by SuperCluster Seed

This algorithm is there both as an example and because it was present in the previous versions of PAT cleaning. It is called `bySuperClusterSeed` , and doesn't take any additional parameter; it just checks if the two superclusters refer to the same seed.

Notes:

- Both objects should be of some type, inheriting from `reco::RecoCandidate` , and should have a non-null SuperCluster reference (e.g. Electron, Photon or PAT GenericParticle made from RecoEcalCandidate).
- The SuperClusters must be accessible, either because their collection is in the ROOT file or because they have been embedded in the PAT objects
- The seeds, instead, don't have to actually be available, as the algorithm just checks if the two seed references point to the same seed and doesn't look at the seed contents.

PAT exercises

You can find a tutorial and exercises on the SWGuidePATCrossObjectCollectionExercise.

Review status

Show Hide

Reviewer/Editor and Date (copy from screen)	Comments
KaiFengChen - 16 May 2010	Structure updates

Responsible: FredericRonga
Last reviewed by: KaiFengChen - 16 May 2010

This topic: CMSPublic > SWGuidePATCrossCleaning
Topic revision: r24 - 2011-12-02 - RogerWolf