# Table of Contents

# PAT MC Matching

Complete:

## This documentation refers to the MC matching in ***PAT*** in _CMSSW_3_6_X_

For documentation on MC & trigger matching in ***PAT*** in older *CMSSW* versions , please look at:

- *CMSSW_3_3_X*: revision **5** of this page (based on *CMSSW_3_3_4*)
- *CMSSW_3_1_X*: revision **4** of this page (based on *CMSSW_3_1_2*)
- *CMSSW_2_2_X*: revision **2** of this page (based on *CMSSW_2_2_13*)

# Introduction

## What is "matching"?

Matching means the association of objects from different collections based on their similarity in spatial coordinates and/or kinematics. Discrete object properties like e.g. a general type or charge can be used to restrict the possible matches additionally.

Goal of the matching is to find representations of *the same object* in different collections.

## Implementation

All matching set-ups described on this page use the same tool, the class template **`reco::PhysObjectMatcher`**. It is invoked by concrete class definitions, specifying the types of the input collections (all derived from base class **`reco::Candidate`**), a (pre-)selector, the matching definition and the ranking. The provided concrete classes are introduced in the following sections.

# MC Matching

The PAT MC matching offers the opportunity to compare and associate PAT objects with generator objects. Of course, this is only applicable to MC and not to real data .

Since the number of meaningful matches is limited, the PAT already provides a comprehensive default within its standard configuration. However, it might be necessary to re-evaluate the existing settings or to create new matches. This is described in this section.

The matching is based on the existence of sufficient generator object collections in the input files. The AOD data tier provides these collections by default. However, it is worth to check if a desired collection is present in the actually used input sample. Especially, generator level jets from taus are *not* in AOD by default and have to be produced as explained in here.

The whole procedure is split into two steps:

- match the MC objects to the PAT objects
- add the matching MC objects to the PAT objects

## Setting up the MC matches

The configuration files for this step are: CMS.PhysicsTools/PatAlgos/python/mcMatchLayer0/*Match_cfi.py. There exists one configuration for each particle type. One can modify the existing default settings or append new ones to the files.

### Match to generator particles

The dedicated module is an *EDFilter* of the name `MCMatcher`, which is based on matches in the - space. It takes nine configurable parameters:

- the `InputTag` **`src`**:
  - the RECO collection to match to;
  - has to be the label of a collection available in AOD;
  - has to be of the type `reco::CandidateView`;
- the `InputTag` **`matched`**:
  - the MC particle collection of type **`reco::GenParticleCollection`** to match;
  - has to be present in the input sample;
- the `vint32` **`mcPdgId`**:
  - defines the particle types to match by PDG ID;
- the `vint32` **`mcStatus`**:
  - PYTHIA status code of particles to match;
  - 1: stable, 2: shower, 3: hard scattering;
- the `bool` **`checkCharge`**:
  - only equally charged objects are matched, if set to `True`;
- the `double` **`maxDeltaR`**:
  - maximum distance in - space to apply the match;
- the `double` **`maxDPtRel`**:
  - maximum relative *Pt* difference to apply the match;
- the `bool` **`resolveAmbiguities`**:
  - only one match per trigger object, if set to `True`;
- the `bool` **`resolveByMatchQuality`**:
  - stores best match instead of first, if set to `True`;
  - works only, if also **`resolveAmbiguities`** is set to `True`.

The values for **`maxDPtRel`** and **`maxDeltaR`** are not tuned yet, but it is recommended to use the values found in th default configurations per object type, which are

| object | electron | photon | muon | tau to jet | jet |
|---|---|---|---|---|---|
| **`maxDPtRel`** | 0.5 | 1.0 | 0.5 | 3.0 | 3.0 |
| **`maxDeltaR`** | 0.5 | 0.2 | 0.5 | 0.1 | 0.4 |

Putting this together, an example module configuration becomes e.g.

```
electronMatch = cms.EDFilter("MCMatcher",
    src     = cms.InputTag("gsfElectrons"),
    matched = cms.InputTag("genParticles"),
    mcPdgId     = cms.vint32(11),
    checkCharge = cms.bool(True),
    mcStatus = cms.vint32(1),
    maxDeltaR = cms.double(0.5),
    maxDPtRel = cms.double(0.5),
    resolveAmbiguities = cms.bool(True),
    resolveByMatchQuality = cms.bool(False),
)
```

As an alternative, there exists also an **`MCMatcherByPt`**, which is based on matches in the *relPt* space. The functionality is identical, since both matchers are instances of the same templated code.

### Match to generator level jets

Generator level jets are jets reconstructed from generator particles. The dedicated module is an *EDFilter* of the name `GenJetMatcher`, which is based on matches in the - space. It takes the identical configurable parameters as the `MCMatcher`, but with the following differences:

- the input collection to **`matched`** has to be of the type `reco::GenJetCollection`;
- the parameters **`mcPdgId`**, **`mcStatus`** and **`checkCharge`** are meaningless in this context and remain undefined.

A possible configuration would be e.g.

```
patJetGenJetMatch = cms.EDFilter("GenJetMatcher",     # cut on deltaR; pick best by deltaR
    src         = cms.InputTag("ak5CaloJets"),        # RECO jets (any View<Jet> is ok)
    matched     = cms.InputTag("ak5GenJets"),         # GEN jets  (must be GenJetCollection)
    mcPdgId     = cms.vint32(),                       # n/a
    mcStatus    = cms.vint32(),                       # n/a
    checkCharge = cms.bool(False),                    # n/a
    maxDeltaR   = cms.double(0.4),                    # Minimum deltaR for the match
    maxDPtRel   = cms.double(3.0),                    # Minimum deltaPt/Pt for the match (not used
    resolveAmbiguities    = cms.bool(True),           # Forbid two RECO objects to match to the sa
    resolveByMatchQuality = cms.bool(False),          # False = just match input in order; True =
)
```

## Setting up the addition of the matches to the PAT objects

The configurations for this step are found in the PAT producer modules CMS.PhysicsTools/PatAlgos/python/producersLayer1/*Producer_cfi.py for leptons, jets and MET.

The set of configurable parameters differs for different types of produced particles. The configurable parameters in the producer modules are:

- the `bool` **`addGenMatch`** (jets: **`addGenPartonMatch`**):
  - ♦ general switch to include MC matches into the PAT objects
- the `bool` **`embedGenMatch`** (jets: **`embedGenPartonMatch`**):
  - ♦ matched generator particles are stored as data members of the PAT objects, if set to `True`;
  - ♦ a reference is stored otherwise
- the `InputTag` *or* `VInputTag` **`genParticleMatch`** (jets: `InputTag` **`genPartonMatch`**):
  - ♦ match(es) to be included;
  - ♦ specified by the run MC matching module(s).

The tau and jet producer have three additional ones for matches to generator level jets:

- the `bool` **`addGenJetMatch`**
- the `bool` **`embedGenJetMatch`**:
  - ♦ matched generator jets are stored as data members of the PAT objects, if set to `True`;
  - ♦ a reference is stored otherwise
- the `InputTag` **`genJetMatch`**

Also the MET has the possibility to add the generator MET by the configurable parameters

- the `bool` **`addGenMET`**
- the `InputTag` **`genMETSource`**

without performing any matching in PAT.

Following the examples in the preceding section, this would lead to these lines in the electron producer:

```
    addGenMatch      = cms.bool(True),
    embedGenMatch    = cms.bool(True),
    genParticleMatch = cms.InputTag("electronMatch")
```

resp. to the following lines in the jet producer:

```
   addGenPartonMatch   = cms.bool(True),
    embedGenPartonMatch = cms.bool(True),
    genPartonMatch      = cms.InputTag("patJetPartonMatch"),
    addGenJetMatch      = cms.bool(True),
    embedGenJetMatch    = cms.bool(True),
    genJetMatch         = cms.InputTag("patJetGenJetMatch"),
    addPartonJetMatch   = cms.bool(False),
    partonJetSource     = cms.InputTag("NOT_IMPLEMENTED"),
```

## Include MC matching into the workflow and event content

### Workflow

The MC matching modules as defined in CMS.PhysicsTools/PatAlgos/python/mcMatchLayer0/*Match_cfi.py are imported into the relevant producer sequences in CMS.PhysicsTools/PatAlgos/python/producersLayer1/*Producer_cff.py, and the matching modules are scheduled there before the actual producer module, e.g.:

```
[...]
from CMS.PhysicsTools.PatAlgos.mcMatchLayer0.electronMatch_cfi import *
[...]
makePatElectrons = cms.Sequence(
[...]
electronMatch *
patElectrons
)
```

In order to include the generator level jets from taus, which are missing in AOD, the sequence in PhysicsTools/PatAlgos/python/producersLayer1/tauProducer_cff.py is extended as follows:

```
[...]
from CMS.PhysicsTools.JetMCAlgos.TauGenJets_cfi import *
from CMS.PhysicsTools.PatAlgos.mcMatchLayer0.tauMatch_cfi import *
[...]
makePatTaus = cms.Sequence(
    [...]
    tauGenJets     *    # produces MC jets from taus
    tauGenJetMatch *    # takes 'tauGenJets' as parameter matched
    patTaus
)
```

Possible MC matching sequences are also provided in PhysicsTools/PatAlgos/python/mcMatchLayer0/mcMatchSequences_cff.py, which however is currently not used in the standard work flow.

### Event content

By default, PAT MC particle matches are stored by reference in PAT objects. This means, that the original collections containing the MC objects need to be kept in the PAT event content. This is maintained in the file PhysicsTools/PatAlgos/python/patEventContent_cff.py. The used event content has to contain the line

```
'keep recoGenParticles_genParticles_*_*'
```

which is e.g. the case for **`patExtraAodEventContent`** (to be use *additionally* to the default **`patEventContent`**). This is necessary, if *any* match is stored by reference. Only if *all* particle matches are embedded, this event content can be omitted.

The matches to MC jets and the MET are stored by embedding in the PAT objects anyway, so the event content needs no modification due to them.

## Analyzing MC matches

In this section, only the existing user interface is described. Examples are being worked out in the hands-on exercise.

The PAT object class template provides the following methods to access MC match information:

- `reco::GenParticleRef genParticleRef(size_t idx=0) const;`
  - ♦ get MC particle reference;
  - ♦ index can be specified, if more than one have been stored;
- `reco::GenParticleRef genParticleById(int pdgId, int status) const;`
  - ♦ get MC particle reference with specified PDG ID and status code; see also this HN message
- `const reco::GenParticle * genParticle(size_t idx=0) const;`
  - ♦ get MC particle pointer;
- `size_t genParticlesSize() const;`
  - ♦ number of stored MC matches;
- `std::vector<reco::GenParticleRef> genParticleRefs() const;`
  - ♦ return all MC particles;
- `void setGenParticleRef(const reco::GenParticleRef &ref, bool embed=false);`
  - ♦ set MC particle reference
- `void addGenParticleRef(const reco::GenParticleRef &ref);`
  - ♦ add MC particle;
  - ♦ embedding, if already embedded MC match exists;
- `void setGenParticle( const reco::GenParticle &particle );`
  - ♦ set MC particle (by embedding);
  - ♦ for MC particles not in the event;
- `void embedGenParticle();`
  - ♦ embed MC particles stored as reference;

In general, returned references are transient, if the MC particles have been embedded.

In addition to these methods, further functionalities are provided by the concrete PAT object classes. The particular interfaces are found in DataFormats/PatCandidates/interface/, especially in:

- Lepton.h,
- Photon.h,
- Tau.h,
- Jet.h and
- MET.h

The interfaces to the used data formats to store MC info are DataFormats/HepMCCandidate/interface/GenParticle.h for MC particles and DataFormats/JetReco/interface/GenJet.h for MC jets.

-- VolkerAdler - 12 Jun 2009

-- SudhirMalik - 8 March 2010 ( updated to CMSSW_3_5_X)

---

This topic: CMSPublic > SWGuidePATMCMatching
Topic revision: r9 - 2014-04-21 - DinkoFerencek