Table of Contents

PAT MC and Trigger Matching

Complete:

This documentation refers to ***PAT v1***.

For documentation on MC and trigger matching in ***PAT v2*** , please look at:

- SWGuidePATTrigger
- SWGuidePATMCMatching

Introduction

This page documents the lecture given on **MC and Trigger Matching** in the framework of the e-learning in CMS activity Using Physics Analysis Toolkit (PAT) in your analysis.

The lecture schedule is found on Indico.

The whole e-learning activity is based on the PAT as integrated in **CMSSW_2_2_3**.

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:

- (*PAT layer 0*)
  match the MC objects to the PAT objects
- (*PAT layer 1*)
  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 PAT **layer 0** collection to match to;
  - has to be the label of a module defined in a file `CMS.PhysicsTools/PatAlgos/python/cleaningLayer0/*Cleaner_cfi`;
  - 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("allLayer0Electrons"),
    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 *Pt* 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.

```
jetGenJetMatch = cms.EDFilter("GenJetMatcher",
    src      = cms.InputTag("allLayer0Jets"),
    matched  = cms.InputTag("iterativeCone5GenJets"),
    mcPdgId  = cms.vint32(),       # n/a
    mcStatus = cms.vint32(),       # n/a
    checkCharge = cms.bool(False), # n/a
    maxDeltaR = cms.double(0.4),
    maxDPtRel = cms.double(3.0),
    resolveAmbiguities = cms.bool(True),
    resolveByMatchQuality = cms.bool(False)
)
```

Setting up the addition of the matches to the PAT objects

The configurations for this step are found in the PAT layer 1 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 electron, muon and photon producer modules are:

- the `bool` **`addGenMatch`**:
  - general switch to include MC matches into the PAT layer 1 objects
- the `bool` **`embedGenMatch`**:
  - matched generator particles are stored as data members of the PAT objects, if set to `True`;
  - a reference is stored otherwise
- the `InputTag` **`genParticleMatch`**:
  - match to be included;
  - specified by the MC matching module run in PAT layer 0.

The tau and jet producer have the same parameters, but also thwo additional ones for matches to generator level jets:

- the `bool` **`addGenJetMatch`**
- the `InputTag` **`genJetMatch`**

Embedding of the MC jets is not provided in this case.
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 layer 0.

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

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

resp. to the following lines in the jet producer:

```
    addGenPartonMatch = cms.bool(True),                # not in the matching example
    embedGenPartonMatch = cms.bool(False),             # not in the matching example
    genPartonMatch    = cms.InputTag("jetPartonMatch"), # not in the matching example
    addGenJetMatch    = cms.bool(True),
    genJetMatch       = cms.InputTag("jetGenJetMatch")
```

Include MC matching into the workflow and event content

Workflow

Possible MC matching sequences are provided in **`PhysicsTools/PatAlgos/python/mcMatchLayer0/mcMatchSequences_cff.py`**, which is imported by **`PhysicsTools/PatAlgos/python/patLayer0_cff.py`** to include the sequences in the PAT layer 0 workflow. In any case, the MC matching sequence(s) have to be performed after the PAT layer 0 cleaners, e.g.

```
patLayer0_withoutTrigMatch = cms.Sequence(
        patBeforeLevel0Reco *
        patLayer0Cleaners *
        patHighLevelReco *
        patMCTruth
)
```

in order to have all collections available needed for the matching.

This file shows also an example for the inclusion of missing generator level jets from taus. It contains the lines

```
from CMS.PhysicsTools.JetMCAlgos.TauGenJets_cfi import tauGenJets
patMCTruth_Tau =  cms.Sequence ( [...]
    tauGenJets *
    tauGenJetMatch
)
```

where the module **`tauGenJetMatch`** has the parameter **`matched`** set to `tauGenJets`.

Event content

By default, PAT MC particle matches are stored by reference in PAT layer 1 objects. This means, that the original collections containing the MC objects need to be kept in the event content in PAT layer 1. This is maintained in the file PhysicsTools/PatAlgos/python/patLayer1_EventContent_cff.py. It has to contain the

line

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

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.

Anyway, if for some reason PAT layer 0 is run separately from layer 1, it has to be ensured that the necessary information is saved in the layer 0 event output EDM file. This is done in the file PhysicsTools/PatAlgos/python/patLayer0_EventContent_cff.py. It has to contain the lines

```
        'keep *_genParticles_*_*',
        'keep *_iterativeCone5GenJets_*_*',
        'keep *_tauGenJets_*_*',
        'keep *_genMet_*_*',
        'keep recoGenParticlesedmAssociation_*_*_*',
        'keep recoGenJetsedmAssociation_*_*_*',
```

which is the default.

To check the intermediate products of the MC matchers as produced in PAT layer 0 even in the PAT layer 1 output, one can append the following lines to the main configuration file:

```
patLayer0EventContentMCMatch = [
    'keep recoGenParticlesedmAssociation_*_*_*',
    'keep recoGenJetsedmAssociation_*_*_*'
]
process.out.outputCommands += patLayer0EventContentMCMatch
```

The according collections of type `edm::Association<reco::GenParticles>` and `edm::Association<reco::GenJets>` appear in the output EDM file and can be browsed resp. analyzed.

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;
- `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.

Trigger Matching

The PAT trigger matching offers the opportunity to compare and associate PAT objects with trigger objects. It can identify objects which actually fired a given trigger, e.g. those trigger(s) a particular analysis is based on. Currently, the trigger matching is limited to *L3* objects only.

Contrary to the MC matching, the variety of triggers compared to generator object collections makes it impractical to provide a comprehensive default within the PAT standard configurations. Thus it is mostly necessary for the user to define trigger matches newly. This is described in this section.

The matching techniques are identical to that used for MC objects. However, the trigger objects to match are not accessible in AOD in a simple way and have to be produced in the appropriate format (`reco::Candidate` resp. inheritors of it) first. Due to that, the whole procedure is split into three steps:

1. (*PAT layer 0*)
   produce the trigger objects of interest,
2. (*PAT layer 0*)
   match the trigger objects to the PAT objects of interest,
3. (*PAT layer 1*)
   add the matching trigger objects as data members to the PAT objects.

However, before one cares for the configuration, one should know which trigger matches are of interest!

Choice of trigger matches of interest

The PAT trigger matching is unlimited in the combination of trigger objects and PAT objects. In principle, any collection of trigger objects can be matched to any collection of PAT objects. This offers e.g. the following combinations:

- match trigger electrons to PAT electrons (obvious),
- match trigger photons to PAT electrons (still obvious),
- match trigger electrons to PAT jets (check for possible fake electron triggers),
- match trigger MET to PAT muons (check for possible fake MET triggers),

- match trigger muons to PAT photons

and many more.

The usual starting point is a given *trigger path*. The corresponding question the trigger matching can address is:

"Which PAT objects let the event pass a given trigger path?"

Lists of available trigger paths are avaible at SWGuideGlobalHLT or in more detail at the TriggerTables.

Setting up the trigger object production

The configuration file for this step is
PhysicsTools/PatAlgos/python/triggerLayer0/patTrigProducer_cfi.py.

Find necessary information

Unfortunately, it is not easy to connect trigger objects to a given trigger path based on the information in AOD. So, the preparation for the set-up to extract the desired trigger objects is rather complicated. One needs to find out, which *filter module* was run in a given path to know the label of the trigger object collection to use. The procedure is described in the heading comments of the configuration file. However, this method to find the filter module is error prone.

A simplification of that procedure is provided by a little CMSSW job (not in release). Copy&paste the following configuration to **my_hltAnalysis_cfg.py**

```
import FWCore.ParameterSet.Config as cms
process = cms.Process( "HLTPROV" )
process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring('file:/afs/cern.ch/cms/PRS/top/cmssw-data/relval200-for-pat
)
process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(1))
process.load( "HLTrigger.HLTcore.hltEventAnalyzerAOD_cfi" )
process.hltEventAnalyzerAOD.triggerName = cms.string( '@' )
process.load( "HLTrigger.HLTcore.triggerSummaryAnalyzerAOD_cfi" )
process.p = cms.Path(
    process.hltEventAnalyzerAOD       +
    process.triggerSummaryAnalyzerAOD
)
```

and run it with a pipe through `grep`:

```
cmsRun my_hltAnalysis_cfg.py | grep -B 3 "'L3' filter in slot"
```

The output is mainly a list of HLT paths run on the input including the last active module and *L3 filter* and looks like

```
[...]
HLTEventAnalyzerAOD::analyzeTrigger: path HLT_MET65_HT350 [33]
 Trigger path status: WasRun=1 Accept=0 Error =0
 Last active module - label/type: hlt1MET65/HLT1CaloMET [24 out of 0-26 on this path]
 'L3' filter in slot 24 - label/type hlt1MET65/HLT1CaloMET
--
HLTEventAnalyzerAOD::analyzeTrigger: path HLT_IsoEle15_LW_L1I [46]
 Trigger path status: WasRun=1 Accept=0 Error =0
 Last active module - label/type: hltL1IsoLargeWindowSingleElectronTrackIsolFilter/HLTElectronTra
```

```
 'L3' filter in slot 46 - label/type hltL1IsoLargeWindowSingleElectronTrackIsolFilter/HLTElectron
--
HLTEventAnalyzerAOD::analyzeTrigger: path HLT_LooseIsoEle15_LW_L1R [47]
 Trigger path status: WasRun=1 Accept=0 Error =0
 Last active module - label/type: hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter/HLTEl
 'L3' filter in slot 64 - label/type hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter/HL
--
[...]
```

Here, one can see the name and number of the trigger path at the end of the first line of an entry (e.g. `HLT_LooseIsoEle15_LW_L1R [47]` in the last entry). The label and type of the relevant L3 filter are shown in the third line (`hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter/HLTElectronTrackIsolFilterRegional`). The label (`hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter`) is, what is needed to configure the trigger matching.

One can also look for a specific trigger path by modifying in **`my_hltAnalysis_cfg.py`** the line

```
process.hltEventAnalyzerAOD.triggerName = cms.string("@")
```

to

```
process.hltEventAnalyzerAOD.triggerName = cms.string("[name of trigger path]")
```

so that the output is shorter, or/and one can omit the pipe through `grep` to have information on all run paths.

⚠ It might happen, that the trigger path of interest does not fire in the first event of the sample. In case it aborts *before* the L3 filter is reached, the filter label will not appear in the output. In this case, the number of processed events has to be increases in **`my_hltAnalysis_cfg.py`** by e.g.

```
process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(100))
```

until the filter appears. Best in this case is to dump the output to a log file to scan.

Create configuration

The trigger object production is set up in the already mentioned configuration file. It contains already some example configurations by default (maybe your desired trigger is already present?). One can simply append a new configuration at the bottom, e.g. for the already "known" trigger path `HLT_LooseIsoEle15_LW_L1R [47]`.

The dedicated module is an *EDProducer* of the name **`PATTrigProducer`**. It takes two configurable parameters:

- the `InputTag` **`triggerEvent`**:
  - points to the source of trigger information in the input which is of the format `trigger::TriggerEvent`;
- the `InputTag` **`filterName`**:
  - points to the actual collection of trigger objects within the `trigger::TriggerEvent`;
  - is found as described earlier;
  - ⚠ needs also the process name!

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

```
myTrigObjects = cms.EDProducer("PATTrigProducer",
    triggerEvent = cms.InputTag("hltTriggerSummaryAOD","","HLT"),
    filterName = cms.InputTag("hltL1NonIsoHLTLooseIsoSingleElectronLWEt15TrackIsolFilter","","HLT
)
```

to be appended to the configuration.

This produces an EDM collection **patTriggerPrimitivesOwned_myTrigObjects__[PAT process name]** of the type `edm::OwnVector<pat::TriggerPrimitive>`.

The data format `pat::TriggerPrimitive`

The interface to this class is in **DataFormats/PatCandidates/interface/TriggerPrimitive.h**.

The **pat::TriggerPrimitive** inherits from **reco::LeafCandidate**. This section explains only additional or modified features.

Data members

On top of the data members of a **reco::LeafCandidate**, the following data members are defined:

- `std::string filterName_;`
  - ♦ label of the trigger filter this object was used in;
- `int triggerObjectType_;`
  - ♦ trigger object type as defined in **DataFormats/HLTReco/interface/TriggerTypeDefs.h**;
  - ♦ ranges from 81 to 90 (**L1**) and from 91 to 102 (**HLT**).

In addition, the data member of `pdgId_` of **reco::LeafCandidate** has a slightly different meaning. In is used to store the trigger object ID as `trigger::TriggerObject::id_`, which is similar but not equal to the usual PDG ID.

Constructors

Beside the default constructor (and destructor), **pat::TriggerPrimitive** provides the following constructors:

- `TriggerPrimitive( const reco::Particle::LorentzVector & aVec, const std::string aFilt = "", const int aType = 0, const int id = 0 );` with the parameters
  - ♦ obligatory `reco::Particle::LorentzVector` to define the object's kinematics;
  - ♦ mandatory `std::string` to set data member `filterName_`;
  - ♦ mandatory `int` to set data member `triggerObjectType_`;
  - ♦ mandatory `int` to set data member `pdgId` of **reco::LeafCandidate**;
- `TriggerPrimitive( const reco::Particle::PolarLorentzVector & aVec, const std::string aFilt = "", const int aType = 0, const int id = 0 );`
  - ♦ similar to the other constructor, but using a polar Lorentz-vector.

Methods

The methods are self-explanatory and correspond to the mandatory arguments of the constructors.

Setters

- `void setFilterName( const std::string aFilt );`
- `void setTriggerObjectType( const int aType );`
- `void setTriggerObjectId( const int id );`

Getters

- `const std::string & filterName() const;`
- `const int triggerObjectType() const;`

- `const int triggerObjectId() const;`

Setting up the trigger matches

The configuration file for this step is **`PhysicsTools/PatAlgos/python/triggerLayer0/patTrigMatcher_cfi.py`**. It contains already some example matches by default, corresponding to the default trigger object producers in **`CMS.PhysicsTools/PatAlgos/python/triggerLayer0/patTrigProducer_cfi.py`**. One can simply append a new configuration at the bottom.

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

- the `InputTag` **`src`**:
  - the PAT **layer 0** collection to match to;
  - has to be the label of a module defined in a file `CMS.PhysicsTools/PatAlgos/python/cleaningLayer0/*Cleaner_cfi`;
- the `InputTag` **`matched`**:
  - the trigger object collection to match;
  - defined by the module name(s) provided in the trigger object production;
- the `double` **`maxDPtRel`**:
  - maximum relative *Pt* difference to apply the match;
- the `double` **`maxDeltaR`**:
  - maximum distance in - space 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, the example module configuration becomes e.g.

```
myTrigMatches = cms.EDFilter("PATTrigMatcher",
    src     = cms.InputTag("allLayer0Electrons"),
    matched = cms.InputTag("myTrigObjects"),
    maxDPtRel = cms.double(0.5),
    maxDeltaR = cms.double(0.5),
    resolveAmbiguities    = cms.bool(True),
    resolveByMatchQuality = cms.bool(False),
)
```

to be appended to the configuration.

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

This produces an EDM collection **`patTriggerPrimitivesOwnededmAssociation_myTrigMatches__[PAT process name]`** of the type `edm::Association<edm::OwnVector<pat::TriggerPrimitive> >`.

Setting up the embedding of the trigger matches into the PAT objects

The configurations for this step are found in the PAT layer 1 producer modules **`CMS.PhysicsTools/PatAlgos/python/producersLayer1/*Producer_cfi.py`**. As the already described configuration files for PAT layer 0, they have some example trigger matches integrated by default.

The two configurable parameters in each producer module are:

- the `bool` **`addTrigMatch`**:
  - general switch to embed trigger matches into the PAT layer 1 objects;
- the `VInputTag` **`trigPrimMatch`**:
  - list of matches to be embedded;
  - possibilities specified by trigger matching modules run in PAT layer 0.

The easiest way to maintain the trigger matching in PAT layer 1 is to steer it centrally from within the main configuration file. Fo this purpose, one adds the following lines to it **after** PAT layer 1 modules have been loaded:

```
process.allLayer1Electrons.addTrigMatch = True
process.allLayer1Electrons.trigPrimMatch = [cms.InputTag("electronTrigMatchHLT1ElectronRelaxed"),
process.allLayer1Jets.addTrigMatch = True
process.allLayer1Jets.trigPrimMatch = [cms.InputTag("jetTrigMatchHLT1ElectronRelaxed"), cms.Input
process.allLayer1METs.addTrigMatch = True
process.allLayer1METs.trigPrimMatch = [cms.InputTag("metTrigMatchHLT1MET65")]
process.allLayer1Muons.addTrigMatch = True
process.allLayer1Muons.trigPrimMatch = [cms.InputTag("muonTrigMatchHLT1MuonNonIso"), cms.InputTag
process.allLayer1Photons.addTrigMatch = True
process.allLayer1Photons.trigPrimMatch = [cms.InputTag("photonTrigMatchHLT1PhotonRelaxed")]
process.allLayer1Taus.addTrigMatch = True
process.allLayer1Taus.trigPrimMatch = [cms.InputTag("tauTrigMatchHLT1Tau")]
```

This is just a replication of the default, but can be modified now in order to customize the trigger matching without excessive file browsing.

To add a newly designed trigger match, append the `InputTag` of the matcher module to the appropriate list. In the described example, the trigger objects (their type is indeed of no interest here) are matched to PAT electrons, so the matcher module is added to the electron production now:

```
process.allLayer1Electrons.trigPrimMatch = [cms.InputTag("electronTrigMatchHLT1ElectronRelaxed"),
```

Now, all steps to have a complete new trigger matching in the PAT are completed and the **CMSSW** job can be run.

Including trigger object production and matching into the workflow and event content

Workflow

Of course, the matching can only be performed after the objects to match have been put into the event, that means after the trigger object production and the PAT layer 0 cleaning. The easiest way to ensure this is exemplified in the PAT default configurations:

The file **`CMS.PhysicsTools/PatAlgos/python/triggerLayer0/patTrigMatcher_cfi.py`** imports **`CMS.PhysicsTools/PatAlgos/python/triggerLayer0/patTrigProducer_cfi.py`**, so that both, producer and matcher modules, are available. Then a sequence is defined, which puts both into the correct order. In the existing example, this looks like

```
myTrigMatchSequence = cms.Sequence(myTrigObjects * myTrigMatches)
```

to be appended to **CMS.PhysicsTools/PatAlgos/python/triggerLayer0/patTrigMatcher_cfi.py**. The asterisk indicates, that `myTrigMatches` produces input needed by `myTrigObjects`.
⚠ Trigger matching modules should not be used more than once!
However, producer modules can provide input to more than one matcher module.

The file **PhysicsTools/PatAlgos/python/triggerLayer0/patTrigMatcher_cfi.py** is imported then by **PhysicsTools/PatAlgos/python/patLayer0_cff.py** via **PhysicsTools/PatAlgos/python/triggerLayer0/trigMatchSequences_cff.py**, so that the matching sequences are available in PAT layer 0. This offers several opportunities to put a new sequence into the workflow:

- **CMS.PhysicsTools/PatAlgos/python/triggerLayer0/trigMatchSequences_cff.py**:
  e.g. in sequence `patTrigMatch`, which is in the default path;
- **CMS.PhysicsTools/PatAlgos/python/patLayer0_cff.py**:
  e.g. appended to sequence `patLayer0`;
- main configuration file, e.g.
  CMS.PhysicsTools/PatAlgos/test/patLayer1_fromAOD_full.cfg.py:
  directly to the path after `process.patLayer0`.

In all these cases, the trigger matching sequence is performed after the PAT layer 0 cleaners. In the last case, the path would look like

```
process.p = cms.Path(
    process.patLayer0 *
    process.myTrigMatchSequence *
    process.patLayer1
)
```

Event content

If the complete PAT (layers 0 & 1) is run at once, no adaption to the event content is needed, since the matches are embedded in the PAT objects.

Anyway, if for some reason PAT layer 0 is run seperately from layer 1, it has to be ensured that the necessary information is saved in the layer 0 output EDM file. This is done in the file **PhysicsTools/PatAlgos/python/patLayer0_EventContent_cff.py**. It has to contain the lines

```
        'keep patTriggerPrimitivesOwned_*_*_*',
        'keep patTriggerPrimitivesOwnededmAssociation_*_*_*'
```

which is the default.

To check the intermediate products of the trigger producers and matchers as produced in PAT layer 0 even in the PAT layer 1 output, one can append the following lines to the main configuration file:

```
patLayer0EventContentTriggerMatch = [
    'keep patTriggerPrimitivesOwned_*_*_*',
    'keep patTriggerPrimitivesOwnededmAssociation_*_*_*'
]
process.out.outputCommands += patLayer0EventContentTriggerMatch
```

The according collections of type `edm::OwnVector<pat::TriggerPrimitive>` and `edm::Association<edm::OwnVector<pat::TriggerPrimitive> >` appear in the output EDM file and can be browsed resp. analyzed.

Switching off the trigger matching

Again, it is easiest to have the steering of the embedding of trigger matches into PAT layer 1 objects centralized as described here.

Switching off particular matches

1. Customize the PAT layer 1 parameters in such a way, that only desired trigger matches are embedded in the PAT objects.
   This step ensures, that your output contains only desired trigger matches.
2. In order to save computation time, also the production of trigger match information in PAT layer 0 should be customized. The most efficient way to do this is to modify the sequences in **`CMS.PhysicsTools/PatAlgos/python/triggerLayer0/trigMatchSequences_cff.py`** in such a way, that only the desired trigger matches are produced.

Switching off completely

1. Set **all** parameters **`addTrigMatch`** of the PAT layer 1 producer modules to **`False`**.
2. Substitute sequence
   1. **`patLayer0`** by **`patLayer0_withoutTrigMatch`** resp.
   2. **`patLayer0_withoutPFTau`** by **`patLayer0_withoutPFTau_withoutTrigMatch`**.

Analyzing trigger matches

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

Get trigger matches from the PAT object

All PAT objects are derived from the class template `pat::PATObject`, which contains a data member

```
std::vector<pat::TriggerPrimitive> triggerMatches_;
```

that holds all embedded trigger matches of an object. The `pat::PATObject` provides access to the trigger matches by the following public methods:

- `const std::vector<pat::TriggerPrimitive> & triggerMatches() const;`
  - provides a reference to the immutable data member itself;
- `const std::vector<pat::TriggerPrimitive> triggerMatchesByFilter(const std::string & aFilt) const;`
  - provides a newly built vector of trigger objects used in a certain filter as defined here;
  - useful to distinguish matches to the same PAT object;
  - output vector has size 1, if **`resolveAmbiguities`** was set to **`True`** in the corresponding matcher module.

Methods for trigger objects

The interface to the trigger objects is in **`DataFormats/PatCandidates/interface/TriggerPrimitive.h`**.

The provided methods have been explained already here.

Exercises

Descriptions of the hands-on exercises and the homework are found here.

-- VolkerAdler - 27 Jan 2009

This topic: CMSPublic > SWGuidePATMCMatchingV1
Topic revision: r22 - 2010-05-03 - RogerWolf