Table of Contents

4.2.2 Physics Analysis Toolkit (PAT): Workflow

Detailed Review status

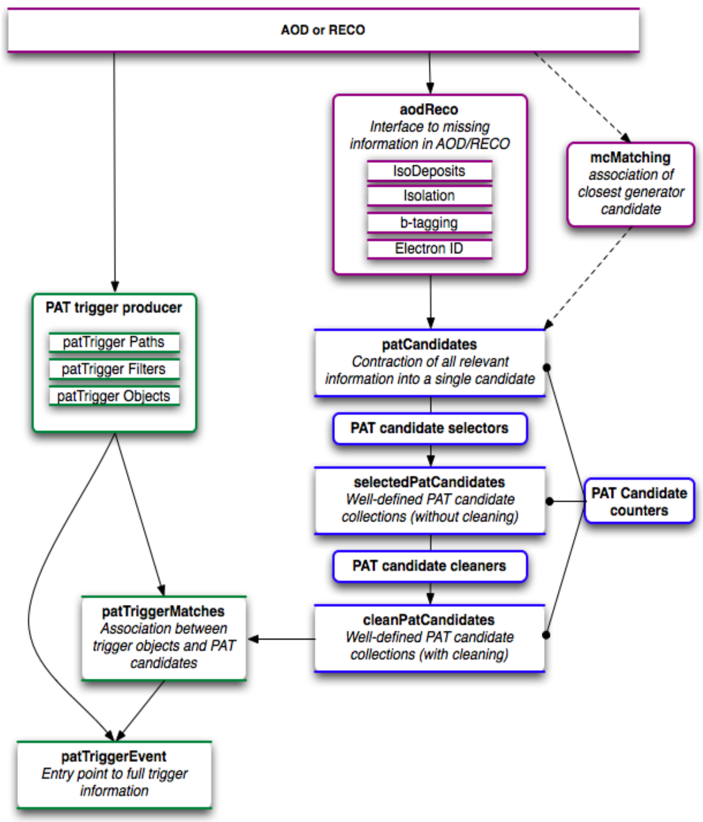
Contents

Introduction

In this section you will find a description of the standard workflow of PAT. The creation of pat::Candidates starts from the AOD (or RECO) tier. The PAT Workflow **(1)** is organised in a main production sequence (called `patDefaultSequence`) and a parallel sequence for associating trigger information. An overview of the sequences is given in the figure below. The production of a standard pat::Tuple **(1)** consists of the following phases:

- Pre-Creation: this phase includes steps, which can or have to be performed before the creation of pat::Candidates starts.
- Candidate Creation: in this phase all extra information will be contracted into the pat::Candidates.
- Candidate Selection: in this phase an extra selection step on the newly created pat::Candidate collections is supported. By default no selection is applied though.
- Candidate Disambiguation: in this phase fully configurable extra information is added to support the disambiguation of reconstructed objects.
- PAT Trigger Event: in this phase all trigger information available in the AOD (RECO) event content is re-keyed and made available to the end-user.

(1) To learn more about regularly used expressions (like pat::Candidate, pat::Tuple, ...) have a look to the WorkBookPATGlossary.


AOD or RECO
aodReco
Interface to missing information in AOD/RECO
IsoDeposits
Isolation
b-tagging
Electron ID
mcMatching
association of closest generator candidate
PAT trigger producer
patTrigger Paths
patTrigger Filters
patTrigger Objects
patCandidates
Contraction of all relevant information into a single candidate
PAT candidate selectors
selectedPatCandidates
Well-defined PAT candidate collections (without cleaning)
PAT Candidate counters
PAT candidate cleaners
patTriggerMatches
Association between trigger objects and PAT candidates
cleanPatCandidates
Well-defined PAT candidate collections (with cleaning)
patTriggerEvent
Entry point to full trigger information

The figure is *clickable*. By clicking on the collection of interest you will be lead to the corresponding description in the text.

You can browse the workflow for the creation of pat::Candidates exploiting the *edmConfigEditor* as explained in WorkBookConfigEditor. The result of the pat::Candidate creation is a pat::Tuple, which in the default configuration contains all supported pat::Candidate collections and may be used transiently (i.e. on the fly) or be made persistent at your home institute. Have a look at WorkBookPATDataFormats to learn more about the data format and content of each corresponding pat::Candidate. The pat::Tuple should **NOT** be viewed as a replacement of the AOD (RECO) event content but rather as a replacement of the flat user ntuple, which is commonly used in analyses. The Analysis Tool (AT) group of CMS strongly recommends the use of pat::Tuples instead of flat ntuples due to the following reasons:

- **Performance**: pat::Tuples and ntuples will have the same I/O performance, so there is no I/O advantage of a flat ntuple over a pat::Tuple. Additionally, the error-prone programming overhead of defining and addressing root branches drops out when using pat::Tuples.

- **Compliance**: The pat::Tuple is fully compliant with CMSSW. You can further process it with CMSSW (full framework) on a batch system or via crab, FWLite executables, or bare Root macros—however you like. The pat::Tuple can have a flexible event content that may range between the size of the AOD format (of ~100 kB/evt) and an extremely slim ntuple (of ~5 kB/evt). This change can be reached just by python configuration. The PAT team supports you with tools, sensible defaults and experience.

- **Provenance**: The pat::Tuple maintains full event provenance. Even if very flexible in its content the objects are always well defined by the configuration file they have been created with and a manageable number of configuration parameters. It is therefore very easy to interchange pat::Tuples between analysis groups for cross checks. You can still transform the pat::Tuple into a flat ntuple at a later point in your analysis if you really think that you will gain from that. You can use tools provided by the Analysis Tools (AT) group for that. The PAT team is also working on an improved support to further slim down a pat::Tuple after it has been created.

- **Support**: PAT canalises the finest certified knowledge of all POG's. With the standard pat::Tuple you start from a sensible default, that you can easily use for first iterations to define the event content you really need for your specific analysis and PAT supports the coherent reduction of information with many features and tools. The PAT support team organises bi-monthly tutorials of which any second one is a full week in-person tutorial. You will profit from a wealth of well maintained documentation TWikis and the support of the PAT team.

Note:

Several tools exist to facilitate the customisation and adaptation of the PAT workflow for the creation and the content of a pat::Tuple to the analyst's needs. To learn more about these tools have a look to SWGuidePATTools. All these tools may also be applied via pull down menus in the *edmConfigEditor*. When made persistent a reasonably configured pat::Tuple has a size between `6kB/evt` and `50kB/evt` with the largest space consumption usually associated to the pat::Jet collections. In the current default configuration of the pat::Tuple has a size of `~15kB/evt`. To learn more about the space consumption of a pat::Tuple in the current default configuration have a look here. Note that these size estimates have been performed on 100 events of an inclusive pythia ttbar sample in CMSW_3_5_7. To learn more about size estimates of pat::Tuples have a look at SWGuidePATEventSize.

Pre-Creation

The main sequence starts with a pre-creation phase of the AOD or RECO event contents if necessary(!) and the Monte Carlo matching, which has to be provided before being folded into the pat::Candidates. During the pre-creation phase useful information or transient data types may be created (if not yet available) which include the following information:

- A set of isolation variables using information from several detectors, which are non-standard and therefore not in the AOD (RECO) event content.
- The association of non-standard POG supported object identification variables, which are not yet part of the AOD (RECO) event content definition.
- A set of tau discriminator variables, which are just too numerous and analysis specific to be all part of the AOD (RECO) event content.
- A compilation of JEC factors for jets, which are retrieved from the event setup and prepared to be folded into the pat::Jet.
- A set of b-discriminator values, which might not be all available in the AOD (RECO) event content.
- The association of charged tracks.
- The corresponding jet-charge.

The complete information is provided in *cff* files, which collect all objects needed to create a certain pat::Candidate collection as defined in the *cfi* file that contains the module definition of the corresponding pat::Candidate EDProducer. You can find the corresponding *cff* and *cfi* files in the producersLayer1 directory of the PatAlgos package. You can find the corresponding interfaces to the AOD (RECO) event content in the recoLayer0 directory of the PatAlgos package. They may change from CMSSW release to CMSSW release as parts of the objects created during this pre-creation phase might migrate into later definitions of the AOD (RECO) event content. Other parts might need to be added when running on input files, which had been produced at times when the release that you are using did not yet exist.

Note:

You can view the role of this pre-creation phase as an interface between, what is naturally available on the AOD (RECO) of your input files (at the time the files were produced) and the full feature catalogue of your current CMSSW release you want to make use of for your analysis and which might not be in sync with the input files you are planning to use.

Note:

Another important step is the matching to Monte Carlo truth information, which is only applicable to simulated events. Many options of matching exist, which are based on the objects similarity on their spatial and kinematic distributions. To find out more about these options have a look to SWGuidePATMCMatching. The corresponding module definitions as used in the standard configuration of PAT can be found in the mcMatchLayer0 directory of the PatAlgos package. This step is part of the standard configuration of PAT. It can easily be switched off by the use of pat::Tools as described on SWGuidePATTools.

Candidate Creation

After the pre-creation phase all extra information, which has been translated into a proper format is collapsed into the pat::Candidates. In the default configuration the following collections are produced during this step:

- **patPhotons**
- **patElectrons**
- **patMuons**
- **patTaus**
- **patJets**
- **patMET**

You can find the corresponding module definitions in the producersLayer1 directory of the PatAlgos package. The production of all supported pat::Candidate collections is steered by the sequence `patCandidates` in the patCandidates_cff.py file. These collections are the inputs for the further operations of candidate selection and candidate disambiguation. In the default configuration of the PAT workflow they will not be made persistent and should therefore be viewed as transient object collections. To learn more about how to change the configuration of the content of a single pat::Candidate have a look to WorkBookPATConfiguration.

Candidate Selection

PAT supports intuitive candidate selection via the SWGuidePhysicsCutParser. With the help of this tool you may apply selections on any available member function of a given object via intuitive selection strings (e.g. "pt>30 & abs(eta)<3"). This will provide you with a new collection of objects that passed the applied selection. During the phase of pat::Candidate selection such candidate selections are applied to the pat::Candidate collections, resulting in the following new collections, which form the first potentially persistent layer of pat::Candidates in the default configuration of the PAT workflow:

- **selectedPatPhotons**
- **selectedPatElectrons**
- **selectedPatMuons**
- **selectedPatTaus**
- **selectedPatJets**
- **patMET**

You can find the corresponding *cfi* files in the selectionLayer1 directory of the PatAlgos package. The production of all supported selected pat::Candidate collections is steered by the sequence `selectedPatCandidates` in the selectedPatCandidates_cff.py file. The selection modules can have any pat::Candidate collection as input, so you can also refine a collection with a selectedPatCandidate collection as input, which will provide you with an additional collection of the candidates that passed this refined selection.

⚠ **Note:**

In the default configuration of the PAT workflow all selection strings are empty: **NO SELECTION IS APPLIED**. So you will get object collections, which apart from their module label are equivalent to the previously produced pat::Candidate collections. Note that physically there is no selection on MET. Therefore there is no selection module provided for MET. (This might change in the future.) You might want to apply any object or event selection that might be suitable for your analysis already during pat::Tuple production or at any later step of your analysis by replacing the dummy selections in the corresponding configuration files or corresponding clones of them. Note that this is always possible with any input collection of pat::Candidate type even after the pat::Tuple has been made persistent.

⚠ **Note:**

The selection modules only act on object collections. To apply an event selection based on a collection of selected objects you might want to make use of the *candidateCountFilter* modules provided in the same directory of the package. Have a look to WorkBookPATConfiguration to find out more about the configuration of *selectedPatCandidate* collections.

Candidate Disambiguation

Due to the way objects are reconstructed in CMS some of the physics measurements, like energy deposits in the calorimeter might be reconstructed several times as different physics objects. For example, an cluster in the electromagnetic calorimeter can be interpreted as a photon, electron or jet. It might therefore be present in several different candidate collections at the same time. Depending on the analysis purpose, this might be of interest, harmless, or harmful. For more complex analyses, which aim at the interpretation of a whole event containing candidates of several types a disambiguation step has to take place. Removing objects from collections should remain a very analysis dependent and high level analysis step though, which has to be performed with care and viewed with caution. It is clear that a common frame as a well defined common basis of object disambiguation, which might also combine the finest knowledge and expertise of all analysis groups within CMS is appreciable.

PAT supports such a common frame of object disambiguation in a user configurable and well defined way. In the default configuration of the PAT workflow no objects are removed from the collections, but overlapping objects from other collections are marked by edm::Ptrs. These pointers are stored in a new collection of pat::Candidates. In the default configuration of the PAT workflow the collections are:

- **cleanPatPhotons**
- **cleanPatElectrons**
- **cleanPatMuons**
- **cleanPatTaus**
- **cleanPatJets**
- **patMET**

You can find the corresponding *cfi* files in the cleaningLayer1 directory of the PatAlgos package. The production of all supported clean pat::Candidate collections is steered by the sequence `cleanPatCandidates` in the cleanPatCandidates_cff.py file. In later analysis steps the stored edm::Ptrs can be used to make an assessment on the overlapping candidate(s) of other collections. It can still be dropped from further considerations, or its energy can be used to modify the energy of the object of interest as for the example of a jet and an overlapping electron.

⚠ **Note:**

PAT supports selection and preselection steps for the overlap checking and overlap checking based on the distance of deltaR or common seeds to super clusters as used for the disambiguation of electrons and photons. Note that in the default configuration of the PAT workflow **NO OBJECT IS REMOVED** from the pat::Candidate collections. Only the extra information of overlapping objects from other collections is added to the pat::Candidates. Also note that physically there is no cleaning on MET. Therefore there is no cleaning module provided for MET.

In the default configuration of the PAT worflow overlap checking is performed in the following way:

- *cleanPatMuons*: are taken as *selectedPatMuons* (no overlap checking).
- *cleanPatElectrons*: are checked for overlaps with *cleanPatMuons*.
- *cleanPatPhotons*: are checked for overlaps with *cleanPatElectrons*.
- *cleanPatTaus*: are checked for overlaps with *cleanPatMuons* and *cleanPatElectrons*.
- *cleanPatJets*: are checked for overlaps with *cleanPatMuons*, *cleanPatPhotons*, *cleanPatElectrons*, *cleanPatTaus* and isolated *cleanPatElectrons*.

The latter overlap check on isolated *cleanPatElectrons* was added as an example of a preselection on overlapping objects.

Note:

The stored edm::Ptrs point to the object in the other candidate collection. It is therefore e.g. easily possible at any later analysis stages to check whether an electron, that is overlapping with a jet itself has an overlap with a photon, aso. All flexibility to judge is therefore still in the hands of the analyst. The clean pat::Candidate collections should be viewed as the default output of the standard pat::Tuple. If you already know that you won't need any object disambiguation it can be completely removed from the default PAT workflow though by the use of PAT tools. You can view this step of pat::Candidate processing **OPTIONAL**.

Note:

It is always possible to add object disambiguation information with any input collection of pat::Candidate type even after the pat::Tuple has been made persistent. So you have the option to create a persistent pat::Tuple at your institute on which you apply a user-defined object disambiguation afterwards on the fly at later point in your analysis. To learn more about the way PAT supports object disambiguation have a look to SWGuidePATCrossCleaning. To learn more about the tools to configure the default PAT workflow have a look to SWGuidePATWorkflow.

PAT Trigger Event

Besides the main PAT production sequence, trigger information is re-keyed into a human readable form in the pat::TriggerEvent and the PAT trigger matching provides the opportunity to connect PAT objects with trigger objects for further studies. Thus the user can easily figure out exactly which object(s) fired a given trigger. The production sequence starts from the PAT trigger producers, and folds the information in the pat::TriggerObject, pat::TriggerFilter, and pat::TriggerPath classes. These classes are contracted into the pat::TriggerEvent, which should be viewed as the central entry point to all trigger information. A detailed description can be found at SWGuidePATTrigger. The association of trigger objects and the pat::Candidates (produced in the main sequence) is provided by the pat::TriggerMatching, and is described in SWGuidePATTrigger#PATTriggerMatcher. A detailed description of the pat::TriggerEvent can be found at SWGuidePATTrigger#TriggerEvent.

Review status

Show ▸ Hide ▾

Reviewer/Editor and Date (copy from screen)	Comments
RogerWolf - 26 Aug 2010	Rework for Sept PAT Tutorial

Responsible: RogerWolf

This topic: CMSPublic > WorkBookPATWorkflow
Topic revision: r34 - 2014-06-23 - ElliotHughes