Table of Contents

4.7 MiniAOD Analysis Documentation

This page documents the MiniAOD data tier as it is implemented from Run 2 in 2015 and early 2016 (CMSSW release 7_4_1 or later); The documentation for the 2014 version used in CSA14 and Phys14 is in WorkBookMiniAOD2014 Four versions versions of 2015 MiniAODs exist:

- **74X version 1** corresponds to `CMSSW_7_4_10` or earlier is what produced in the RunIISpring15DR74 MC, Run2015B and Run2015C prompt reco, and 17Jul2015 re-miniAOD (and also the few datasets produced in `CMSSW_7_5_1`)
- **74X version 2** corresponds to `CMSSW_7_4_12` or later, corresponding to Run2015D prompt reco and the re-miniAOD or later re-recos of data. This was a major update including new e/ energy scales and identification algorithms, new jet calibrations, met filters, and an updated tau reconstruction
- **76X version 1** corresponds to `CMSSW_7_6_1`, corresponding to the first 76X MC campaing **`Fall15MiniAODv1`**. This version includes only a few small improvements wrt 74X version 2, which will be marked with UPDATED below. There has been no change in the high level calibrations for jets and e/ , but the underlying calibration and reconstruction has been updated.
- **76X version 2** corresponds to `CMSSW_7_6_3`, corresponding to the 76X 2015 Data re-reco **`16Dec2015`** and re-miniAOD **`Fall15MiniAODv2`** . Bugfix production wrt 76X version 1, fixing tau reconstruction, updating some MET filters, adding a few b-tagging discriminators

See WorkBookMiniAOD for a more comprehensive list of different versions and associated campaigns and CMSSW releases.

- `CMSSW_7_4_X` must be used on **74X** miniAODs, and **`CMSSW_7_6_X`** on **76X** MiniAODs

Introduction

The MiniAOD format

The MiniAOD is a new high-level data tier introduced in Spring 2014 to serve the needs of the mainstream physics analyses while keeping a small event size (30-50 kb/event).

The main contents of the MiniAOD are:

- High level physics objects (leptons, photons, jets, E_T^{miss}), with detailed information in order to allow e.g. retuning of identification criteria, saved using PAT dataformats.
  Some preselection requirements are applied on the objects, and objects failing these requirements are either not stored or stored only with a more limited set of information.
  Some high level corrections are applied: L1+L2+L3(+residual) corrections to jets, type1 corrections to E_T^{miss}.
- The full list of particles reconstructed by the ParticleFlow, though only storing the most basic quantities for each object (4-vector, impact parameter, pdg id, some quality flags), and with reduced numerical precision; these are useful to recompute isolation, or to perform jet substructure studies. For charged particles with $p_T > 0.9$ GeV, more information about the associated track is saved, including the covariance matrix, so that they can be used for b-tagging purposes.
- MC Truth information: a subset of the genParticles enough to describe the hard scattering process, jet flavour information, and final state leptons and photons; GenJets with $p_T > 8$ GeV are also stored, and so are the other mc summary information (e.g event weight, LHE header, PDF, PU information).
  In addition, all the stable genParticles with mc status code 1 are also saved, to allow reclustering of GenJets with different algorithms and substructure studies.
- Trigger information: MiniAOD contains the trigger bits associated to all paths, and all the trigger objects that have contributed to firing at least one filter within the trigger. In addition, we store all objects reconstructed at L1 and the L1 global trigger summary, and the prescale values of all the

triggers.

Structure of this documentation

This documentation will focus on analyzing MiniAOD files, since the production of MiniAODs is normally done centrally (but in the last part some instructions will be given also on how to produce MiniAOD files privately). The examples will be given both in the context of full CMSSW running using EDAnalyzers and in the context of FWLite with python whenever applicable. FWLite in C++ is of course also supported, but it's up to the reader to guess what the code should be starting from the two examples.

The color scheme of the twikipage is as follows:

- Shell commands will be embedded in grey box, e.g.

```
cmsrel CMSSW_7_4_1
```

- Python code will be embedded in light blue box, e.g.

```
from DataFormats.FWLite import Handle, Events
```

- C++ code will be embedded in light green box, e.g.

```
iEvent.getByToken(muons_, muons);
```

- Example output will be in a yellow box, e.g.

```
boh?
```

Setting up your environment

The documentation will use the **`CMSSW_7_4_1`** release and assume you're working on a SLC6 node (e.g. **`lxplus6.cern.ch`**) with **`SCRAM_ARCH`** set to `slc6_amd64_gcc491`. To setup the release, you can do

```
cmsrel CMSSW_7_4_1
cd CMSSW_7_4_1/src
cmsenv
```

As input test files in the examples we will use
`/store/relval/CMSSW_7_4_1/RelValTTbar_13/MINIAODSIM/PU25ns_MCRUN2_74_V9_gensim71X-v1/00000/72C84B`
which contains 6k events of ttbar production, but any other MINIAODSIM file will work equally well.

Skeleton for CMSSW Analysis

To create a skeleton of c++ analyzer, you can do the following

```
mkdir Test
cd Test
mkedanlzr MiniAnalyzer

New package "MiniAnalyzer" of EDAnalyzer type is successfully generated
MiniAnalyzer/
|  plugins/
|  |-- BuildFile.xml
|  |-- MiniAnalyzer.cc
|  python/
|  |-- CfiFile_cfi.py
|  |-- ConfFile_cfg.py
|  test/
|  doc/
```

```
Total: 4 directories, 4 files
```

Then you'll need to edit the **`BuildFile.xml`** to add a **`<use name="DataFormats/PatCandidates"/>`**

Skeleton for Python FWLite Analysis

```
# import ROOT in batch mode
import sys
oldargv = sys.argv[:]
sys.argv = [ '-b-' ]
import ROOT
ROOT.gROOT.SetBatch(True)
sys.argv = oldargv

# load FWLite C++ libraries
ROOT.gSystem.Load("libFWCoreFWLite.so");
ROOT.gSystem.Load("libDataFormatsFWLite.so");
ROOT.AutoLibraryLoader.enable()

# load FWlite python libraries
from DataFormats.FWLite import Handle, Events

# open file (you can use 'edmFileUtil -d /store/whatever.root' to get the physical file name)
events = Events("root://eoscms//eos/cms/store/relval/CMSSW_7_4_1/RelValTTbar_13/MINIAODSIM/PU25ns

for i,event in enumerate(events):
    print "\nEvent", i
    if i > 10: break
```

Analyzing MiniAOD

High level physics objects

High level physics objects in miniAOD are saved in the PAT dataformats, which in turn derive from the RECO/AOD formats, so almost everything that you can find in the WorkBook documentation on PAT and physics objects applies to them direcly.

Object	Label	C++ class	Selection	
Muons	**`slimmedMuons`**	**`std::vector<pat::Muon>`**	includes `muons` with p_T > 5 GeV or that pass the PF muon ID (see below)	All s isola outer `muon` `tune`
Electrons	**`slimmedElectrons`**	**`std::vector<pat::Electron>`**	all `gedGsfElectrons` electrons	For e infor (supe intere id va For e only seed(id an zeroe
Taus	**`slimmedTaus`**	**`std::vector<pat::Tau>`**	taus from `hpsPFTauProducer` with p_T > 18 GeV, and passing the basic `decayModeFindingNewDMs`	All P discr to the provi **Note**

			id	**versi reco the T**
Photons	**slimmedPhotons**	**std::vector<pat::Photon>**	`gedPhotons` with $p_T > 14$ GeV and `hadTowOverEm() < 0.15` Since 74X In version 2 MiniAOD, keep also pt 10-14 GeV if `chargedHadronIso() < 10`	For p r9 or infor (supe intere id va The r of the `char` `char`
Jets	**slimmedJets**	**std::vector<pat::Jet>**	`ak4PFJetsCHS` with $p_T >$ 10 GeV	L1+L are a jet id Link: const
	slimmedJetsPuppi	**std::vector<pat::Jet>**	puppi jets with $p_T > 20$ GeV	L1+L are a infor are p candi Since the P the tu 2015
	slimmedGenJets	**std::vector<reco::GenJet>**	`ak4GenJets` with $p_T > 8$ GeV	links parti
	slimmedJetsAK8	**std::vector<pat::Jet>**	`ak8PFJetsCHS` with $p_T >$ 200 GeV (for substructure) Since 74X version 2 miniAODs, the p_T threshold is relaxed to 170 GeV	L1+L are a corre preco provi const them
	slimmedJetsAK8PFCHSSoftDropPacked	**std::vector<pat::Jet>**	Subjets of soft drop algorithm	L1+L are a corre infor are p candi
	slimmedJetsCMSTopTagCHSPacked	**std::vector<pat::Jet>**	Subjets of CMS top tagger algorithm	L1+L are a corre infor are p candi
MET	**slimmedMETs**	**std::vector<pat::MET>**	the type1 PF MET	MET Cont gen r Since energ corre

				ak4P follo JetM Since have uncer C++
	slimmedMETsNoHF	**std::vector<pat::MET>**	as above but excluding HF; available **only** in 74X version 2 miniAODs	
	slimmedMETsPuppi	**std::vector<pat::MET>**	puppi-corrected MET	MET PFCa with

Muons

Muons from the AOD `muons` collection are included in MiniAOD if they pass at least one of these three requirements:

- $p_T > 5$ GeV
- $p_T > 3$ GeV and any of the following IDs: PF, global, arbitrated tracker, standalone, RPCMuLoose
- any p_T, if they pass the PF ID.

Muon id information is available and accessible as usual through the **`isLooseMuon`**, **`isTightMuon`**, **`isSoftMuon`**, **`isHighPtMuon`** and **`muonID`** methods as provided in the **`pat::Muon`** class.

The following tracks are embedded by value in the **`pat::Muon`**, if available: tracker, outer and global tracks, `muonBestTrack` and `typePMuonBestTrack`. Compared to CSA14 and Phys14, now DYT and TeV-muon refits are also available for muons with pt > 100 GeV

The ECal energy associated to the PF Muon is also stored, accessible as `muon.pfEcalEnergy()` . This has been used in some analyses, e.g. H ZZ 4l, for collinear FSR recovery.

Electrons

All `gedGsfElectrons` electrons are saved.

For electrons of $p_T > 5$ GeV, detailed information is saved:

- Supercluster, basic clusters and seed clusters, accessible through the **`reco::GsfElectron`** information (though they're stored externally in the **`reducedEgamma`** collections). The final superclusters and clusters are saved, not also the intermediate unrefined "PF" superclusters and clusters.
- Interesting rechits, saved in **`reducedEGamma:reducedEBRecHits`**, **`reducedEGamma:reducedEERecHits`**, **`reducedEGamma:reducedESRecHits`** for barrel, endcap and pre-showers respectively.
- Shower shape variables computed from the standard superclusters are provided in the basic `reco::GsfElectron` interface, e.g. **`sigmaIetaIeta()`** (the only missing one in AOD is **`sigmaIetaIphi()`** which is added in **`pat::Electron`** interface).
  In addition, for backwards compatibility MiniAOD also includes the shower shape variables computed using the full 5x5 superclusters like in Run 1, which are accessible through the `pat::Electron` methods, e.g. `full5x5_sigmaIetaIeta()`.
- Isolation computed from PF Clusters, similar to what done in the HLT, is also available through the methods `ele.ecalPFClusterIso()` and `ele.hcalPFClusterIso()`

For electrons that don't satisfy this cut, only the supercluster and seed clusters are provided.

The following features have been updated in 74X version 2 miniAODs:

- The electron energy and energy error is updated with a new regression trained on spring15 MC
- The set of embedded electron IDs has been updated to contain: cut-based `Spring15_25ns_V1` `Spring15_50ns_V1` (and `PHYS14_PU20bx25_V2`), `heepElectronID_HEEPV60`, `mvaElectronID_Spring15_25ns_nonTrig_V1`
  The electron IDs can be accessed via the electronID methods of the `pat::Electron` object. The output discriminator for the non-triggering MVA id is accessible as `userFloat("ElectronMVAEstimatorRun2Spring15NonTrig25nsV1Values")`

Photons

For photons, the source collection is the `gedPhotons`, with two levels of selection:

- A basic selection defining which objects are kept: **p_T > 14 GeV and `hadTowOverEm() < 0.15`**
  Since 74X version 2 miniAODs, photons with p_T 10-14 GeV are also lept, if they satisfy `hadTowOverEm() < 0.15 && chargedHadronIso()<10`
- A tighter selection defining for which objects the detailed information is kept: it's the logical or of a R9 requirement **`r9()>0.8`** , and an absolute and relative isolation cuts done with charged hadrons only, **`chargedHadronIso()<20`**, **`chargedHadronIso()<0.3*pt()`**

Similarly to electrons, the basic information is only the supercluster and seed clusters, while the detailed information includes basic clusters and interesting rechits.

Note that electrons and photons share a common collection of interesting clusters and rechits, and so if the same supercluster gives rise to both an electron and a photon the two high-level physics objects will really reference to the same object (as it happens in AOD). For more technical details, refer to reducedEgamma_cfi.py

The following features have been updated in 74X version 2 miniAODs:

- The photon p_T preselection has been updated, as written above
- The photon energy and energy error is updated with a new regression trained on spring15 MC
- The photon isolation variables have been updated with better vetos (%ICON{todo}% can somebody explain better the change?)
- The set of embedded electron IDs has been updated to contain: cut-based `Spring15_50ns_V1` and `PHYS14_PU20bx25_V2`, MVA `Spring15_25ns_nonTrig_V2` and `Spring15_50ns_nonTrig_V2`
  %ICON{todo}%Put information on how to access them

Taus

For taus, MiniAOD includes all objects with p_T > 18 GeV and passing the `decayModeFindingNewDMs` discriminator (which is strictly looser than the `decayModeFinding` used up to 72X). Tighter tau ID requirements can be applied using the **`tauID(name)`** method, all POG-supported discriminators are included. The tau ID discriminators recommended for 13 TeV are described on this wiki.

Tau reconstruction depends on the miniAOD version:

- 74X version 2 introduced the dynamic strip algorithm, and an updated set of discriminators
- 76X version 1 introduced further features, but unfortunately also some bugs in the decay more reconstruction causing a loss of efficiency
- 76X version 2 fixes the bugs in introduced in 76X version 1 decay modes, and updates the

anti-electron MVA and the mass assignment (see slides)

Additional information on the taus, e.g. the impact parameter of the reconstructed tau vertex, are saved in the TauPFEssential format which is embedded in the pat::Tau object.

Links to the packed PF candidates corresponding to the PFCandidates used to reconstruct a given tau are accessible through the methods like `leadChargedHadrCand()`, `leadNeutralCand()`, `leadCand()`, `signalCands()`, `isolationCands()`, ...
Note instead that the older methods with similar name but with a `PF` inside, like `signalPFCands()`, are not usable on MiniAOD since they're not compatible with the packed PF candidate format used in MiniAOD.

Jets

Several collections of jets are saved:

- one general purpose collection of **AK4** jets **`slimmedJets`**, made from **`ak4PFJetsCHS`**, i.e. using anti-k_T clustering out of Particle Flow candidates, after rejecting the charged hadrons that are associated to a pileup primary vertex. These have standard jet energy corrections applied (L1FastJet, L2, L3), and a p_T cut at 10 GeV
- one general purpose collection of **AK4** jets with the PUPPI algorithm applied **`slimmedJetsPuppi`**; they have the same clustering as the `slimmedJets`, but pileup mitigation is performed by weighting the particles according to the PUPPI algorithm, i.e. the jet 4-vector is the weighted sum of the particle 4-vectors.
  - note that the clustering is performed with the weighted particles, which can lead to a different set of jets in some events
  - these jets have L1, L2 and L3 jet energy corrections. n.b. L1 corrections are small, since the PUPPI procedure corrects the response for the pileup already.
    These jets have a p_T cut at 20 GeV
- one collection for dedicated studies of substructure, **`slimmedJetsAK8`** , made from **`ak8PFJetsCHS`**, i.e. the same clustering and pile-up subtraction of the `slimmedJets` but with a larger distance parameter R=0.8 instead of R=0.4 (R corresponds *approximately* to the radius of the jet)
  These jets have a p_T cut at 200 GeV, matching the threshold at which jet merging starts to be relevant
  - Since 74X version 1 MiniAOD version 2, the cut has been relaxed to 170 GeV
- two subjet collections, "W-like" **`slimmedJetsAK8PFCHSSoftDropPacked`** and "top-like" **`slimmedJetsCMSTopTagCHSPacked`**, for the **`slimmedJetsAK8`**, ad described below.

For the AK4 jet collections, links to the daughters are available through the `daughter(idx)` and `numberOfDaughters()` methods, and they will point into the collection of packed PF candidates described below.
It should be noted that the `getPFConstituent` and `getPFConstituents` methods instead will NOT work, since they explicitly require the daughter to be a full **`reco::PFCandidate`** and not a packed PF one.

For the AK8 jet collection, the links to the daughters are stored in an efficient way, but it is not the same way to access the daughters as in the past in `pat::Jet`. The jets have (by default) two substructure algorithms run, Soft Drop and the CMS Top Tagger. Both of these have subjets, available through the `subjets` method in `pat::Jet`. However, each also point to their respective constituents. **To save space, the Soft Drop Subjets are stored in positions 0 and 1 in the constituent list**. These subjets contain pointers to their own constituents. The remaining constituents that are groomed away by the soft drop method are stored in the constituent vector in indices 2,3,....N. As such, to access all daughters, you must loop over the constituents of all of the constituents (see code snippet below). As with the AK4 collections, the `daughter(idx)` method will be the appropriate one, while `getPFConstituent` and `getPFConstituents` will also NOT work.

For the **AK4 jets** (both with and without PUPPI), the following information is provided:

- b-tagging discriminators, available from the `bDiscriminator(name)` method, for: `combinedSecondaryVertexBJetTags` , `pfJetBProbabilityBJetTags` , `pfJetProbabilityBJetTags` , `pfTrackCountingHighPurBJetTags` , `pfTrackCountingHighEffBJetTags` , `pfSimpleSecondaryVertexHighEffBJetTags` , `pfSimpleSecondaryVertexHighPurBJetTags` , `pfCombinedSecondaryVertexV2BJetTags` , `pfCombinedInclusiveSecondaryVertexV2BJetTags` , `pfCombinedSecondaryVertexSoftLeptonBJetTags` , `pfCombinedMVABJetTags` ;
  - If you were using `combinedInclusiveSecondaryVertexV2BJetTags` in 72X (the "new" version of CSV), then you should move to **`pfCombinedInclusiveSecondaryVertexV2BJetTags`** in 74X which basically the same algorithm, but based on PF candidates instead of tracks. The same is true for other discriminators as well (e.g. `pfTrackCountingHighEffBJetTags` is the new equivalent of the old `trackCountingHighEffBJetTags`).
    See the b-tagging section later in this page for more information.
  - In 74X miniAOD version 2, some updates to the b-tagging discriminators have been deployed, though very small changes are expected for the standard **`pfCombinedInclusiveSecondaryVertexV2BJetTags`**
  - In 76X miniAOD version 2, introduces some c-tagging and double b-tagging discriminators UPDATED
- additional variables related to the associated secondary vertex if there is one, stored as `userFloats`: the mass of the vertex (`vtxMass`), the number of tracks in it (`vtxNtracks`), and the decay length value and significance (`vtx3DVal`, `vtx3DSig`)
- for standard jets (not PUPPI), the discriminator for the MVA PileUp id, saved as `userFloat("pileupJetId:fullDiscriminant")`.
  - It should be noted that the training used in version 1 miniAODs is for `ak5PFJetsCHS` in CMSSW 5.3.X and Run 1 pileup, so it's probably not optimal for Run 2.
  - In 74X version 2 miniAODs, the training has been updated to 74X, but the performance is still poor outside the coverage of the tracker
- For standard jets (not PUPPI), the pt and emEnergyFraction=(1-hadEnergyFraction) of the matched (dR<0.4) CaloJet are saved as `userFloat("caloJetMap:pt")` and `userFloat("caloJetMap:emEnergyFraction")`.

For **AK8 jets**, the following information is included:

- pruned, trimmed and filtered masses, available as `userFloats` with labels `ak8PFJetsCHSSoftDropMass`, `ak8PFJetsCHSPrunedMass`, `ak8PFJetsCHSTrimmedMass`, `ak8PFJetsCHSFilteredMass`
- n-subjettiness variables tau1, tau2, and tau3, available as `userFloats` with labels `NjettinessAK8:tau1`, `NjettinessAK8:tau2`, and `NjettinessAK8:tau3`
- the CMS top tagger reclusters the AK8 constituents with the Cambridge-Aachen algorithm with D=0.8 (CA8). It outputs between one and four subjets corresponding to the decay products of the top quark sequential decay (quarks from W decay, and the b quark). This information is available as `tagInfo("caTop").properties().topMass`, `tagInfo("caTop").properties().minMass`, `tagInfo("caTop").properties().nSubJets`
- **NEW for 74x** : the soft drop tagger is now added.
- **NEW for 74x** : the subjets for the soft drop and CMS top taggers are added. To access them use the `subjets` method within `pat::Jet` with labels `SoftDrop` or `CMSTopTag`.
- The documentation to use `userFloats` is highlighted here. A simple snippet to perform a very simple W/Z/H tagging selection would be :

Snippet to perform simple W/Z/H tagging Hide example CMSSW code

```
double tau1 = jet.userFloat("NjettinessAK8:tau1");     //
double tau2 = jet.userFloat("NjettinessAK8:tau2");     //  Access the n-subjettiness variables
double tau3 = jet.userFloat("NjettinessAK8:tau3");     //

double softdrop_mass = jet.userFloat("ak8PFJetsCHSSoftDropMass"); // access to filtered mass
double trimmed_mass = jet.userFloat("ak8PFJetsCHSTrimmedMass");   // access to trimmed mass
```

```
double pruned_mass = jet.userFloat("ak8PFJetsCHSPrunedMass");     // access to pruned mass
double filtered_mass = jet.userFloat("ak8PFJetsCHSFilteredMass"); // access to filtered mass


bool mySimpleWTagger = (tau2/tau1) < 0.6 && softdrop_mass > 50.0;
```

- The documentation to use `TagInfos` is highlighted here. The CMS top tagger utilizes this functionality as well, although it is no longer used in Run 2. To implement a simple top tagger with one of the latest working points for 2015 data here, a simple snippet would be :

Snippet to perform simple top tagging Hide example CMSSW code

```
   double topMass = jet.userFloat("ak8PFJetsCHSSoftDropMass");
   double tau1 = jet.userFloat("NjettinessAK8:tau1");     //
   double tau2 = jet.userFloat("NjettinessAK8:tau2");     //  Access the n-subjettiness variables
   double tau3 = jet.userFloat("NjettinessAK8:tau3");     //
   double tau32 = 1.0;
   if ( tau2 > 0.0001 )
      tau32 = tau3 / tau2;

   if ( topMass > 110.0 && topMass < 210. && tau32 < 0.5 )
      topTagged = true;
```

B-tagging

To use b-tagging with MiniAOD, please, refer to the b-tagging CMS DAS school short exercise at this link. Note that the exercise is currently still being updated for the 74X release.

A short, essential, example on how to produce a jet collection and add the Run2 recommended discriminator for analyses (pfCombinedInclusiveSecondaryVertexV2BJetTags) is given below, in the Advanced topics section.

E_T^{miss}

The type1 corrected E_T^{miss} is used in the MiniAOD. Also, on MC the generated E_T^{miss} is provided. The p_T, and sumE_T of the calo E_T^{miss} and of the uncorrected E_T^{miss} are also saved within that object

Since 74X version 2 miniAODs the type1 corrections is computed from `ak4PFJetsCHS` jets with $p_T > 15$ GeV, as per the updated prescribed by the JetMET group (version 1 used `ak4PFJets` and $p_T > 10$ GeV)

The more sophisticated algorithm like MVAmet is not available at the moment. The metsignificance stored refers to the algorithm used at the beginning of run1 and the one used at the end of run1 is not available at the moment. Standalone recipes to run on top of miniAOD are needed to recalculate those.

Since 74X, in addition to the standard type-1 PF missing energy, also the E_T^{miss} reconstructed with the PUPPI pileup mitigation algorithm is also provided, in the **`slimmedMETsPuppi`** collection

- TODO document how this is made, and how to use it.

The pat::MET object also contains the information related to the type of **corrections** and **uncertainties** from the standard POG recipe of shifting and smearing objects, from the **`corXyz(level)`** methods and **`shiftedXyz(uncertainty, level)`** methods:

- `Xyz` can be any of **`P2, P3, P4, Px, Py, Pz, Phi, SumEt`**
- `uncertainty` can be any of the 19 variations considered (e.g. `UnclusteredEnUp` or `JetResDown`)
- level is `Type1` (default), `Raw`, `Type1p2`, etc; note that not all uncertainties are filled for levels.
- the standard methods `pt()`, `phi()`, `p4()` etc all return the default type1 corrected MET

You can find the list of all values in version 1 and version 2 miniAODs.

- Values of **the enumerators have changed between version 1 and version 2 of MiniAOD**. If you use CMSSW versions `7_4_12` or newer on version 1 miniAODs to access MET uncertainties (**not recommended**), you have to use the enumerator value corresponding to that uncertainty in version 1. e.g. `JetEnUp` was mapped to a value of `0` in version 1, but in newer CMSSW versions it is mapped to a value of `2`, and instead `0` is `JetResUp`; so, if you want the MET shift for a JES shift up from an old sample, you should use `METUncertainty(0)` or `JetResUp`, and not `JetEnUp`. Apologies for the confusion this may cause.

In 74X MiniAOD v2 it was also added the **`slimmedMETsNoHF`** MET computed excluding HF. This is meant to be a temporary solution while waiting for the HF calibration. The JetMET group recommendation is to use `slimmedMETs` for 25ns and `slimmedMETsNoHF` for 50ns. This MET is not available in 76X, and the recommendation is to use the `slimmedMETs` .

The **raw PF E_T^{miss}** is available, but the way to access it depends on the miniAOD version and CMSSW release:

- in `CMSSW_7_6_X` or later, on 76X vminiAODs, you can use the **`met.uncorPt()`**, **`met.uncorPhi()`** etc methods
- in `CMSSW_7_4_12` or later, on 74X version 2 miniAODs, you can use the **`met.uncorPt()`**, **`met.uncorPhi()`** etc methods
- in `CMSSW_7_4_11` or earlier, on 74X version 1 miniAODs, you can use the **`met.uncorrectedPt()`**, **`met.uncorrectedPhi()`**, **`met.uncorrectedSumEt()`** methods
- in `CMSSW_7_4_12` or later, on 74X version 1 miniAODs, things are a bit less clean due to the change of dataformat, and you have to use `met.shiftedP2_74x(12,0)` and `met.shiftedSumEt_74x(12,0)` (or, in C++, `met.shiftedP2_74x(pat::MET::METUncertainty(12),pat::MET::Raw)` etc)

The **raw calo E_T^{miss}** can be accessed wit **`met.caloMETPt()`**, **`met.caloMETPhi()`**, **`met.caloMETSumEt()`** methods.

Example code accessing all high-level physics objects TODO

TODO update with new variables and collections the C++ example. The python one is more up to date.

Show example CMSSW code Hide example CMSSW code

```
// system include files
#include <memory>

// user include files
#include "FWCore/Framework/interface/Frameworkfwd.h"
#include "FWCore/Framework/interface/EDAnalyzer.h"

#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/MakerMacros.h"

#include "FWCore/ParameterSet/interface/ParameterSet.h"

#include "DataFormats/VertexReco/interface/VertexFwd.h"
#include "DataFormats/VertexReco/interface/Vertex.h"
#include "DataFormats/PatCandidates/interface/Muon.h"
#include "DataFormats/PatCandidates/interface/Electron.h"
#include "DataFormats/PatCandidates/interface/Tau.h"
```

```
#include "DataFormats/PatCandidates/interface/Photon.h"
#include "DataFormats/PatCandidates/interface/Jet.h"
#include "DataFormats/PatCandidates/interface/MET.h"
#include "DataFormats/PatCandidates/interface/PackedCandidate.h"
//
// class declaration
//

class MiniAnalyzer : public edm::EDAnalyzer {
   public:
      explicit MiniAnalyzer(const edm::ParameterSet&);
      ~MiniAnalyzer();

   private:
      virtual void analyze(const edm::Event&, const edm::EventSetup&) override;

      // ----------member data ---------------------------
      edm::EDGetTokenT<reco::VertexCollection> vtxToken_;
      edm::EDGetTokenT<pat::MuonCollection> muonToken_;
      edm::EDGetTokenT<pat::ElectronCollection> electronToken_;
      edm::EDGetTokenT<pat::TauCollection> tauToken_;
      edm::EDGetTokenT<pat::PhotonCollection> photonToken_;
      edm::EDGetTokenT<pat::JetCollection> jetToken_;
      edm::EDGetTokenT<pat::JetCollection> fatjetToken_;
      edm::EDGetTokenT<pat::METCollection> metToken_;
};

MiniAnalyzer::MiniAnalyzer(const edm::ParameterSet& iConfig):
    vtxToken_(consumes<reco::VertexCollection>(iConfig.getParameter<edm::InputTag>("vertices"))),
    muonToken_(consumes<pat::MuonCollection>(iConfig.getParameter<edm::InputTag>("muons"))),
    electronToken_(consumes<pat::ElectronCollection>(iConfig.getParameter<edm::InputTag>("electro
    tauToken_(consumes<pat::TauCollection>(iConfig.getParameter<edm::InputTag>("taus"))),
    photonToken_(consumes<pat::PhotonCollection>(iConfig.getParameter<edm::InputTag>("photons")))
    jetToken_(consumes<pat::JetCollection>(iConfig.getParameter<edm::InputTag>("jets"))),
    fatjetToken_(consumes<pat::JetCollection>(iConfig.getParameter<edm::InputTag>("fatjets"))),
    metToken_(consumes<pat::METCollection>(iConfig.getParameter<edm::InputTag>("mets")))
{
}

MiniAnalyzer::~MiniAnalyzer()
{
}


void
MiniAnalyzer::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
    edm::Handle<reco::VertexCollection> vertices;
    iEvent.getByToken(vtxToken_, vertices);
    if (vertices->empty()) return; // skip the event if no PV found
    const reco::Vertex &PV = vertices->front();

    edm::Handle<pat::MuonCollection> muons;
    iEvent.getByToken(muonToken_, muons);
    for (const pat::Muon &mu : *muons) {
        if (mu.pt() < 5 || !mu.isLooseMuon()) continue;
        printf("muon with pt %4.1f, dz(PV) %+5.3f, POG loose id %d, tight id %d\n",
                mu.pt(), mu.muonBestTrack()->dz(PV.position()), mu.isLooseMuon(), mu.isTightMuon(
    }

    edm::Handle<pat::ElectronCollection> electrons;
    iEvent.getByToken(electronToken_, electrons);
    for (const pat::Electron &el : *electrons) {
        if (el.pt() < 5) continue;
        printf("elec with pt %4.1f, supercluster eta %+5.3f, sigmaIetaIeta %.3f (%.3f with full5x
                    el.pt(), el.superCluster()->eta(), el.sigmaIetaIeta(), el.full5x5_sigmaIetaIe
    }
```

```
    edm::Handle<pat::PhotonCollection> photons;
    iEvent.getByToken(photonToken_, photons);
    for (const pat::Photon &pho : *photons) {
        if (pho.pt() < 20 or pho.chargedHadronIso()/pho.pt() > 0.3) continue;
        printf("phot with pt %4.1f, supercluster eta %+5.3f, sigmaIetaIeta %.3f (%.3f with full5x
                    pho.pt(), pho.superCluster()->eta(), pho.sigmaIetaIeta(), pho.full5x5_sigmaIe
    }


    edm::Handle<pat::TauCollection> taus;
    iEvent.getByToken(tauToken_, taus);
    for (const pat::Tau &tau : *taus) {
        if (tau.pt() < 20) continue;
        printf("tau  with pt %4.1f, dxy signif %.1f, ID(byMediumCombinedIsolationDeltaBetaCorr3Hi
                    tau.pt(), tau.dxy_Sig(), tau.tauID("byMediumCombinedIsolationDeltaBetaCorr3Hi
    }


    edm::Handle<pat::JetCollection> jets;
    iEvent.getByToken(jetToken_, jets);
    int ijet = 0;
    for (const pat::Jet &j : *jets) {
        if (j.pt() < 20) continue;
        printf("jet  with pt %5.1f (raw pt %5.1f), eta %+4.2f, btag CSV %.3f, CISV %.3f, pileup m
            j.pt(), j.pt()*j.jecFactor("Uncorrected"), j.eta(), std::max(0.f,j.bDiscriminator("co
        if ((++ijet) == 1) { // for the first jet, let's print the leading constituents
            std::vector<reco::CandidatePtr> daus(j.daughterPtrVector());
            std::sort(daus.begin(), daus.end(), [](const reco::CandidatePtr &p1, const reco::Cand
            for (unsigned int i2 = 0, n = daus.size(); i2 < n && i2 <= 3; ++i2) {
                const pat::PackedCandidate &cand = dynamic_cast<const pat::PackedCandidate &>(*da
                printf("         constituent %3d: pt %6.2f, dz(pv) %+.3f, pdgId %+3d\n", i2,cand.
            }
        }
    }


    edm::Handle<pat::JetCollection> fatjets;
    iEvent.getByToken(fatjetToken_, fatjets);
    for (const pat::Jet &j : *fatjets) {
        printf("AK8j with pt %5.1f (raw pt %5.1f), eta %+4.2f, mass %5.1f ungroomed, %5.1f softdr
            j.pt(), j.pt()*j.jecFactor("Uncorrected"), j.eta(), j.mass(), j.userFloat("ak8PFJetsC




        // To get the constituents of the AK8 jets, you have to loop over all of the
        // daughters recursively. To save space, the first two constituents are actually
        // the Soft Drop SUBJETS, which will then point to their daughters.
        // The remaining constituents are those constituents removed by soft drop but
        // still in the AK8 jet.
std::vector<reco::Candidate const *> constituents;
        for ( unsigned ida = 0; ida < j.numberOfDaughters(); ++ida ) {
  reco::Candidate const * cand = j.daughter(ida);
  if ( cand->numberOfDaughters() == 0 )
            constituents.push_back( cand ) ;
  else {
    for ( unsigned jda = 0; jda < cand->numberOfDaughters(); ++jda ) {
      reco::Candidate const * cand2 = cand->daughter(jda);
              constituents.push_back( cand2 );
            }
          }
        }
std::sort( constituents.begin(), constituents.end(), [] (reco::Candidate const * ida, reco::Candi

for ( unsigned int ida = 0; ida < constituents.size(); ++ida ) {
  const pat::PackedCandidate &cand = dynamic_cast<const pat::PackedCandidate &>(*constituents[ida
```

```
          "printf(  constituent %3d: pt %6.2f, dz(pv) %+.3f, pdgId %+3d\n", ida,cand.pt(),cand.dz(
        }


auto wSubjets = j.subjets("SoftDrop");
for ( auto const & iw : wSubjets ) {
          "printf(  Subjet with pt %5.1f (raw pt %5.1f), eta %+4.2f, mass %5.1f ungroomed\n",
                  iw->pt(), iw->pt()*iw->jecFactor("Uncorrected"), iw->eta(), iw->mass() );

        }
auto tSubjets = j.subjets("CMSTopTag");
for ( auto const & it : tSubjets ) {
          "printf(  Subjet with pt %5.1f (raw pt %5.1f), eta %+4.2f, mass %5.1f ungroomed\n",
                  it->pt(), it->pt()*it->jecFactor("Uncorrected"), it->eta(), it->mass() );

        }
    }

    edm::Handle<pat::METCollection> mets;
    iEvent.getByToken(metToken_, mets);
    const pat::MET &met = mets->front();
    printf("MET: pt %5.1f, phi %+4.2f, sumEt (%.1f). genMET %.1f. MET with JES up/down: %.1f/%.1f
        met.pt(), met.phi(), met.sumEt(),
        met.genMET()->pt(),
        met.shiftedPt(pat::MET::JetEnUp), met.shiftedPt(pat::MET::JetEnDown));

    printf("\n");
}

//define this as a plug-in
DEFINE_FWK_MODULE(MiniAnalyzer);
```

python configuration to run it

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("Demo")

process.load("FWCore.MessageService.MessageLogger_cfi")
process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(10) )

process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring(
        '/store/cmst3/user/gpetrucc/miniAOD/v1/TT_Tune4C_13TeV-pythia8-tauola_PU_S14_PAT.root'
    )
)

process.demo = cms.EDAnalyzer("MiniAnalyzer",
    vertices = cms.InputTag("offlineSlimmedPrimaryVertices"),
    muons = cms.InputTag("slimmedMuons"),
    electrons = cms.InputTag("slimmedElectrons"),
    taus = cms.InputTag("slimmedTaus"),
    photons = cms.InputTag("slimmedPhotons"),
    jets = cms.InputTag("slimmedJets"),
    fatjets = cms.InputTag("slimmedJetsAK8"),
    mets = cms.InputTag("slimmedMETs"),
)


process.p = cms.Path(process.demo)
```

Show example python code Hide example python code

```
# import ROOT in batch mode
import sys
oldargv = sys.argv[:]
sys.argv = [ '-b-' ]
```

```
import ROOT
ROOT.gROOT.SetBatch(True)
sys.argv = oldargv

# load FWLite C++ libraries
ROOT.gSystem.Load("libFWCoreFWLite.so");
ROOT.gSystem.Load("libDataFormatsFWLite.so");
ROOT.AutoLibraryLoader.enable()

# load FWlite python libraries
from DataFormats.FWLite import Handle, Events

muons, muonLabel = Handle("std::vector<pat::Muon>"), "slimmedMuons"
electrons, electronLabel = Handle("std::vector<pat::Electron>"), "slimmedElectrons"
photons, photonLabel = Handle("std::vector<pat::Photon>"), "slimmedPhotons"
taus, tauLabel = Handle("std::vector<pat::Tau>"), "slimmedTaus"
jets, jetLabel = Handle("std::vector<pat::Jet>"), "slimmedJets"
fatjets, fatjetLabel = Handle("std::vector<pat::Jet>"), "slimmedJetsAK8"
mets, metLabel = Handle("std::vector<pat::MET>"), "slimmedMETs"
vertices, vertexLabel = Handle("std::vector<reco::Vertex>"), "offlineSlimmedPrimaryVertices"
verticesScore = Handle("edm::ValueMap<float>")

# open file (you can use 'edmFileUtil -d /store/whatever.root' to get the physical file name)
events = Events("root://cms-xrd-global.cern.ch//store/relval/CMSSW_7_4_1/RelValTTbar_13/MINIAODSI

for iev,event in enumerate(events):
    if iev >= 10: break
    event.getByLabel(muonLabel, muons)
    event.getByLabel(electronLabel, electrons)
    event.getByLabel(photonLabel, photons)
    event.getByLabel(tauLabel, taus)
    event.getByLabel(jetLabel, jets)
    event.getByLabel(fatjetLabel, fatjets)
    event.getByLabel(metLabel, mets)
    event.getByLabel(vertexLabel, vertices)
    event.getByLabel(vertexLabel, verticesScore)

    print "\nEvent %d: run %6d, lumi %4d, event %12d" % (iev,event.eventAuxiliary().run(), event.

    # Vertices
    if len(vertices.product()) == 0 or vertices.product()[0].ndof() < 4:
        print "Event has no good primary vertex."
        continue
    else:
        PV = vertices.product()[0]
        print "PV at x,y,z = %+5.3f, %+5.3f, %+6.3f, ndof: %.1f, score: (pt2 of clustered objects

    # Muons
    for i,mu in enumerate(muons.product()):
        if mu.pt() < 5 or not mu.isLooseMuon(): continue
        print "muon %2d: pt %4.1f, dz(PV) %+5.3f, POG loose id %d, tight id %d." % (
            i, mu.pt(), mu.muonBestTrack().dz(PV.position()), mu.isLooseMuon(), mu.isTightMuon(PV

    # Electrons
    for i,el in enumerate(electrons.product()):
        if el.pt() < 5: continue
        print "elec %2d: pt %4.1f, supercluster eta %+5.3f, sigmaIetaIeta %.3f (%.3f with full5x5
                    i, el.pt(), el.superCluster().eta(), el.sigmaIetaIeta(), el.full5x5_sigmaIeta

    # Photon
    for i,pho in enumerate(photons.product()):
        if pho.pt() < 20 or pho.chargedHadronIso()/pho.pt() > 0.3: continue
        print "phot %2d: pt %4.1f, supercluster eta %+5.3f, sigmaIetaIeta %.3f (%.3f with full5x5
                    i, pho.pt(), pho.superCluster().eta(), pho.sigmaIetaIeta(), pho.full5x5_sigma

    # Tau
    for i,tau in enumerate(taus.product()):
```

```
    if tau.pt() < 20: continue
    print "tau  %2d: pt %4.1f, dxy signif %.1f, ID(byMediumCombinedIsolationDeltaBetaCorr3Hit
                i, tau.pt(), tau.dxy_Sig(), tau.tauID("byMediumCombinedIsolationDeltaBetaCorr


# Jets (standard AK4)
for i,j in enumerate(jets.product()):
    if j.pt() < 20: continue
    print "jet %3d: pt %5.1f (raw pt %5.1f, matched-calojet pt %5.1f), eta %+4.2f, btag run1(
        i, j.pt(), j.pt()*j.jecFactor('Uncorrected'), j.userFloat("caloJetMap:pt"), j.eta(),
    if i == 0: # for the first jet, let's print the leading constituents
        constituents = [ j.daughter(i2) for i2 in xrange(j.numberOfDaughters()) ]
        constituents.sort(key = lambda c:c.pt(), reverse=True)
        for i2, cand in enumerate(constituents):
            if i2 > 4:
                    print "          ....."
                    break
            print "          constituent %3d: pt %6.2f, dz(pv) %+.3f, pdgId %+3d" % (i2,cand.p

# Fat AK8 Jets
for i,j in enumerate(fatjets.product()):
    print "jetAK8 %3d: pt %5.1f (raw pt %5.1f), eta %+4.2f, mass %5.1f ungroomed, %5.1f softd
        i, j.pt(), j.pt()*j.jecFactor('Uncorrected'), j.eta(), j.mass(), j.userFloat('ak8PFJe


    # To get the constituents of the AK8 jets, you have to loop over all of the
    # daughters recursively. To save space, the first two constituents are actually
    # the Soft Drop SUBJETS, which will then point to their daughters.
    # The remaining constituents are those constituents removed by soft drop but
    # still in the AK8 jet.
    constituents = []
    for ida in xrange( j.numberOfDaughters() ) :
        cand = j.daughter(ida)
        if cand.numberOfDaughters() == 0 :
            constituents.append( cand )
        else :
            for jda in xrange( cand.numberOfDaughters() ) :
                cand2 = cand.daughter(jda)
                constituents.append( cand2 )
    constituents.sort(key = lambda c:c.pt(), reverse=True)
    for i2, cand in enumerate(constituents):
        if i2 > 4:
                    print "          ....."
                    break
        print "          constituent %3d: pt %6.2f, pdgId %+3d, #dau %+3d" % (i2,cand.pt(),can

    wSubjets = j.subjets('SoftDrop')
    for iw,wsub in enumerate( wSubjets ) :
        print "   w subjet %3d: pt %5.1f (raw pt %5.1f), eta %+4.2f, mass %5.1f " % (
            iw, wsub.pt(), wsub.pt()*wsub.jecFactor('Uncorrected'), wsub.eta(), wsub.mass()
            )
    tSubjets = j.subjets('CMSTopTag')
    for it,tsub in enumerate( tSubjets ) :
        print "   t subjet %3d: pt %5.1f (raw pt %5.1f), eta %+4.2f, mass %5.1f " % (
            it, tsub.pt(), tsub.pt()*tsub.jecFactor('Uncorrected'), tsub.eta(), tsub.mass()
            )

# MET:
met = mets.product().front()
print "MET: pt %5.1f, phi %+4.2f, sumEt (%.1f). rawMET: %.1f, genMET %.1f. MET with JES up/do
    met.pt(), met.phi(), met.sumEt(),
    met.uncorrectedPt(), #<<?-- seems to return the same as pt(): help needed from JetMET exp
    met.genMET().pt(),
    met.shiftedPt(ROOT.pat.MET.JetEnUp), met.shiftedPt(ROOT.pat.MET.JetEnDown));
```

Packed ParticleFlow Candidates UPDATED

All candidates reconstructed by the ParticleFlow algorithm are saved in MiniAOD in the **`packedPFCandidates`** collection, using as dataformat pat::PackedCandidate.

For all packed PF candidates, this information is stored:

- the momentum 4-vector, accessible through the usual methods like `pt()`, `eta()`, `phi()`, `mass()`, `energy()`, `px()`, `p4()`, `polarP4()`.
  An additional method `phiAtVtx()` is provided, returning the phi of the candidate's track at the vertex; this is identical to `phi()` for the vast majority of the particles, but the two might differ for some of them if the calorimeters had contributed significantly in defining the 4-vector of the particle.
- the particle charge and pdgId: 11, 13, 22 for ele/mu/gamma, 211 for charged hadrons, 130 for neutral hadrons, 1 and 2 for hadronic and em particles in HF.
- quality flags:
  - Inner hit information: `lostInnerHits()` will return -1 (`validHitInFirstPixelBarrelLayer`) if the track has a valid hit in the first pixel barrel layer, 0 (`noLostInnerHits`) if it doesn't have such hit because of geometrical or detector inefficiencies (i.e. the hit wasn't expected to be there), 1 (`oneLostHit`) if the track extrapolation towards the beamline crosses an active detector but no hit is found there, 2 (`moreLostHits`) if there are at least two missing expected inner hits.
    For electrons, this information refers to the `gsfTrack()`, i.e. the usual cuts `electron.gsfTrack()->trackerExpectedHitsInner().numberOfLostHits() <= X` can be implemented as `cand.lostInnerHits() <= X` (but only for X equal to 0 or 1)
  - High-purity track: `trackHighPurity()` will return true for charged candidates whose `reco::Teack` had the `highPurity` quality flag set.
- Puppi Weights can be accessed via `puppiWeight()` and `puppiWeightNoLep()` functions.
  Two sets of weights are included, except in 74X version 1, so that one can compute puppi isolation for muons, electrons and taus (recipes forthcoming). For completeness, `puppiWeight` are used to cluster the jets and `puppiWeightNoLep` are used to compute the puppi MET.
- Since 76X, the packed PF candidates corresponding to neutral hadrons (pdgId 130) and HF hadrons (pdgId 1) contain the fraction of energy measured from the hadronic calorimeter, accessible via the **`hcalFraction()`** method. UPDATED

PV Assignment

Starting from 74X, more information about PV association are provided in order to tune the PU rejection techniques to the analyses needs. The following features are added:

- `vertexRef()` now returns for each packed candidate the associated PV (before 740 it was always the `pv[0]`). The association is based on CommonTools/RecoAlgos/python/sortedPFPrimaryVertices_cfi.py
- `pvAssociationQuality()` method is added and returns the quality of PV-candidate association, in particular (please note that in the following *the PV* means the associated PV returned by vertexRef()):
  - **UsedInFitTight**: Returns 7, the track is used in *the PV* fit and the weight is above 0.5
  - **UsedInFitLoose**: Returns 6, the track is used in *the PV* fit and the weight is below 0.5
  - **CompatibilityDz**: Returns 5, the track is not used in fit but is very close in dZ to *the PV* (significance of dZ < 5 or dZ < 300um)
  - **CompatibilityBTag**: Returns 4, the track is not compatible with *the PV* but it is close to the nearest jet axis starting from the PV (distance to jet axis < 700um)
  - **NotReconstructedPrimary**: Returns 0, the track is not associted to any PV and is compatible with the BeamSpot hence it is likely to be originating from an interaction for which we did not reconstruct *the PV* (beamspot compatiblity: dxy sig < 2sigma and dxy < 200um)
  - **OtherDeltaZ**: Returns 1, none of the above criteria is satisfied, hence the closest in dZ vertex

is associated

The old `fromPV()` is still provided and is expected to given in 99.99% of the tracks the same behaviour as in previous versions. In addition `fromPV(int i)` provides the same information of `fromPV` but computed wrt PV[i], this allow to rerun CHS changing the signal vertex.

The meaning of `fromPV()` results are unchanged, `fromPV()` returns a number between 3 and 0 to define how tight the association with *the PV* is:

- the tighest, 3 (`PVUsedInFit`), is if the track is used in *the PV* fit;
- 2 (`PVTight`) is if the track is not used in the fit of any of the other PVs, and is closest in z to *the PV*,
- 1 (`PVLoose`) is if the track is closest in z to a PV other then *the PV*.
- 0 (`NoPV`) is returned if the track is used in the fit of another PV.

The definition normally used for isolation calculations is `fromPV() > 1`; the definition used for CHS subtraction in jets is `fromPV() > 0`.

Impact Parameter

- the longitudinal and transverse impact parameters with respect to the PV: `dxz()`, `dz()`.
  The impact parameters can also be recomputed with respect to any other position by calling the methods `dxz(point)`, `dz(point)`).
  The `vertex()` method returns the position of the point of closest approach to the PV. This shouldn't be confused with the `vertexRef()` method, that returns a reference to the PV itself.

In addition the impact parameter accessors behaviour is still compatible with previous versions but it provides additional features, namely:

- `dz()` still returns the ip wrt PV[0]
- `dz(point)` still return the ip wrt a given 3D point
- `dz(int i)` returns the ip wrt PV[i]
- `dzAssociatedPV()` returns the ip wrt the PV associated to this candidate

Embedded track information

For charged packed PF candidates with $p_T > 0.95$ GeV, additional information is stored:

- the uncertainty on the impact parameter `dzError()`, `dxyError()`
- the number of hits and pixel hits on the track (`numberOfHits()`, `numberOfPixelHits()`)
- the `reco::Track` of the candidate is provided by the `pseudoTrack()` method, with:
  - the momentum of the particle
  - an approximate covariance matrix of the track state at the vertex
  - approximate `hitPattern()` and `trackerExpectedHitsInner()` that yield the correct number of hits, pixel hits and the information returned by `lostInnerHits()`
    (but nothing more, so e.g. you can't rely on methods counting layers, or expected outer hits, or lost hits within the track)
  - the track normalized chisquare (truncated to an integer)
  - the `highPurity` quality flag set, if the original track had it.

Pointers and navigation

All physics objects except the MET contain pointers to the packed PF candidates corresponding to the PFCandidates they were made from:

- For all objects, this information is returned by the methods `numberOfSourceCandidatePtrs()` and `sourceCandidatePtr(index)` methods.
- For jets, they are also accessible through the natural interface `daughter(index)` and `numberOfDaughters()`, with the following caveat:
  - in case of `slimmedJets` (i.e. default AK4CHS jets) the daughters of the jets are directly the packed candidates
  - in case of `slimmedJetsAK8` (i.e. fat jets used for substructures analysis) the daughter Ptrs are a mixed collection of SubJets and PackedCandidates. In order to have the full list of PackedCandidates you have to add the daughters of the subjets (see example below)

Show example python code Hide example python code

```
#! /usr/bin/env python
import ROOT
from DataFormats.FWLite import Events, Handle
events = Events (["root://cms-xrd-global.cern.ch//store/relval/CMSSW_7_4_0/RelValTTbar_13/MINIAOD

handleFatJets  = Handle ("std::vector<pat::Jet>")
labelFatJets = ("slimmedJetsAK8")
c=0
for event in events:
    event.getByLabel (labelFatJets,handleFatJets)
    fatjets = handleFatJets.product()
    for fj in fatjets :
        if c > 10 :
                break
        c+=1

        print "FatJet",fj.pt(),fj.eta()
        allconstituents = []
        for constituent in fj.daughterPtrVector() :
                if constituent.numberOfDaughters() > 0 :
                        allconstituents.extend([x for x in constituent.daughterPtrVector()])
                else:
                        allconstituents.append(constituent)
        print " n. constituents  " , len(fj.daughterPtrVector())
        print " n. subjets       " , len(fj.subjets())
        print " n. all constituents      " , len(allconstituents)
```

- For taus, they are also accessible through the more specific methods like `leadChargedHadrCand()`, `leadNeutralCand()`, `leadCand()`, `signalCands()`, `isolationCands()`, ...
  (note instead that the methods with similar name but with a `PF` inside, like `signalPFCands()`, are not usable on MiniAOD)
- For muons and for electrons or photons associated to a single PF candidate, the `originalObjectRef()` also points to that PF candidate.
- For electrons and photons, they're also accessible through the `associatedPackedPFCandidates()` method.

Examples

The packed PF candidates can be used either directly in the analysis code, e.g. to recompute isolation variables, or it is possible to use them as input to standard CMSSW tools in order to re-reconstruct jets with different algorithms or after masking the leptons, to run b-tagging, and so on.

In the code examples below we will show how to use the packed PF candidates to compute some isolation variable or some quantity based on the constituents of a jet. The more advanced CMSSW patterns are instead covered in the last section of this tutorial.

Show example CMSSW code Hide example CMSSW code

Code snippets for an EDAnalyzer

```
// system include files
#include <memory>
#include <cmath>

// user include files
#include "FWCore/Framework/interface/Frameworkfwd.h"
#include "FWCore/Framework/interface/EDAnalyzer.h"
#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/MakerMacros.h"
#include "FWCore/ParameterSet/interface/ParameterSet.h"

#include "DataFormats/Math/interface/deltaR.h"
#include "DataFormats/PatCandidates/interface/PackedCandidate.h"
#include "DataFormats/PatCandidates/interface/Muon.h"
#include "DataFormats/PatCandidates/interface/Electron.h"
#include "DataFormats/PatCandidates/interface/Jet.h"

class PackedCandAnalyzer : public edm::EDAnalyzer {
   public:
      explicit PackedCandAnalyzer(const edm::ParameterSet&);
      ~PackedCandAnalyzer() {}

   private:
      virtual void analyze(const edm::Event&, const edm::EventSetup&) override;

      edm::EDGetTokenT<pat::ElectronCollection> electronToken_;
      edm::EDGetTokenT<pat::MuonCollection> muonToken_;
      edm::EDGetTokenT<pat::JetCollection> jetToken_;
      edm::EDGetTokenT<pat::PackedCandidateCollection> pfToken_;
};

PackedCandAnalyzer::PackedCandAnalyzer(const edm::ParameterSet& iConfig):
    electronToken_(consumes<pat::ElectronCollection>(iConfig.getParameter<edm::InputTag>("electro
    muonToken_(consumes<pat::MuonCollection>(iConfig.getParameter<edm::InputTag>("muons"))),
    jetToken_(consumes<pat::JetCollection>(iConfig.getParameter<edm::InputTag>("jets"))),
    pfToken_(consumes<pat::PackedCandidateCollection>(iConfig.getParameter<edm::InputTag>("pfCand
{
}

void PackedCandAnalyzer::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
    edm::Handle<pat::MuonCollection> muons;
    iEvent.getByToken(muonToken_, muons);
    edm::Handle<pat::ElectronCollection> electrons;
    iEvent.getByToken(electronToken_, electrons);
    edm::Handle<pat::PackedCandidateCollection> pfs;
    iEvent.getByToken(pfToken_, pfs);
    edm::Handle<pat::JetCollection> jets;
    iEvent.getByToken(jetToken_, jets);

    std::vector<const reco::Candidate *> leptons;
    for (const pat::Muon &mu : *muons) leptons.push_back(&mu);
    for (const pat::Electron &el : *electrons) leptons.push_back(&el);

    for (const reco::Candidate *lep : leptons) {
        if (lep->pt() < 5) continue;
        // initialize sums
        double charged = 0, neutral = 0, pileup  = 0;
        // now get a list of the PF candidates used to build this lepton, so to exclude them
        std::vector<reco::CandidatePtr> footprint;
        for (unsigned int i = 0, n = lep->numberOfSourceCandidatePtrs(); i < n; ++i) {
            footprint.push_back(lep->sourceCandidatePtr(i));
        }
        // now loop on pf candidates
        for (unsigned int i = 0, n = pfs->size(); i < n; ++i) {
            const pat::PackedCandidate &pf = (*pfs)[i];
```

```
            if (deltaR(pf,*lep) < 0.2) {
                // pfcandidate-based footprint removal
                if (std::find(footprint.begin(), footprint.end(), reco::CandidatePtr(pfs,i)) != f
                    continue;
                }
                if (pf.charge() == 0) {
                    if (pf.pt() > 0.5) neutral += pf.pt();
                } else if (pf.fromPV() >= 2) {
                    charged += pf.pt();
                } else {
                    if (pf.pt() > 0.5) pileup += pf.pt();
                }
            }
        }
        // do deltaBeta
        double iso = charged + std::max(0.0, neutral-0.5*pileup);
        printf("%-8s of pt %6.1f, eta %+4.2f: relIso = %5.2f\n",
                    abs(lep->pdgId())==13 ? "muon" : "electron",
                    lep->pt(), lep->eta(), iso/lep->pt());

    }

    // Let's compute the fraction of charged pt from particles with dz < 0.1 cm
    for (const pat::Jet &j :  *jets) {
        if (j.pt() < 40 || fabs(j.eta()) > 2.4) continue;
        double in = 0, out = 0;
        for (unsigned int id = 0, nd = j.numberOfDaughters(); id < nd; ++id) {
            const pat::PackedCandidate &dau = dynamic_cast<const pat::PackedCandidate &>(*j.daugh
            if (dau.charge() == 0) continue;
            (fabs(dau.dz())<0.1 ? in : out) += dau.pt();
        }
        double sum = in + out;
        printf("Jet with pt %6.1f, eta %+4.2f, beta(0.1) = %+5.3f, pileup mva disc %+.2f\n",
                j.pt(),j.eta(), sum ? in/sum : 0, j.userFloat("pileupJetId:fullDiscriminant"));
    }
}

//define this as a plug-in
DEFINE_FWK_MODULE(PackedCandAnalyzer);
```

Python configuration

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("Demo")

process.load("FWCore.MessageService.MessageLogger_cfi")
process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(10) )

process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring(
        '/store/cmst3/user/gpetrucc/miniAOD/v1/TT_Tune4C_13TeV-pythia8-tauola_PU_S14_PAT.root'
    )
)

process.demo = cms.EDAnalyzer("PackedCandAnalyzer",
    electrons = cms.InputTag("slimmedElectrons"),
    muons = cms.InputTag("slimmedMuons"),
    jets = cms.InputTag("slimmedJets"),
    pfCands = cms.InputTag("packedPFCandidates"),
)

process.p = cms.Path(process.demo)
```

Show example python code Hide example python code

```
# after loading ROOT and FWlite libraries as described in the introduction
```

```
# define deltaR
from math import hypot, pi
def deltaR(a,b):
    dphi = abs(a.phi()-b.phi());
    if dphi > pi: dphi = 2*pi-dphi
    return hypot(a.eta()-b.eta(),dphi)

muons, muonLabel = Handle("std::vector<pat::Muon>"), "slimmedMuons"
electrons, electronLabel = Handle("std::vector<pat::Electron>"), "slimmedElectrons"
jets, jetLabel = Handle("std::vector<pat::Jet>"), "slimmedJets"
pfs, pfLabel = Handle("std::vector<pat::PackedCandidate>"), "packedPFCandidates"

# open file (you can use 'edmFileUtil -d /store/whatever.root' to get the physical file name)
events = Events("root://eoscms//eos/cms/store/cmst3/user/gpetrucc/miniAOD/v1/TT_Tune4C_13TeV-pyth

for iev,event in enumerate(events):
    event.getByLabel(muonLabel, muons)
    event.getByLabel(electronLabel, electrons)
    event.getByLabel(pfLabel, pfs)
    event.getByLabel(jetLabel, jets)

    print "\nEvent %d: run %6d, lumi %4d, event %12d" % (iev,event.eventAuxiliary().run(), event.

    # Let's compute lepton PF Isolation with R=0.2, 0.5 GeV threshold on neutrals, and deltaBeta
    leps  = [ p for p in muons.product() ] + [ p for p in electrons.product() ]
    for lep in leps:
        # skip those below 5 GeV, which we don't care about
        if lep.pt() < 5: continue
        # initialize sums
        charged = 0
        neutral = 0
        pileup  = 0
        # now get a list of the PF candidates used to build this lepton, so to exclude them
        footprint = set()
        for i in xrange(lep.numberOfSourceCandidatePtrs()):
            footprint.add(lep.sourceCandidatePtr(i).key()) # the key is the index in the pf colle
        # now loop on pf candidates
        for ipf,pf in enumerate(pfs.product()):
            if deltaR(pf,lep) < 0.2:
                # pfcandidate-based footprint removal
                if ipf in footprint: continue
                # add up
                if (pf.charge() == 0):
                    if pf.pt() > 0.5: neutral += pf.pt()
                elif pf.fromPV() >= 2:
                    charged += pf.pt()
                else:
                    if pf.pt() > 0.5: pileup += pf.pt()
        # do deltaBeta
        iso = charged + max(0, neutral-0.5*pileup)
        print "%-8s of pt %6.1f, eta %+4.2f: relIso = %5.2f" % (
                    "muon" if abs(lep.pdgId())==13 else "electron",
                    lep.pt(), lep.eta(), iso/lep.pt())

    # Let's compute the fraction of charged pt from particles with dz < 0.1 cm
    for i,j in enumerate(jets.product()):
        if j.pt() < 40 or abs(j.eta()) > 2.4: continue
        sums = [0,0]
        for id in xrange(j.numberOfDaughters()):
            dau = j.daughter(id)
            if (dau.charge() == 0): continue
            sums[ abs(dau.dz())<0.1 ] += dau.pt()
        sum = sums[0]+sums[1]
        print "Jet with pt %6.1f, eta %+4.2f, beta(0.1) = %+5.3f, pileup mva disc %+.2f" % (
                j.pt(),j.eta(), sums[1]/sum if sum else 0, j.userFloat("pileupJetId:fullDiscrimin

    if iev > 10: break
```

MC Truth

The standard PAT methods to access the MC truth are working with miniAOD, in particular you can for example use

```
pat::Electron::genParticle()
pat::Muon::genParticle()
pat::Jet::genParton()
pat::Jet::hadronFlavour()
pat::Jet::partonFlavour()
```

The only important difference wrt normal PAT is that not all gen particles are stored. In fact gen particles are stored in two different collections as explained below.

The **prunedGenParticles** are standard GenParticle selected with the GenParticlePruner, in particular

- only a subset (about 50-100 per event) of the gen particle are stored
- the selection is made trying to keep the most important information; see prunedGenParticles_cfi.py
- the navigation between the particles is working but it takes "shortcuts" (i.e. you do not go through the dropped particles)
- due to switch to Pythia8 (no more status = 3), the exact content is still being tuned. However, the selection used in 74X should contain essentially everything that was status 3 in pythia 6

In 74X and later, GenParticles now contain additional status flags that allow to select particles depending on their history without the need of navigating through the mother-daughter links manually. The information is provided by the GenStatusFlags class, which can be accessed via methods of the GenParticle object.
A presentation describing this feature was given in the PPD general meeting of 17 Jun 2015: slides

The pruned genParticles are the ones pointed by the MC matching of the high level patObjectes (e.g. `pat::Electron::genParticle()`)

The **packedGenParticles** collection contains all status=1 GenParticle in a reduced format as pat::PackedGenParticle, similar to the one used for PFCandidates. Besides the 4-vector, pdgId, a link to the first ancestor is saved in the **prunedGenParticles** collection (this can be accessed via the **`motherRef()`** or **`mother(0)`** methods; note that some generator level particles may be motherless, in that case **`motherRef`** will return a null reference, **`numberOfMothers`** will return zero, and **`mother(0)`** will return a null c++ pointer). This collection is used to re-cluster the MC truth (e.g. if you want to study a different jet algorithm or some substructure techniques).

- Note that particles are kept only op to a rapidity 6 (note that it's not a cut |y| < 6 and not on |&eta| < 6), so they should not be used for studying particles outside the detector acceptance, nor to recompute the generator level missing energy (that's anyway already available from the **`pat::MET`** object)
- Also the **packedGenParticles** now include information from the new status flags

The following examples show how to navigate between the two collections: the B mesons (500 < pdg ID < 600) are looked for in the **pruned** collection (because B are important particles, but they are not stable) and then we search which of the stable (status = 1) particles originated from a B decay. In order to do so we first ask to each packed candidate its "pruned" mother (NB, it is actually the first ancestor that survived the pruning), then we navigate back the pruned collection from such initial mother and see if we find the B meson.

Show example CMSSW python FWLite analyzer Hide example CMSSW python FWLite analyzer

```
import ROOT
import sys
from DataFormats.FWLite import Events, Handle
from math import *
```

```
def isAncestor(a,p) :
        if a == p :
                return True
        for i in xrange(0,p.numberOfMothers()) :
                if isAncestor(a,p.mother(i)) :
                        return True
        return False



events = Events (['root://cms-xrd-global.cern.ch//store/mc/Spring14miniaod/TTJets_MSDecaysCKM_cen

handlePruned  = Handle ("std::vector<reco::GenParticle>")
handlePacked  = Handle ("std::vector<pat::PackedGenParticle>")
labelPruned = ("prunedGenParticles")
labelPacked = ("packedGenParticles")

# loop over events
count= 0
for event in events:
    event.getByLabel (labelPacked, handlePacked)
    event.getByLabel (labelPruned, handlePruned)
    # get the product
    packed = handlePacked.product()
    pruned = handlePruned.product()

    for p in pruned :
        if abs(p.pdgId()) > 500 and abs(p.pdgId()) < 600 :
                print "PdgId : %s   pt : %s  eta : %s   phi : %s" %(p.pdgId(),p.pt(),p.eta(),p.ph
                print "     daughters"
                for pa in packed:
                        mother = pa.mother(0)
                        if mother and isAncestor(p,mother) :
                              print "     PdgId : %s   pt : %s  eta : %s   phi : %s" %(pa.pdgId()
```

Show example CMSSW C++ analyzer Hide example CMSSW C++ analyzer

```
// Original Author:  Andrea RIZZI
//         Created:  Mon, 07 Jul 2014 07:56:38 GMT

// system include files
#include <memory>

// user include files
#include "FWCore/Framework/interface/Frameworkfwd.h"
#include "FWCore/Framework/interface/EDAnalyzer.h"

#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/MakerMacros.h"

#include "FWCore/ParameterSet/interface/ParameterSet.h"
#include "DataFormats/PatCandidates/interface/PackedGenParticle.h"
#include "DataFormats/Candidate/interface/Candidate.h"
#include "DataFormats/HepMCCandidate/interface/GenParticle.h"
//
// class declaration
//

class MiniAODGenPartAnalyzer : public edm::EDAnalyzer {
   public:
      explicit MiniAODGenPartAnalyzer(const edm::ParameterSet&);
      ~MiniAODGenPartAnalyzer();
      bool isAncestor(const reco::Candidate * ancestor, const reco::Candidate * particle);
```

```
   private:
      virtual void beginJob() override;
      virtual void analyze(const edm::Event&, const edm::EventSetup&) override;
      virtual void endJob() override;

      edm::EDGetTokenT<edm::View<reco::GenParticle> > prunedGenToken_;
      edm::EDGetTokenT<edm::View<pat::PackedGenParticle> > packedGenToken_;
};

MiniAODGenPartAnalyzer::MiniAODGenPartAnalyzer(const edm::ParameterSet& iConfig):
  prunedGenToken_(consumes<edm::View<reco::GenParticle> >(iConfig.getParameter<edm::InputTag>("pr
  packedGenToken_(consumes<edm::View<pat::PackedGenParticle> >(iConfig.getParameter<edm::InputTag
{
}


MiniAODGenPartAnalyzer::~MiniAODGenPartAnalyzer()
{
}

//Check recursively if any ancestor of particle is the given one

bool MiniAODGenPartAnalyzer::isAncestor(const reco::Candidate* ancestor, const reco::Candidate *
{
//particle is already the ancestor
        if(ancestor == particle ) return true;

//otherwise loop on mothers, if any and return true if the ancestor is found
        for(size_t i=0;i< particle->numberOfMothers();i++)
        {
                if(isAncestor(ancestor,particle->mother(i))) return true;
        }
//if we did not return yet, then particle and ancestor are not relatives
        return false;
}

void
MiniAODGenPartAnalyzer::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
        using namespace edm;
        using namespace reco;
        using namespace pat;

        // Pruned particles are the one containing "important" stuff
        Handle<edm::View<reco::GenParticle> > pruned;
        iEvent.getByToken(prunedGenToken_,pruned);

        // Packed particles are all the status 1, so usable to remake jets
        // The navigation from status 1 to pruned is possible (the other direction should be made
        Handle<edm::View<pat::PackedGenParticle> > packed;
        iEvent.getByToken(packedGenToken_,packed);

        //let's try to find all status1 originating directly from a B meson decay

        for(size_t i=0; i<pruned->size();i++){
                if(abs((*pruned)[i].pdgId()) > 500 && abs((*pruned)[i].pdgId()) <600){
                        const Candidate * bMeson = &(*pruned)[i];
                        std::cout << "PdgID: " << bMeson->pdgId() << " pt " << bMeson->pt() << "
                        std::cout << "  found daugthers: " << std::endl;
                        for(size_t j=0; j<packed->size();j++){
//get the pointer to the first survied ancestor of a given packed GenParticle in the prunedCollec
                                const Candidate * motherInPrunedCollection = (*packed)[j].mother(
                                if(motherInPrunedCollection != nullptr && isAncestor( bMeson , mo
                                        std::cout << "     PdgID: " << (*packed)[j].pdgId() << "
                                }
                        }
                }
```

```
        }


}


// ------------ method called once each job just before starting event loop  ------------
        void
MiniAODGenPartAnalyzer::beginJob()
{
}

// ------------ method called once each job just after ending the event loop  ------------
        void
MiniAODGenPartAnalyzer::endJob()
{
}

DEFINE_FWK_MODULE(MiniAODGenPartAnalyzer);
```

Python cfg for cmsRun

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("Demo")

process.load("FWCore.MessageService.MessageLogger_cfi")

process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(-1) )

process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring("root://cms-xrd-global.cern.ch//store/mc/Spring14miniaod/TT
)

process.demo = cms.EDAnalyzer('MiniAODGenPartAnalyzer',
packed = cms.InputTag("packedGenParticles"),
pruned = cms.InputTag("prunedGenParticles")
)


process.p = cms.Path(process.demo)
```

Please note that you need to the following libraries in your BuildFile.xml

```
DataFormats/PatCandidates
DataFormats/HepMCCandidate
```

Flavour information has been updated in 74X version 2 miniAODs (see 1, 2). The differences wrt to version 1 are the following:

- hadron-based flavour is no longer given priority over the parton-based flavour. This means that `partonFlavour()` will return a purely parton-based flavor (in version 1 if there was a mismatch between the parton- and hadron-based flavour, the priority was given to the hadron-based flavour). The `hadronFlavour()` will return a purely hadron-based flavour, as was the case in version 1
- the selection of generator partons matched to jets has been updated so that `genParton().pdgId()` can be used to obtain the *physics* definition of the jet flavour
- generator particles used in the jet flavour definition are now selected from the **prunedGenParticles** collection (in version 1 the **genParticles** collection was used), which means that more jets will have undefined `partonFlavour()` because some of the soft partons, which would otherwise be used to set the jet flavor, are dropped from the **prunedGenParticles** collection

- generator leptons are clustered by default, in addition to generator partons and hadrons. This is potentially useful for identifying jets from hadronic tau decays
- `JetFlavourInfo` is embedded into PAT jets by default making it possible to get additional information about the clustered generator-level partons, hadrons, and leptons

For more information about different jet flavour definitions, please refer to SWGuideBTagMCTools.

One collections of gen jets is saved, **`slimmedGenJets`**, made from **`ak4GenJets`**, i.e. using anti-k_T clustering out of stable gen particles. Starting from 74X neutrinos and other invisible BSM particles are excluded from GenJets, while they were included in CMSSW 70X (CSA14) and 72X (Phys14) campaigns.

Links to the daughters are available through the **`daughter(idx)`** and **`numberOfDaughters()`** methods, and they will point into the collection of packed gen candidates described below.
It should be noted that the **`getGenConstituents`** method instead will NOT work, since they explicitly require the daughter to be a full **`reco::GenParticle`** and not a packed one. Issues with missing daughter references that affected forward jets in CSA14 and Phys14 campaigns should be solved now in 74X, for jets within the detector acceptance.

Trigger

Trigger bits for each path are stored as edm::TriggerResults, as standard in CMSSW.

In order to access paths by trigger name, you can get an edm::TriggerNames instance from the Event (both in full framework and in fwlite, see example at the bottom of the section), which translates indices in names. The mapping is the same in all events sharing the same trigger configuration (which is identified by the `parameterSetID()` method of the trigger results object).

Trigger objects associated to filters in the HLT are saved as instances of pat::TriggerObjectStandAlone, so that they can be used e.g. for trigger matching and measurements of trigger efficiency with the tag-and-probe method. The trigger names stored in the objects are packed, but they can be unpacked by passing an `edm::TriggerNames` instance, as shown in the example code at the bottom of this section.

Trigger prescales are encoded for each path into an pat::PackedTriggerPrescales, which can be unpacked with an `edm::TriggerNames` if access by path name is needed, just like for the `TriggerResults`.

- Since 74X version 2 miniAODs, the L1 prescales are also saved. Since a single HLT path may be seeded by multiple L1s with different prescales, there are two collections with instance labels "l1min" and "l1max": those are the maximum and minimum non-zero prescale for a given path. Useful cases are l1min = 1 (at least one seed is unprescaled), and l1min = l1max (the path has only one L1 seed, or all seeds have with the same prescale); paths which do not fall in either of the two cases should probably be handled with more care using the HLTConfigProvider (which works also on miniAOD).
- Trigger prescales for prescaled paths are not reported correctly in Run2015B 50ns PromptReco data (physTools/3378). This will be addressed in a next MiniAOD production, and is solved for newer eras (2015C or later).

Some extra trigger-level information copied directly from the AOD could also be available, namely the `l1extra` and `L1GlobalTriggerReadoutRecord`

Show example CMSSW code Hide example CMSSW code
EDAnalyzer code

```
// system include files
#include <memory>
#include <cmath>
```

```
// user include files
#include "FWCore/Framework/interface/Frameworkfwd.h"
#include "FWCore/Framework/interface/EDAnalyzer.h"
#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/MakerMacros.h"
#include "FWCore/ParameterSet/interface/ParameterSet.h"

#include "DataFormats/Math/interface/deltaR.h"
#include "FWCore/Common/interface/TriggerNames.h"
#include "DataFormats/Common/interface/TriggerResults.h"
#include "DataFormats/PatCandidates/interface/TriggerObjectStandAlone.h"
#include "DataFormats/PatCandidates/interface/PackedTriggerPrescales.h"

class MiniAODTriggerAnalyzer : public edm::EDAnalyzer {
   public:
      explicit MiniAODTriggerAnalyzer(const edm::ParameterSet&);
      ~MiniAODTriggerAnalyzer() {}

   private:
      virtual void analyze(const edm::Event&, const edm::EventSetup&) override;

      edm::EDGetTokenT<edm::TriggerResults> triggerBits_;
      edm::EDGetTokenT<pat::TriggerObjectStandAloneCollection> triggerObjects_;
      edm::EDGetTokenT<pat::PackedTriggerPrescales> triggerPrescales_;
};

MiniAODTriggerAnalyzer::MiniAODTriggerAnalyzer(const edm::ParameterSet& iConfig):
    triggerBits_(consumes<edm::TriggerResults>(iConfig.getParameter<edm::InputTag>("bits"))),
    triggerObjects_(consumes<pat::TriggerObjectStandAloneCollection>(iConfig.getParameter<edm::In
    triggerPrescales_(consumes<pat::PackedTriggerPrescales>(iConfig.getParameter<edm::InputTag>("
{
}

void MiniAODTriggerAnalyzer::analyze(const edm::Event& iEvent, const edm::EventSetup& iSetup)
{
    edm::Handle<edm::TriggerResults> triggerBits;
    edm::Handle<pat::TriggerObjectStandAloneCollection> triggerObjects;
    edm::Handle<pat::PackedTriggerPrescales> triggerPrescales;

    iEvent.getByToken(triggerBits_, triggerBits);
    iEvent.getByToken(triggerObjects_, triggerObjects);
    iEvent.getByToken(triggerPrescales_, triggerPrescales);

    const edm::TriggerNames &names = iEvent.triggerNames(*triggerBits);
    std::cout << "\n === TRIGGER PATHS === " << std::endl;
    for (unsigned int i = 0, n = triggerBits->size(); i < n; ++i) {
        std::cout << "Trigger " << names.triggerName(i) <<
                ", prescale " << triggerPrescales->getPrescaleForIndex(i) <<
                ": " << (triggerBits->accept(i) ? "PASS" : "fail (or not run)")
                << std::endl;
    }
    std::cout << "\n === TRIGGER OBJECTS === " << std::endl;
    for (pat::TriggerObjectStandAlone obj : *triggerObjects) { // note: not "const &" since we wa
        obj.unpackPathNames(names);
        std::cout << "\tTrigger object:  pt " << obj.pt() << ", eta " << obj.eta() << ", phi " <<
        // Print trigger object collection and type
        std::cout << "\t   Collection: " << obj.collection() << std::endl;
        std::cout << "\t   Type IDs:   ";
        for (unsigned h = 0; h < obj.filterIds().size(); ++h) std::cout << " " << obj.filterIds()
        std::cout << std::endl;
        // Print associated trigger filters
        std::cout << "\t   Filters:    ";
        for (unsigned h = 0; h < obj.filterLabels().size(); ++h) std::cout << " " << obj.filterLa
        std::cout << std::endl;
        std::vector<std::string> pathNamesAll  = obj.pathNames(false);
        std::vector<std::string> pathNamesLast = obj.pathNames(true);
        // Print all trigger paths, for each one record also if the object is associated to a '13
```

```
        // definition used in the PAT trigger producer) and if it's associated to the last filter
        // means that this object did cause this trigger to succeed; however, it doesn't work on
        std::cout << "\t   Paths (" << pathNamesAll.size()<<"/"<<pathNamesLast.size()<<"):    ";
        for (unsigned h = 0, n = pathNamesAll.size(); h < n; ++h) {
            bool isBoth = obj.hasPathName( pathNamesAll[h], true, true );
            bool isL3   = obj.hasPathName( pathNamesAll[h], false, true );
            bool isLF   = obj.hasPathName( pathNamesAll[h], true, false );
            bool isNone = obj.hasPathName( pathNamesAll[h], false, false );
            std::cout << "    " << pathNamesAll[h];
            if (isBoth) std::cout << "(L,3)";
            if (isL3 && !isBoth) std::cout << "(*,3)";
            if (isLF && !isBoth) std::cout << "(L,*)";
            if (isNone && !isBoth && !isL3 && !isLF) std::cout << "(*,*)";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;

}

//define this as a plug-in
DEFINE_FWK_MODULE(MiniAODTriggerAnalyzer);
```

Python configuration

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("Demo")

process.load("FWCore.MessageService.MessageLogger_cfi")
process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(10) )

process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring(
        '/store/cmst3/user/gpetrucc/miniAOD/v1/TT_Tune4C_13TeV-pythia8-tauola_PU_S14_PAT.root'
    )
)

process.demo = cms.EDAnalyzer("MiniAODTriggerAnalyzer",
    bits = cms.InputTag("TriggerResults","","HLT"),
    prescales = cms.InputTag("patTrigger"),
    objects = cms.InputTag("selectedPatTrigger"),
)

process.p = cms.Path(process.demo)
```

Show example python code Hide example python code

```
# import ROOT in batch mode
import sys
oldargv = sys.argv[:]
sys.argv = [ '-b-' ]
import ROOT
ROOT.gROOT.SetBatch(True)
sys.argv = oldargv

# load FWLite C++ libraries
ROOT.gSystem.Load("libFWCoreFWLite.so");
ROOT.gSystem.Load("libDataFormatsFWLite.so");
ROOT.AutoLibraryLoader.enable()

# load FWlite python libraries
from DataFormats.FWLite import Handle, Events

triggerBits, triggerBitLabel = Handle("edm::TriggerResults"), ("TriggerResults","","HLT")
triggerObjects, triggerObjectLabel  = Handle("std::vector<pat::TriggerObjectStandAlone>"), "selec
triggerPrescales, triggerPrescaleLabel  = Handle("pat::PackedTriggerPrescales"), "patTrigger"
```

```
# open file (you can use 'edmFileUtil -d /store/whatever.root' to get the physical file name)
events = Events("root://eoscms//eos/cms/store/cmst3/user/gpetrucc/miniAOD/v1/TT_Tune4C_13TeV-pyth

for iev,event in enumerate(events):
    event.getByLabel(triggerBitLabel, triggerBits)
    event.getByLabel(triggerObjectLabel, triggerObjects)
    event.getByLabel(triggerPrescaleLabel, triggerPrescales)

    print "\nEvent %d: run %6d, lumi %4d, event %12d" % (iev,event.eventAuxiliary().run(), event.
    print "\n === TRIGGER PATHS ==="
    names = event.object().triggerNames(triggerBits.product())
    for i in xrange(triggerBits.product().size()):
        print "Trigger ", names.triggerName(i), ", prescale ", triggerPrescales.product().getPres

    print "\n === TRIGGER OBJECTS ==="
    for j,to in enumerate(triggerObjects.product()):
        to.unpackPathNames(names);
        print "Trigger object pt %6.2f eta %+5.3f phi %+5.3f  " % (to.pt(),to.eta(),to.phi())
        print "         collection: ", to.collection()
        print "         type ids: ", ", ".join([str(f) for f in to.filterIds()])
        print "         filters: ", ", ".join([str(f) for f in to.filterLabels()])
        pathslast = set(to.pathNames(True))
        print "         paths:   ", ", ".join([("%s*" if f in pathslast else "%s")%f for f in to.
    if iev > 10: break
```

Miscellanea

Primary vertices and BeamSpot

The BeamSpot is copied as is from the AOD in the **`offlineBeamSpot`** collection.

For the primary vertices, a collection **`offlineSlimmedPrimaryVertices`** is provided, that has the same content as the AOD `offlinePrimaryVertices` collection. Slimmed primary vertices differ from the AOD collection as follows:

- vertices don't contain track references, but the association of candidates to vertices is provided in the packed candidates themself
- the primary vertex covariance matrix for vertices beyond the first one are stored in reduced precision

In addition the score used to rank primary vertices is stored in `floatedmValueMap_offlineSlimmedPrimaryVertices__PAT`.

Pileup Summary Info

The MC truth information about pileup is available in the form of a vector of **`PileupSummaryInfo`** objects for the different bunch crossings:

- in 74X miniAOD version 1, the collection is `addPileupInfo` taken directly from AODSIM
- in 74X miniAOD version 2 and later, the collection is `slimmedAddPileupInfo` slimmed by not saving the detailed information for the out-of-time bunch crossings.

E_T^{miss} filters

Some filters for E_T^{miss} cleaning are run at MiniAOD production stage, and saved as a trigger results (an **`edm::TriggerResults`** of the **`RECO`** or **`PAT`** process, depeding on whether MiniAOD was produced simultaneously with RECO or as a separate step). Remember, **Trigger bits for each path** are stored as edm::TriggerResults, as standard in CMSSW. The E_T^{miss} cleaning filters are run in separate paths and the path success/failure is saved as the filter decision bit.

The individual filters can be accessed with the same code as for the HLT paths, except of course for the process name.

- For the RunIISpring15DR74 MC campaing, the process name in **PAT**.
- For Run2015B PromptReco Data, the process name is **RECO**.
- For Run2015B re-MiniAOD Data 17Jul2015, the process name is **PAT**.
- MET filters are available in PromptReco since run 251585; for the earlier run range (251162-251562) use the re-MiniAOD 17Jul2015
- Recommendations on how to use MET filers are given in https://twiki.cern.ch/twiki/bin/viewauth/CMS/MissingETOptionalFiltersRun2 . Note in particular that the HBHE noise filter must be re-run from MiniAOD instead of using the flag stored in the TriggerResults; this applies to all datasets (MC, PromptReco, 17Jul2015 re-MiniAOD)

The list of all the paths and filters included and their definitions can be found in PhysicsTools/PatAlgos/python/slimming/metFilterPaths_cff.py.

In addition, just like in AOD the HCal noise information is provided in a HcalNoiseSummary object **hcalnoise** . Since 76X version 1 miniAODs, also **CSCHaloData** , **BeamHaloSummary** have been included UPDATED

UPDATED 76X version 2 miniAODs include a bugfix to the EcalDeadCellTriggerPrimitiveFilter filter, and new filters to flag events with bad tracks or bad muons

Bunch spacing UPDATED

Since 76X version 1 miniAODs, the events contain a per-event `unsigned int` product **bunchSpacingProducer** saying whether the given event has 50ns or 25ns bunch spacing.

Fast-jet rhos UPDATED

76X miniAODs contain four generally-useful rho variables

- `fixedGridRhoFastjetAll` : computed from all PF candidates
- `fixedGridRhoFastjetCentral` : computed from all PF candidates with $|\ \ | < 2.5$
- `fixedGridRhoFastjetCentralNeutral` computed from all neutral PF candidates with $|\ \ | < 2.5$ (neutral hadrons and photons)
- `fixedGridRhoFastjetCentralChargedPileUp` computed from all PF charged hadrons associated to pileup vertices and with $|\ \ | < 2.5$

74X miniAODs contain:

- `fixedGridRhoFastjetAll`
- `fixedGridRhoFastjetCentralChargedPileUp` and `fixedGridRhoFastjetCentralNeutral` which, because of a problem in the implementation, ended up being computed from all PF candidates with $|\ \ | < 2.5$ irrespectively of the charge and pileup assignment

Advanced topics: re-clustering, event interpretation

The following example shows how to re-run jet reconstruction with a different algorithm (in the example using AK5 instead of miniAOD default that is ak4). The example also shows how to rerun b-tagging.

Show example CMSSW python configuration Hide example CMSSW python configuration

```
import FWCore.ParameterSet.Config as cms

from FWCore.ParameterSet.VarParsing import VarParsing
```

```
process = cms.Process("USER")

process.load("Configuration.StandardSequences.MagneticField_cff")
process.load("Configuration.Geometry.GeometryRecoDB_cff")
process.load("Configuration.StandardSequences.FrontierConditions_GlobalTag_cff")
from Configuration.AlCa.GlobalTag import GlobalTag
process.GlobalTag = GlobalTag(process.GlobalTag, 'auto:run2_mc')

## Events to process
process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(10) )

## Input files
process.source = cms.Source("PoolSource",
    fileNames = cms.untracked.vstring(
        '/store/relval/CMSSW_7_4_1/RelValTTbar_13/MINIAODSIM/PU50ns_MCRUN2_74_V8_gensim_740pre7-v
    )
)

## Output file
from PhysicsTools.PatAlgos.patEventContent_cff import patEventContent
process.OUT = cms.OutputModule("PoolOutputModule",
    fileName = cms.untracked.string('test.root'),
    outputCommands = cms.untracked.vstring(['drop *','keep patJets_selectedPatJetsAK5PFCHS_*_*'])
)
process.endpath= cms.EndPath(process.OUT)

################################################
## Remake jets
################################################

## Filter out neutrinos from packed GenParticles
process.packedGenParticlesForJetsNoNu = cms.EDFilter("CandPtrSelector", src = cms.InputTag("packe
## Define GenJets
from RecoJets.JetProducers.ak5GenJets_cfi import ak5GenJets
process.ak5GenJetsNoNu = ak5GenJets.clone(src = 'packedGenParticlesForJetsNoNu')

## Select charged hadron subtracted packed PF candidates
process.pfCHS = cms.EDFilter("CandPtrSelector", src = cms.InputTag("packedPFCandidates"), cut = c
from RecoJets.JetProducers.ak5PFJets_cfi import ak5PFJets
## Define PFJetsCHS
process.ak5PFJetsCHS = ak5PFJets.clone(src = 'pfCHS', doAreaFastjet = True)

################################################
## Remake PAT jets
################################################

## b-tag discriminators
bTagDiscriminators = [
    'pfCombinedInclusiveSecondaryVertexV2BJetTags'
]

from PhysicsTools.PatAlgos.tools.jetTools import *
## Add PAT jet collection based on the above-defined ak5PFJetsCHS
addJetCollection(
    process,
    labelName = 'AK5PFCHS',
    jetSource = cms.InputTag('ak5PFJetsCHS'),
    pvSource = cms.InputTag('offlineSlimmedPrimaryVertices'),
    pfCandidates = cms.InputTag('packedPFCandidates'),
    svSource = cms.InputTag('slimmedSecondaryVertices'),
    btagDiscriminators = bTagDiscriminators,
    jetCorrections = ('AK5PFchs', ['L1FastJet', 'L2Relative', 'L3Absolute'], 'None'),
    genJetCollection = cms.InputTag('ak5GenJetsNoNu'),
    genParticles = cms.InputTag('prunedGenParticles'),
    algo = 'AK',
    rParam = 0.5
```

```
)

getattr(process,'selectedPatJetsAK5PFCHS').cut = cms.string('pt > 10')

process.p = cms.Path(process.selectedPatJetsAK5PFCHS)

from PhysicsTools.PatAlgos.tools.pfTools import *
## Adapt primary vertex collection
adaptPVs(process, pvCollection=cms.InputTag('offlineSlimmedPrimaryVertices'))

process.options = cms.untracked.PSet(
        wantSummary = cms.untracked.bool(True), # while the timing of this is not reliable in uns
        allowUnscheduled = cms.untracked.bool(True)
)
```

Event interpretations can be easily run starting from miniAOD with the following differences with respect to standard EI

- The input candidates are the packedCandidates
- The selector should be CandPtrSelector so that the Ptr to the original candidates are always kept
- The projector to use should be CandPtrProjector

The following code shows a commented example of EI configuration.

Show example CMSSW python configuration Hide example CMSSW python configuration

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("EX")
# Input source
process.source = cms.Source("PoolSource",
                                fileNames = cms.untracked.vstring(
        '/store/relval/CMSSW_7_4_1/RelValTTbar_13/MINIAODSIM/PU50ns_MCRUN2_74_V8_gensim_740pre7-v
    )
)

process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(10) )

#select isolated  muons and electrons collections
#tune the requirements to whatever ID and isolation you prefer

process.selectedMuons = cms.EDFilter("CandPtrSelector", src = cms.InputTag("slimmedMuons"), cut =
       (pfIsolationR04().sumChargedHadronPt+
        max(0.,pfIsolationR04().sumNeutralHadronEt+
        pfIsolationR04().sumPhotonEt-
        0.50*pfIsolationR04().sumPUPt))/pt < 0.20 &&
        (isPFMuon && (isGlobalMuon || isTrackerMuon)) )'''
process.selectedElectrons = cms.EDFilter("CandPtrSelector", src = cms.InputTag("slimmedElectrons"
        gsfTrack.isAvailable() &&
        gsfTrack.hitPattern().numberOfLostHits(\'MISSING_INNER_HITS\') < 2 &&
        (pfIsolationVariables().sumChargedHadronPt+
        max(0.,pfIsolationVariables().sumNeutralHadronEt+
        pfIsolationVariables().sumPhotonEt-
        0.5*pfIsolationVariables().sumPUPt))/pt <0.15'''
## Do projections
process.pfCHS = cms.EDFilter("CandPtrSelector", src = cms.InputTag("packedPFCandidates"), cut = c
process.pfNoMuonCHS =  cms.EDProducer("CandPtrProjector", src = cms.InputTag("pfCHS"), veto = cms
process.pfNoElectronsCHS = cms.EDProducer("CandPtrProjector", src = cms.InputTag("pfNoMuonCHS"),

#Import RECO jet producer for ak4 PF and GEN jet
from RecoJets.JetProducers.ak4PFJets_cfi import ak4PFJets
from RecoJets.JetProducers.ak4GenJets_cfi import ak4GenJets
process.ak4PFJetsCHS = ak4PFJets.clone(src = 'pfNoElectronsCHS', doAreaFastjet = True)
process.packedGenParticlesForJetsNoNu = cms.EDFilter("CandPtrSelector", src = cms.InputTag("packe
process.ak4GenJetsNoNu = ak4GenJets.clone(src = 'packedGenParticlesForJetsNoNu')
```

```
# The following is make patJets, but EI is done with the above
process.load("Configuration.StandardSequences.MagneticField_cff")
process.load("Configuration.Geometry.GeometryRecoDB_cff")
process.load("Configuration.StandardSequences.FrontierConditions_GlobalTag_cff")
from Configuration.AlCa.GlobalTag import GlobalTag
process.GlobalTag = GlobalTag(process.GlobalTag, 'auto:run2_mc')

bTagDiscriminators = [
    'pfCombinedInclusiveSecondaryVertexV2BJetTags'
]

from PhysicsTools.PatAlgos.tools.jetTools import addJetCollection
addJetCollection(
    process,
    labelName = 'AK4PFCHS',
    jetSource = cms.InputTag('ak4PFJetsCHS'),
    pvSource = cms.InputTag('offlineSlimmedPrimaryVertices'),
    pfCandidates = cms.InputTag('packedPFCandidates'),
    svSource = cms.InputTag('slimmedSecondaryVertices'),
    btagDiscriminators = bTagDiscriminators,
    jetCorrections = ('AK4PFchs', ['L1FastJet', 'L2Relative', 'L3Absolute'], 'None'),
    genJetCollection = cms.InputTag('ak4GenJetsNoNu'),
    genParticles = cms.InputTag('prunedGenParticles'),
    algo = 'AK',
    rParam = 0.4
)

#adjust PV used for Jet Corrections
process.patJetCorrFactorsAK4PFCHS.primaryVertices = "offlineSlimmedPrimaryVertices"

#new PAT default running is "unscheduled" so we just need to say in the outputCommands what we wa
process.options = cms.untracked.PSet(
        allowUnscheduled = cms.untracked.bool(True)
)

process.OUT = cms.OutputModule("PoolOutputModule",
    fileName = cms.untracked.string('test.root'),
    outputCommands = cms.untracked.vstring(['drop *','keep patJets_patJetsAK4PFCHS_*_*','keep *_*
)

process.endpath= cms.EndPath(process.OUT)
```

Producing MiniAOD UPDATED

To run miniAOD, use cmsDriver to create a cfg file, specifying the kind of data and the global tag

```
cmsDriver.py miniAOD-prod -s PAT --eventcontent [ MINIAOD | MINIAODSIM ] --runUnscheduled [ --dat
```

(the **`--fast`** is needed on fast sim to switch off some MET filters)

for example, for 74X 50ns MC in `CMSSW_7_4_1`

```
cmsDriver.py miniAOD-prod -s PAT --eventcontent MINIAODSIM --runUnscheduled --mc --filein /store/
```

note: in 7.2.X and later, the **`--conditions`** option take global tags without the extra `::All` postifx that was there in older releases

The right and most recent global tag for a release can also be obtained with the `auto:` syntax in cmsDriver, for example `auto:run2_mc_50ns` automatically expands to the right GT for run2 MC with 50ns bunch spacing scenario. The full list of available tags is printed by cmsDriver if the **`--conditions`** option is omitted.

Phys14 and older global tags, from
https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideFrontierConditions#miniAOD_production are:

GT name	Comments
PHYS14_25_V2	Phys14 GT for 25ns in 72X (asymptotic alignment and calibration scenario), with the new Phys14 jet energy
PHYS14_25_V1	Phys14 GT for 25ns in 72X (asymptotic alignment and calibration scenario), with CSA14 jet energy corrections **as used in the central Phys14 production**
PLS170_V7AN1	CSA14 GT for 25ns in 70X (asymptotic alignment and calibration scenario), as POSTLS170_V7 + updated JEC
PLS170_V6AN1	CSA14 GT for 50ns in 70X (more pessimistic alignment and calibration scenario), as POSTLS170_V6 + updated JEC
GR_70_V2_AN1	CSA14 GT for data in 70X, as GR_R_70_V2 + updated JEC

This topic: CMSPublic > WorkBookMiniAOD2015
Topic revision: r98 - 2017-07-27 - DanielDiaz