## CONTENTS

1 About this documentation ........................................ 1

2 Philosophy ......................................................... 3
   2.1 Contents .................................................... 3
ABOUT THIS DOCUMENTATION

An online version of this documentation is available at http://readthedocs.org/docs/final-state-analysis/en/latest/. This manual uses the Sphinx documentation engine. The source can be found in docs/source directory. You can re-build the documentation by installing sphinx and then running make html in the docs directory.
The final state analysis package is built around the data format PATFinalState. This abstract which encapsulates all of the interesting information needed by the analyst. Another way of imagining the PATFinalState is that it represents a single row in an ntuple. The advantage of the single object is that is both lightweight and that it holds references to all of the interesting information in the event. Thus you can compose complex observables using only a single object, enabling many tasks to use the string cut parser. This allows new cuts to be implemented quickly, and without writing any C++ code.

2.1 Contents

2.1.1 Workflow

The analysis proceeds in three steps. First, a pat tuple containing the final states is produced from AOD content. The PAT tuple can then be analyzed (selections + plots) directly, using the PATFinalStateAnalysis tool. The analysis typically runs at about 500+ events/second. As an optional third step, the PATFinalStateAnalysis can produce a bare ROOT ntuple at any stage of the processing. One can then apply additional selections.

PAT Tuple production

The pat tuple production runs on sod events and runs at about two events per second. The PAT tuplization does the following:

• Compute and embed all object IDs
• Apply corrections and embed systematics
• Produce event weights
• Compose all possible final states of interest. Example: DoubleMu + Tau

The tuplization only needs to be done occasionally. The PAT tuple is configured in four places.

• PatTools/python/datadefs.py provides a global definition of all MC and data samples.
• PatTools/python/patTupleProduction.py defines the PAT production sequence, the corrections applied to different objects, and the PATFinalState production.
• PatTools/test/patTuple_cfg.py is the top-level cfg which builds the tuple.
• PatTools/test/submit_tuplization.py submits the jobs to condor.
2.1.2 FinalStateAnalysis Package Description

The Final State Analysis (FSA) package is a CMSSW analysis framework. Common utilities are organized as subpackages. Each analysis (Higgs to tau, SSDL, etc) should exists as a separate subpackage.


DataFormats

Definitions of custom, EDM persist-able data formats used by the framework.

DataAlgos

The DataAlgos package defines the algorithm implementations used by the member functions of the DataFormats package. This improves code reuse, eases backward compatibility, and improves compilation speed.

MetaData

This package holds “data about data.” It knows what samples exist, how to find them in DBS, and what their cross sections are. It also holds the central definition of plot styles used for different data samples. Also, reference type code (such as getting Higgs boson properties from lookup tables, etc, are hosted here).

RecoTools

The RecoTools package contains plugin modules and utilities for dealing with RECO and AOD content. This package is not FWKLITE compatible.

PatTools

The PAT tools package contains everything needed to build the FSA pat tuple. It is standalone.

Selectors

The Selectors package defines the “analyzeFinalStates” FWKLITE binary, which is the final analysis builder. It additionally contains additional helper classes to analyze PATFinalStates, as well as the python definitions of common cuts to be applied. New selections and plots should be defined in Selectors/python/selectors and Selectors/python/plotting, respectively.

Utilities

Contains various command–line tools and C++ functionality.

docs and recipe

The recipe section contains scripts which automate installation of related packages. The docs folder just contains all the documentation.
2.1.3 Analysis Specific Packages

Each analysis, which uses the above packages, is configured in a separate sub-package. In general, an analysis can/should have some variation of the following content:

- **python/selection.py**: Defines the selections and plots (from the Selectors package) used in the analysis.
- **test/analyze_cfg.py**: Defines the final ntuple production cfg. This is the steering file for the analyzeFinalStatesBinary.
- **test/submit_analysis.py**: Submits the analyze_cfg.py jobs to condor/GRID/etc.
- **test/plotting/***: Tools to analyze the final level ntuple and produce plots.

**TagAndProbe**

Tools for generating Tag and Probe like analysis for muons and taus.

**VHiggs**

Associated Higgs to tau analysis.

2.1.4 Installation

Current CMSSW versions: **4.2.8** or **5.2.5**. The installation instructions are the same for both. NB CMSSW_5_2_4 is not supported.

Steps:

Get a supported CMSSW release area:

```
scram pro -n MyWorkingAreaName CMSSW CMSSW_VERSION
```

Checkout the FinalStateAnalysis repository:

```
cd MyWorkingAreaName/src
git clone https://ekfriis@github.com/ekfriis/FinalStateAnalysis.git
```

Checkout the needed CMSSW tags:

```
cd FinalStateAnalysis/recipe/
# You need to have CVS access
kinit me@CERN.CH
# Make sure your CMSSW environment is set up
cmsenv
# Checkout needed packages and apply patches
./recipe.sh
```

Install the custom python virtualenv and extra packages:

```
./install_python.sh
```

This might take a while - the script will download and compile the Numpy library.

You should be able to check the packages installed in the new python virtualenv by setting up the environment:

```
source environment.sh
```

and using the “yolk” python tool to query the installed packages:
yolk -l

It should look something like this:

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyYAML</td>
<td>3.10</td>
<td>active</td>
</tr>
<tr>
<td>Python</td>
<td>2.6.4</td>
<td>active development</td>
</tr>
<tr>
<td>Rivet</td>
<td>1.4.0</td>
<td>active development</td>
</tr>
<tr>
<td>distribute</td>
<td>0.6.24</td>
<td>active</td>
</tr>
<tr>
<td>elementtree</td>
<td>1.2.6-20050316</td>
<td>active development</td>
</tr>
<tr>
<td>ipython</td>
<td>0.12</td>
<td>active</td>
</tr>
<tr>
<td>matplotlib</td>
<td>1.0.1</td>
<td>active</td>
</tr>
<tr>
<td>numpy</td>
<td>1.6.1</td>
<td>active</td>
</tr>
<tr>
<td>pip</td>
<td>1.0.2</td>
<td>active</td>
</tr>
<tr>
<td>pyMinuit2</td>
<td>0.0.1</td>
<td>active development</td>
</tr>
<tr>
<td>python-ldap</td>
<td>2.3.5</td>
<td>active development</td>
</tr>
<tr>
<td>scipy</td>
<td>0.8.0</td>
<td>active development</td>
</tr>
<tr>
<td>setuptools</td>
<td>0.6.11</td>
<td>active</td>
</tr>
<tr>
<td>termcolor</td>
<td>1.1.0</td>
<td>active</td>
</tr>
<tr>
<td>uncertainties</td>
<td>1.8</td>
<td>active</td>
</tr>
<tr>
<td>wsgiref</td>
<td>0.1.2</td>
<td>active development</td>
</tr>
<tr>
<td>yolk</td>
<td>0.4.3</td>
<td>active</td>
</tr>
</tbody>
</table>

2.1.5 Troubleshooting

A summary of frequently encountered errors and their solutions.

Step ZERO

If something doesn’t work, try running `cmsenv` and `source FinalStateAnalysis/environment.sh` and then try again.

General CMSSW problems

ERROR: Unable to find release area

You have the wrong `SCRAM_ARCH` set. You need to do:

```
export SCRAM_ARCH=slc5_amd64_gcc434
```

for CMSSW_4_X_Y and:

```
export SCRAM_ARCH=slc5_amd64_gcc462
```

for CMSSW_5_X_Y. Note these will change for future CMSSW versions!

Getting the code (git problems)

These are generally problems that happen when you are trying to checkout or update the code (`git pull origin master`).

fatal: Not a git repository

This error occurs when you try to `git pull` outside of the FinalStateAnalysis directory (which is the “root” of the git repository). Solution is to run the command in that directory.
error: Your local changes to the following files would be overwritten by merge

This means that you have files changed in your working area that are also changed in the code update you are trying to pull. You can see locally modified files by running `git status`.

You have two options:

- **Commit your changes** `git add [the files],git commit -m "my commit message", and then pull again.`
- **Reset (lose/throw away/discard) your changes (danger!)** `git checkout -- [files]`
- **“Stash” your changes (advanced)** `git stash`

Permission denied (publickey).

You need to create a github.com account, and generate a SSH key pair. Instructions to do this are at Generating SSH keys:

**Runtime Errors**

First, always make sure that you did:

```
source environment.sh
```

in the `FinalStateAnalysis` directory to setup your environment.

**ImportError: No module named ...**

Generally this means that you have either not setup your environment, or not compiled the area. (`scram b -j 6` in the `src/` directory)

**ImportError: No module named rootpy**

This is a special case, generally only needed for plotting - `rootpy` is an external package that needs to be downloaded and installed:

```
cd $CMSSW_BASE/src/FinalStateAnalysis/recipe/external/src
git clone git@github.com:ekfriis/rootpy.git
cd $CMSSW_BASE/src/FinalStateAnalysis/recipe
./install_python.sh
```

**2.1.6 PAT Tuple Content**

**Skim**

Before tuplization, the events are skimmed at the AOD level. The OR of the following requirements is applied:

- One global muon with pt > 20 and eta < 2.4
- One electron with pt > 20 and eta < 2.5
- One global muon with pt > 14, eta < 2.4 and one tau pt > 18, eta < 2.3
PF Isolation

The PF isolation values (0.4) are available for electrons and muons via:

- chargedHadronIso()
- neutralHadronIso()
- photonIso()
- pfPUChargedHadrons() - for applying the Delta Beta correction

The H2Tau analysis uses custom veto cones, defined at the H2Tau working twiki. You can get the H2Tau isolations by:

Electrons:
- userIso(0) - all PF charged particles vetos: EB 0.01, EE 0.01
- userIso(1) - PF photon isolation vetos: EE + EB 0.08
- userIso(2) - PF PU isolation vetos: none (maybe this isn’t right??)

Muons:
- userIso(0) - all PF charged particles vetos: EB 0.001, EE 0.001

Note that you probably need to update your PAT tags in 42X to get the required version of DataFormats/PatCandidates.

For convenience, the fastjet energy is embedded in electrons and muons:

- userFloat(’rho’) - uses deterministic Voronoi rho produced during tau ID
- userFloat(’zzRho’) - uses ZZ recipe

Muons

Collection: cleanPatMuons

The following cut-based muon IDs are embedded:

- userInt(’WWID’)
- userInt(’WWID2011’)
- userInt(’VBTF’)
- userInt(’tightID’) - 2012 Muon POG recommendation

The WWID2011 is the same as defined in the UWAnalysis packages.

You can get a ref to to the associated PFMuon via:

- pfCandidateRef(),

if this ref isNull(), there is no muon ID’d by PF.

The following IP information is embedded as userFloats:

- ipDXY
- dz
The following systematics candidates are embedded (as userCands). The energy scale uncertainty is taken from the muon MuscleFit.

- uncorr (no muon energy scale, same as pat::Muon p4)
- corr (nominal ES correction)
- mes− (down 1 sigma)
- mes+ (up 1 sigma)

The closest PF patJet is available via the userCand('patJet') function. This ref may be null if the closest jet is farther than DR=0.5! The jet pt is stored as userFloat('jetPt'). If the jet doesn’t exist, the “jet pt” is equal to the muon Pt. The distance to the jet is userFloat('jetDR').

**Electrons**

Collection: cleanPatElectrons

The following electron IDs are embedded as userFloats:

- wp80
- wp90
- wp95
- WWID
- MITID - the 2011 MVA ID by the MIT people

The following 2012 electron MVA IDs (see EGamma ID Recipe.) and RECO IDs are embedded as eIDs:

- cicTight --> eidTight
- cicLoose --> eidLoose
- cicMedium --> eidMedium
- electronID('mvaNonTrigV0')
- electronID('mvaTrigV0')

The following 2011 electron MVA ID related information is embedded:

- userFloat("hasConversion")
- userInt("missingHits") - number of missing hits
- userFloat("idDZ") - dz used for MVA id
- userFloat("MVA") - raw MVA value
- userFloat("MVAprReID") - pre-ID cuts used for the MVA
- userFloat("MITID") - MIT MVA ID working point binary value
An MVA working point for the electron ID is embedded, again from the H2Tau 2012 twiki. The ID working point is based on the “NonTrig” MVA.

- `userInt('mvaidwp')`

The following IP information is embedded as `userFloats`:

- `ipDXY`
- `dz`
- `vz`
- `ip3D`
- `ip3DS - significance`
- `tip`
- `tipS - significance`

The following systematics candidates are embedded (as `userCands`). The electron energy scale uncertainty is currently configured to be 6% (I think this is a fixme)

- `uncorr` (no muon energy scale)
- `ees-` (down 1 sigma)
- `ees+` (up 1 sigma)

returns a `reco::CandidatePtr` pointing to a `reco::GsfElectron`.

**Jets**

Collection: `selectedPatJets`

The following jet IDs are embedded into the PFJets as `userFloats`. They correspond to the official PFJet IDs listed on the JetMET twiki.

- `idLoose`
- `idMedium`
- `idTight`

The raw MVA-based PU jet IDs (see MVAMet) are embedded as:

- `userFloat('fullDiscriminant')`
- `userFloat('philv1Discriminant')`
- `userFloat('simpleDiscriminant')`

and the integer working points as:

- `userInt('fullIdLoose')`
- `userInt('philv1IdLoose')`
- `userInt('simpleIdLoose')`

where XXX is Loose, Medium or Tight.

Available b-tag discriminators:

- `jetBProbabilityBJetTagsAOD`
- `jetProbabilityBJetTagsAOD`
• trackCountingHighPurBJetTagsAOD
• trackCountingHighEffBJetTagsAOD
• simpleSecondaryVertexNegativeBJetTagsAOD
• simpleSecondaryVertexHighEffBJetTagsAOD
• simpleSecondaryVertexHighPurBJetTagsAOD
• combinedSecondaryVertexBJetTagsAOD
• combinedSecondaryVertexMVABJetTagsAOD
• softMuonBJetTagsAOD
• softMuonByPtBJetTagsAOD
• softMuonByIP3dBJetTagsAOD

Corrections

The L1FastJet, L2Relative, L3Absolute corrections are applied to MC & Data. The L2L3Residual corrections are additionally applied to Data. Reference: IntroToJEC twiki.

In simulation, a smearing correction (see PAS JME-10-014) is additionally computed to correct the simulated jet energy resolution. The energy corrections are applied after the smearing is done.

The uncorrected, and 1 sigma uncertainties on the JEC are available from the pat::Jets via;

• userCand("uncorr") - no corrections or smearing applied
• userCand("smeared") - applying GEN-DATA resolution correction
• userCand("smear+") - smear error up
• userCand("smear-") - smear error down
• userCand("jes+") - using the JES uncertainty from the CondDB
• userCand("jes-") - using the JES uncertainty from the CondDB
• userCand("ues+") - using the UES uncertainty of 10%
• userCand("ues-") - using the UES uncertainty of 10%

Taus

Collection: cleanPatTaus

The taus are HPS PFTaus.

Discriminators

The standard complement of discriminators are available.

• decayModeFinding
• byVLooseIsolation
• byLooseIsolation
• byMediumIsolation

2.1. Contents
• byTightIsolation
• byVLooseIsolationDeltaBetaCorr
• byLooseIsolationDeltaBetaCorr
• byMediumIsolationDeltaBetaCorr
• byTightIsolationDeltaBetaCorr
• byVLooseCombinedIsolationDeltaBetaCorr
• byLooseCombinedIsolationDeltaBetaCorr
• byMediumCombinedIsolationDeltaBetaCorr
• byTightCombinedIsolationDeltaBetaCorr
• byIsolationMVARaw
• byLooseIsolationMVA
• byMediumIsolationMVA
• byTightIsolationMVA
• againstElectronLoose
• againstElectronMedium
• againstElectronTight
• againstElectronMVA
• againstElectronMVA2raw
• againstElectronVLooseMVA2
• againstElectronLooseMVA2
• againstElectronMediumMVA2
• againstElectronTightMVA2
• againstMuonLoose
• againstMuonMedium
• againstMuonTight

The seed jets are available via the userCand(‘patJet’) function. The corrected jet pt is stored as userFloat(‘jetPt’). This always exists, as taus are seeded by jets.

The following IP information is embedded as userFloats:

• ipDXY
• dz
• vz
• ip3D
• ip3DS - significance
• tip
• tipS - significance

The following systematics candidates are embedded (as userCands). The tau energy scale uncertainty is currently configured to be 3%
MET

Collection: systematicsMET

The following four-vector systematics are embedded as userCands:

- userCand("type1") - Type 1 correct MET (jets only)
- userCand("mes+") - Muon scale uncertainty
- userCand("tes+") - Tau scale uncertainty
- userCand("jes+") - Jet scale uncertainty
- userCand("ues+") - Unclustered energy scale uncertainty

Charge conjugation is implied.

2.1.7 Generating PAT Tuples

Installation

Setup a 4_2_8_patch7 and/or 5_3_3 area:
First run cmsenv.

The JOB_ID label should be agreed upon before hand. In general, the JOB_ID is formatted as “YYYY-MM-DD-XTeV-PatTuple”.

Get the code:

git clone git://github.com/uwcms/FinalStateAnalysis.git
cd FinalStateAnalysis

Now add all the dependencies and compile (takes forever):

cd recipe
./recipe.sh
cd ../..
scram b -j 8

Setup your environment variables:

cd FinalStateAnalysis/
source environment.sh

Testing locally

You can run on a local file by doing:

cd FinalStateAnalysis/PatTools/test
./patTuple_cfg.py isMC=0 globalTag=$datagt inputFiles=file:$dataAODFile maxEvents=1000 outputFile=myTestFile.root
The critical options you need to pass are `globalTag` and `isMC`, which should be either `$datagt` or `$mcgt`, and 0 or 1, respectively. The AOD file to process must be set as `inputFiles`. Note that the global tag environment variables are setup automatically in the `environment.sh` file. Also, for quick test convenience, working AOD files for data and MC are set in the `$dataAODFile` and `$mcAODFile` variables.

**Batch submission**

Build the crab submitters:
```bash
cd PatTools/test
python submit_tuplization_crab.py JOB_ID
```

You can pass wildcards to `--samples` to submit only some samples. The wildcard matches the key name in Meta-Data/python/data{7,8}TeV.py

Example:
```bash
python submit_tuplization_crab.py JOB_ID --samples "Zjets*" "WplusJets*"
```

Will create submissions for Zjets and WplusJets only.

For UW group mega-submissions, add `--responsible YOUR_NAME`. YOUR_NAME should be either “Maria”, “Josh”, “Ian”, or “Evan”

The submit_tuplization_crab.py script will create a directory JOB_ID with a multicrab.cfg in it.

Now setup your grid stuff, submit the jobs, and say goodbye to your quota:
```bash
source /cms/sw/glite3_2_5/etc/profile.d/grid_env.sh
source /cms/sw/CRAB_2_8_1/crab.sh
multicrab -create
multicrab -submit
```

**Note on JSON lumi masks**

The JSON lumi masks are stored in RecoTools/data/masks. To get the latest lumi masks:
```bash
cd $fsa/RecoTools/data/masks
./update.sh
```

this will copy all the new lumi masks from the official AFS area. Now add and commit the golden (excluding Muon physics) ones to the repository and commit:
```bash
ls *txt | grep -v MuonPhys | xargs git add -f
git commit -m "Adding new JSON lumimasks"
```

**Computing the processed luminosity**

**If using farmout**

Get the processed runs/lumis for a data sample:
```bash
export PATH=/afs/hep.wisc.edu/cms/cmsprod/farmoutCmsJobs/:$PATH
jobReportSummary.py /path/to/sample/submit/dir/*xml --json-out my_lumis.json
```
If using CRAB

Get the processed lumi JSON file via:

crab -report

Query the lumi database

Once you have the processed luminosity, check the recorded integrated luminosity of those run-lumis:

lumiCalc2.py -i my_lumis.json recorded

See the LumiCalc twiki for more details.
### 2.1.8 Published PAT Tuples

May 5th, 2012

8 TeV

<table>
<thead>
<tr>
<th>AOD DBS</th>
<th>PAT DBS</th>
<th>Files</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>/DYJetsToLL_M-50_TuneZ2Star_8TeV-madgraph-tarball/Summer12-PU_S7_START52_V9-v2/AODSIM</td>
<td>/DYJetsToLL_M-50_TuneZ2Star_8TeV-madgraph-tarball/swanson-Zjets_M50_2012-05-28-8TeV-PatTuple-8a107b9-4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>109 51 56 48 56 53 58 51</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/DoubleElectron/Run2012A-PromptReco-v1/AOD</td>
<td>/DoubleElectron/Run2012B-PromptReco-v1/AOD</td>
<td>134 204 201 527 750 668 109 51 56 48 56 53 58 51</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/DoubleMu/Run2012B-PromptReco-v1/AOD</td>
<td>/tauola/Summer12-v2/AODSIM</td>
<td>40780</td>
<td>127092 8149913</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-110_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-110_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>1168 15</td>
<td>363868 2403794 3718035</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-115_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-115_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>165 299 299 165</td>
<td>334857 373739</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-120_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-120_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>12B_PromptReco_v1_a_2012-05-29-8TeV-PatTuple-67c1f94-a7f10efca7dd683ad59c7e946715fa59/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-125_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-125_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>12B_PromptReco_v1_a_2012-05-29-8TeV-PatTuple-67c1f94-a7f10efca7dd683ad59c7e946715fa59/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-130_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-130_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-135_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-135_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-140_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-140_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-145_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-145_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-150_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-150_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>N/A N/A</td>
</tr>
<tr>
<td>/GluGluToHToTauTau_M-155_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>/GluGluToHToTauTau_M-155_8TeV-powheg-pythia6/Summer12-PU_S7_START52_V9-v1/AODSIM</td>
<td>4729152ae17d7e409729a1d0d9e952d/USER</td>
<td>N/A N/A</td>
</tr>
</tbody>
</table>
2.1.9 Tools

These are various scripts that automate tedious tasks.

**pickEvents.py**

Location: PatTools/scripts/pickEvents.py

Given a json file which maps dataset names to lists of run/evt/lumis, create a file which will call the appropriate copyPickMerge commands.

The json file should have the following format:

```json
{
  "DATASET_ALIAS" : [ [run1, lumi1, evt1], [run2, lumi2, evt2] ],
  "ANOTHER_DATASET_ALIAS" : [ [run1, lumi1, evt1], [run2, lumi2, evt2] ]
}
```

The actual dataset corresponding to a dataset alias is mapped in the datadefs dictionary in PatTools/datadefs.py

Usage:

```bash
pickEvents.py json_file > pickers.sh
bash < pickers.sh
```

**printEvents.py**

Location: PatTools/scripts/printEvents.py

Companion to pickEvents (above). Prints out a nicely formatted list given the run-lumi-evt json file.

**deltaR.py**

This stupid thing just figures out the deltaR on the command line

Usage:

```bash
deltaR.py eta1,phi1 eta2,phi2
```

**dropLumiInfo.py**

If you merge a very tight skim together into one EDM root file, it still has the lumi information from every skimmed lumisection/run. This can take a lot of space O(1GB). This script just takes an input file and drops all the lumi and run auxiliary information from it.

Usage:

```bash
dropLumiInfo.py inputFiles=[input_file] outputFile=[output_file]
```

**addSelectedHPSTaus.py**

Fireworks (cmsShow) works on a collection basis, so it doesn’t work well w/ the PFTau discriminator model. This script just takes an input file, runs PFTau, and adds a collection “hpsLooseTaus”, which are taus which pass the decay mode and loose combined iso discriminators.

Usage:
addSelectedHPSTaus.py inputFiles=[input_file] outputFile=[output_file]

**trimJSON.py**

Location: Utilities/scripts/trimJSON.py

Apply a run selection to a JSON file.

Usage:

```
trimJSON.py -i json_in -o json_out [-firstRun X] [-lastRun Y]
```

**compareEventLists.py**

Location: Utilities/scripts

Compare to event lists and find the differences.

The file can be almost any format.

Usage:

```
compareEventLists.py file1 file2
```

**eventListJSONDump.py**

Location: Utilities/scripts

Dump a event list json file to a simple format.

[format] can be python (list of tuples) or colons (for edmCopyPickMerge)

Usage:

```
eventListJSONDump.py json_file [format]
```

### 2.1.10 List of EDM Plugins

**CandInfoPrinter**

Prints (to stdout) information about each candidate in a collection.

Example configuration:

```python
process.printElectrons = cms.EDAnalyzer(
    "CandInfoPrinter",
    src = cms.InputTag("cleanPatElectrons"),
    pt = cms.string("pt"),
    eta = cms.string("eta"),
    superclusterEta = cms.string("superCluster.eta"),
    mva = cms.string("userFloat('MVA')"),
    mitID = cms.string("userFloat('MITID')"),
)
```
2.1.11 Version Control

The Final State Analysis package is tracked using the Git version control system. Git is different than CVS in that it's distributed - each repository is local and can stand by itself. Git makes it easy to pass changes to other repositories. This means you can make commits offline, and without worrying about messing up the central version control system.

Links

- Git For Scientists
- The “central” repo

Downloading Code

To get the code, you can clone from the master repository:

git clone git@github.com:ekfriis/FinalStateAnalysis.git

Setting up your own online repo

Go to github.com and set up an account. Then “fork” the master repository. Now you have your own version at:

https://github.com/YOURNAME/FinalStateAnalysis

Now get a local copy of your remote version:

git clone git@github.com:YOURNAME/FinalStateAnalysis.git

You can now edit the code, and commit it as often as you like (see below). You can’t ever mess anyone else up with conflicts like in CVS so do it often. When you want to share it, “push” it to your github account:

git push origin master

Now you can request that this gets “pulled” into the master repository by going to your github.com site and clicking “Pull Request”.

Committing Code

To commit a file, first add it to the “index” of changes to be committed:

git add file1 [file2]

Once you’re ready to commit, run:

git commit -m "my commit message"