Table of Contents

PAT Selector Utilities **1**
- Recipe for the impatient 1
- Links to Selectors of Interest 1
- Selector 1
  - Accessing via index class 2
  - Turning cuts on and off 2
  - Masking cuts 2
  - Concrete Simple Example 2
    - Example Use Snippet 3
- Selector<edm::EventBase> 4
  - Concrete Complex Example 4
    - Example Use Snippet 6
    - Example Use Snippet, Masking A Cut 7
- Using Selector in the full framework 8
  - Using Selector to write selected objects 8

PAT Selector Utilities

Object selection in physics analysis is important to structure correctly. The PAT has a great deal of tools available, as well as string-parsed object selectors for single objects. However this can be limited if the user wants to study the cuts being applied. Furthermore, it is difficult to apply these tools in `FWLite`.

This page refers to using Selectors in the 3.3.1 release and later. For the 3.1.4 version, please see SWGuidePATSelectors31x.

To use this software, start with the version in WorkBookWhichRelease. The software resides in

`CMS.PhysicsTools/SelectorUtils`

Recipe for the impatient

See the most recent updates here.

Links to Selectors of Interest

- Here are a bunch of selectors used in commissioning:
  - Primary Vertex Selector
  - Calo Jet ID
  - PF Jet ID
- Here are a bunch of selectors from the October Exercise (will be updated soon, most likely):
  - Muon ID
  - Electron ID
  - W+Jets Selection

`Selector`

The `Selector` interface is provided to fill the gap present in using "user-level" cuts in an analysis scenario. The `Selector` interface can be used in the full framework, or in `FWLite`.

The base class is found here with an interface shown in this snippet:

```
template<class T>
class Selector : public std::binary_function<T, std::strbitset, bool>
```

The `Selector` class template is an implementation of the `std::binary_function`, with arguments of `T` (the class being operated on), `std::strbitset`, a string-indexed bitset with source code here, and returns a bool based on whether the object passed the selection.

This interface makes use of the `std::strbitset` implemented for `CMSSW`. It performs identically to an ordinary `std::bitvector` except that the only interface to the information is via `std::string` keys. Thus, the user can access cuts and values by name rather than by index (which is inherently dangerous and difficult to synchronize).

The `Selector` class template provides an API as follows:

- "Declare" the selection in a registry (via the `push_back` method).
  - This can include selection criteria of doubles, ints, or bools.
  - These selection criteria can also be accessed subsequently via string-index.
- "Select" the object via an overloaded `operator()` method.

    - ♦ This also returns a `strbitset` with the bits for each cut set, so the user can see what passed and what failed.
  - "Print" the selection criteria to give the user information about what passed and what failed.

Accessing via index class

It is now possible to use a smart index class for `strbitset`. This allows the user to keep the semantics of the string accessor, while having the speed benefit of an integer access.

The new accessors can be seen here and here.

The example here shows the usage.

Specifically, the user is now able to "cache" the index of a strbitset, and can then use that cached index instead of doing string comparisons.

Turning cuts on and off

In order to turn the cuts on and off, the calls are:

- Turn the cut on:

```
set("CutName", true);
```

- Turn the cut off:

```
set("CutName", false);
```

This will "auto-pass" events based on this criterion.

Masking cuts

The `std::strbitset` can be used like any other bitset, so a "mask" can be selected and used as follows:

```
std::strbitset ret = selector.getBitTemplate();
std::strbitset mask = selector.getBitTemplate();
mask[string("CutName")] = true;
bool result = ret | mask;
```

Concrete Simple Example

A concrete simple example can be found here. We will now step through the code in detail.

- Register the selection criteria and turn all the cuts "on".
- Get cached values of the strbitset indices (`muonPtIndex` and `metIndex`).

```
WSelector( edm::ParameterSet const & params ) :
    muonSrc_(params.getParameter<edm::InputTag>("muonSrc")),
    metSrc_ (params.getParameter<edm::InputTag>("metSrc"))
  {
    double muonPtMin = params.getParameter<double>("muonPtMin");
    double metMin = params.getParameter<double>("metMin");
    push_back("Muon Pt", muonPtMin );
    push_back("MET", metMin );
    set("Muon Pt");
    set("MET");
```

```
    muonPtIndex_ = index_type(&bits_, std::string("Muon Pt") );
    metIndex_    = index_type(&bits_, std::string("MET") );


    wMuon_ = 0;
    met_ = 0;
  }
```

- Define the selection:

```
// Here is where the selection occurs
  virtual bool operator()( edm::EventBase const & event, std::strbitset & ret){

    ret.set(false);

    // Handle to the muon collection
    edm::Handle<std::vector<pat::Muon> > muons;
    // Handle to the MET collection
    edm::Handle<std::vector<pat::MET> > met;

    // Get the objects from the event
    bool gotMuons = event.getByLabel(muonSrc_, muons);
    bool gotMET = event.getByLabel(metSrc_, met);

    // get the MET, require to be > minimum
    if ( gotMET ) {
      met_ = &met->at(0);
      if ( met_->pt() > cut(metIndex_,   double()) || ignoreCut(metIndex_) )
        passCut(ret, metIndex_);
    }

    // get the highest pt muon, require to have pt > minimum
    if ( gotMuons ) {
      if ( !ignoreCut(muonPtIndex_) )  {
if ( muons->size() > 0 ) {
          wMuon_ = &muons->at(0);
  if ( wMuon_->pt() > cut(muonPtIndex_, double()) || ignoreCut(muonPtIndex_) )
            passCut(ret, muonPtIndex_);
        }
      } else {
        passCut( ret, muonPtIndex_);
      }
    }

    setIgnored(ret);
    return (bool)ret;

  }
```

Example Use Snippet

An example snippet of how to use this is:

```
// Now get the W selector
  edm::ParameterSet wSelectorParams = fwliteParameters.getParameter<edm::ParameterSet>("wSelector
  WSelector wSelector( wSelectorParams );
  std::strbitset wSelectorReturns = wSelector.getBitTemplate();


  // book a set of histograms
  fwlite::TFileService fs = fwlite::TFileService("analyzePatBasics.root");
  TFileDirectory theDir = fs.mkdir("analyzeBasicPat");
  TH1F* muonPt_  = theDir.make<TH1F>("muonPt", "pt",    100,  0.,300.);
  TH1F* muonEta_ = theDir.make<TH1F>("muonEta","eta",   100, -3.,  3.);
  TH1F* muonPhi_ = theDir.make<TH1F>("muonPhi","phi",   100, -5.,  5.);
```

```
  // open input file (can be located on castor)
  TFile* inFile = TFile::Open( "file:PATLayer1_Output.fromAOD_full.root" );

  // ----------------------------------------------------------------------
  // Second Part:
  //
  //  * loop the events in the input file
  //  * receive the collections of interest via fwlite::Handle
  //  * fill the histograms
  //  * after the loop close the input file
  // ----------------------------------------------------------------------

  // loop the events
  unsigned int iEvent=0;
  fwlite::Event ev(inFile);
  for(ev.toBegin(); !ev.atEnd(); ++ev, ++iEvent){
    edm::EventBase const & event = ev;

    // break loop after end of file is reached
    // or after 1000 events have been processed
    if( iEvent==1000 ) break;

    // simple event counter
    if(iEvent>0 && iEvent%1==0){
      std::cout << "  processing event: " << iEvent << std::endl;
    }


    if ( wSelector(event, wSelectorReturns ) ) {


      pat::Muon const & wMuon = wSelector.wMuon();

      muonPt_ ->Fill( wMuon.pt()  );
      muonEta_->Fill( wMuon.eta() );
      muonPhi_->Fill( wMuon.phi() );

    }
  }
```

`Selector<edm::EventBase>`

There is also the possibility to create event selectors from the `Selector` interface. In software releases in the 3.3.x cycle, this can work on the entire event because there was the creation of a common base class for `edm::Event` and `fwlite::Event` (`edm::EventBase`).

This very trivially implements the `Selector` to a concrete instantiation.

Concrete Complex Example

Please see here for a complex and realistic example of using selectors in an analysis.

The same API is used for the event selector as for the object selectors, so the same sequence applies.

- Register the selection criteria:

```
WPlusJetsEventSelector::WPlusJetsEventSelector( edm::ParameterSet const & params ) :
  EventSelector(),
  muonTag_          (params.getParameter<edm::InputTag>("muonSrc") ),
  electronTag_      (params.getParameter<edm::InputTag>("electronSrc") ),
  jetTag_           (params.getParameter<edm::InputTag>("jetSrc") ),
```

```
  metTag_           (params.getParameter<edm::InputTag>("metSrc") ),
  trigTag_          (params.getParameter<edm::InputTag>("trigSrc") ),
  muTrig_           (params.getParameter<std::string>("muTrig")),
  eleTrig_          (params.getParameter<std::string>("eleTrig")),
  pvSelector_       (params.getParameter<edm::ParameterSet>("pvSelector") ),
  muonIdTight_      (params.getParameter<edm::ParameterSet>("muonIdTight") ),
  electronIdTight_  (params.getParameter<edm::ParameterSet>("electronIdTight") ),
  muonIdLoose_      (params.getParameter<edm::ParameterSet>("muonIdLoose") ),
  electronIdLoose_  (params.getParameter<edm::ParameterSet>("electronIdLoose") ),
  jetIdLoose_       (params.getParameter<edm::ParameterSet>("jetIdLoose") ),
  pfjetIdLoose_     (params.getParameter<edm::ParameterSet>("pfjetIdLoose") ),
  minJets_          (params.getParameter<int> ("minJets") ),
  muPlusJets_       (params.getParameter<bool>("muPlusJets") ),
  ePlusJets_        (params.getParameter<bool>("ePlusJets") ),
  muPtMin_          (params.getParameter<double>("muPtMin")),
  muEtaMax_         (params.getParameter<double>("muEtaMax")),
  elePtMin_         (params.getParameter<double>("elePtMin")),
  eleEtaMax_        (params.getParameter<double>("eleEtaMax")),
  muPtMinLoose_     (params.getParameter<double>("muPtMinLoose")),
  muEtaMaxLoose_    (params.getParameter<double>("muEtaMaxLoose")),
  elePtMinLoose_    (params.getParameter<double>("elePtMinLoose")),
  eleEtaMaxLoose_   (params.getParameter<double>("eleEtaMaxLoose")),
  jetPtMin_         (params.getParameter<double>("jetPtMin")),
  jetEtaMax_        (params.getParameter<double>("jetEtaMax")),
  jetScale_         (params.getParameter<double>("jetScale")),
  metMin_           (params.getParameter<double>("metMin"))
{
  // make the bitset
  push_back( "Inclusive"      );
  push_back( "Trigger"        );
  push_back( "PV"             );
  push_back( ">= 1 Lepton"    );
  push_back( "== 1 Lepton"    );
  push_back( "MET Cut"        );
  push_back( "Z Veto"         );
  push_back( "Conversion Veto");
  push_back( "Cosmic Veto"    );
  push_back( "= 0 Jets"       );
  push_back( "= 1 Jets"       );
  push_back( "= 2 Jets"       );
  push_back( "= 3 Jets"       );
  push_back( "= 4 Jets"       );
  push_back( ">=5 Jets"       );

  // turn everything on by default
  set( "Inclusive"      );
  set( "Trigger"        );
  set( "PV"             );
  set( ">= 1 Lepton"    );
  set( "== 1 Lepton"    );
  set( "MET Cut"        );
  set( "Z Veto"         );
  set( "Conversion Veto");
  set( "Cosmic Veto"    );
  set( "= 0 Jets"       );
  set( "= 1 Jets"       );
  set( "= 2 Jets"       );
  set( "= 3 Jets"       );
  set( "= 4 Jets"       );
  set( ">=5 Jets"       );

  dR_ = 0.3;

  if ( params.exists("cutsToIgnore") )
    setIgnoredCuts( params.getParameter<std::vector<std::string> >("cutsToIgnore") );
```

```
  retInternal_ = getBitTemplate();
}
```

- Define the selection. Here is a subset, as this one is quite long:

```
//// initialization not shown


  if ( ignoreCut("= 0 Jets") ||
       int>(cleanedJets_.size()) ==  0 ){
    passCut(ret,"= 0 Jets");
    // end if 0 tight jets }

  if ( ignoreCut("= 1 Jets") ||
       int>(cleanedJets_.size()) ==  1 ){
    passCut(ret,"= 1 Jets");
    // end if 1 tight jets }

  if ( ignoreCut("= 2 Jets") ||
       int>(cleanedJets_.size()) ==  2 ){
    passCut(ret,"= 2 Jets");
    // end if 2 tight jets }

  if ( ignoreCut("= 3 Jets") ||
       int>(cleanedJets_.size()) ==  3 ){
    passCut(ret,"= 3 Jets");
    // end if 3 tight jets }

  if ( ignoreCut("= 4 Jets") ||
       int>(cleanedJets_.size()) ==  4 ){
    passCut(ret,"= 4 Jets");
    // end if 4 tight jets }

  if ( ignoreCut(">=5 Jets") ||
       int>(cleanedJets_.size()) >=  5 ){
    passCut(ret,">=5 Jets");
    // end if >=5 tight jets }
```

Example Use Snippet

An example snippet of how to use this is shown here.

```
//cout << "Making event selector" << endl;
  WPlusJetsEventSelector wPlusJets( shyftParameters );
  std::strbitset ret = wPlusJets.getBitTemplate();

  //loop through each event
  for( ev.toBegin();
       ! ev.atEnd();
       ++ev) {
    ret.set(false);

    bool passed = wPlusJets(ev, ret);
    std::vector<reco::ShallowClonePtrCandidate> const & electrons =  wPlusJets.selectedElectrons(
    std::vector<reco::ShallowClonePtrCandidate> const & muons     =  wPlusJets.selectedMuons();
    std::vector<reco::ShallowClonePtrCandidate> const & jets      =  wPlusJets.cleanedJets();
    std::vector<reco::ShallowClonePtrCandidate> const & jetsBeforeClean = wPlusJets.selectedJets(

    string bit_;

    bit_ = "Trigger" ;
    bool passTrigger = ret[ bit_ ];
```

```
    bit_ = "== 1 Lepton";
    bool passOneLepton = ret[ bit_ ];


    bit_ = "Trigger" ;
    bool passTrigger = ret[ bit_ ];
    bit_ = "== 1 Lepton";
    bool passOneLepton = ret[ bit_ ];
    bit_ = "= 0 Jets";
    bool jet0 = ret[bit_];
    bit_ = "= 1 Jets";
    bool jet1 = ret[bit_];
    bit_ = "= 2 Jets";
    bool jet2 = ret[bit_];
    bit_ = "= 3 Jets";
    bool jet3 = ret[bit_];
    bit_ = "= 4 Jets";
    bool jet4 = ret[bit_];
    bit_ = ">=5 Jets";
    bool jet5 = ret[bit_];

    bool anyJets = jet1 || jet2 || jet3 || jet4 || jet5;

    if ( anyJets && passOneLepton && passTrigger ) {
      cout << "Nele = " << electrons.size() << ", Nmuo = " << muons.size() << ", Njets_all = " <<
    }

  } //end event loop

  //cout << "Printing" << endl;
  wPlusJets.print(std::cout);
```

The printout at the end will give you a cut-flow summary of the cuts, in the order that you specify. In this case, it will look something like this:

```
    0 :            Inclusive       1496
    1 :              Trigger       1496
    2 :                   PV       1494
    3 :          >= 1 Lepton        611
    4 :          == 1 Lepton        512
    5 :              MET Cut        512
    6 :               Z Veto        512
    7 :      Conversion Veto        512
    8 :          Cosmic Veto        512
    9 :             = 0 Jets          0
   10 :             = 1 Jets          3
   11 :             = 2 Jets         12
   12 :             = 3 Jets         59
   13 :             = 4 Jets        135
   14 :             >=5 Jets        303
```

Example Use Snippet, Masking A Cut

Let's now turn to the use case where you want to study the analysis before and after some event selection (for instance, to study the trigger bias). This is now trivial. The above code becomes:

```
//loop through each event
  for( ev.toBegin();
       ! ev.atEnd();
      ++ev) {

    std::strbitset ret = wPlusJets.getBitTemplate();
    std::strbitset mask = wPlusJets.getBitTemplate();
    mask[string("Trigger")] = true;
```

```
    bool passed = wPlusJets(ev, ret);

    if ( ret | mask ) {
       // PLOTTING CODE BEFORE TRIGGER SELECTION HERE

       if ( ret ) {
           // PLOTTING CODE AFTER TRIGGER SELECTION HERE
       }
    }
  } //end event loop

  cout << "Printing" << endl;
  wPlusJets.print(std::cout);
```

Using `Selector` in the full framework

There is a class templates to facilitate the usage of the `Selector` in the full framework, FWLiteFilterWrapper. An instance of this will run the `Selector` in a sequence, and return "true" if there are any that pass the selection. As with the other Candidate utilities, setting `filter = true` will actually filter the events.

An example use case would be, using the same `WPlusJetsEventSelector` as in the previous example:

```
#include "CMS.PhysicsTools/UtilAlgos/interface/FWLiteFilterWrapper.h"

typedef edm::FWLiteFilterWrapper<WPlusJetsEventSelector> EDWPlusJets;

DEFINE_FWK_MODULE(EDWPlusJets);
```

The `EDWPlusJets` module can then be run as normal in a CMSSW sequence, here shown with several options changed in the `WPlusJetsEventSelector` for demonstration.

```
process.topMuPlusJetsCalo = cms.EDFilter( 'EDWPlusJets',
                                    inputwplusjetsAnalysis.clone(
                                        jetPtMin = cms.double(25.0),
                                        muJetDR = cms.double(0.),
                                        cutsToIgnore = cms.vstring( ['MET Cut'        ,
                                                                     'Z Veto'         ,
                                                                     'Conversion Veto',
                                                                     'Cosmic Veto'    ,
                                                                     '>=1 Jets'       ,
                                                                     '>=2 Jets'       ,
                                                                     '>=3 Jets'       ,
                                                                     '>=4 Jets'       ,
                                                                     '>=5 Jets'
                                                                     ] ),
                                        muonIdTight = inputwplusjetsAnalysis.muonIdTight.clone(
                                            cutsToIgnore=cms.vstring(options.muonIdIgnoredCuts)
                                            )
                                        )
                                    )
```

Using `Selector` to write selected objects

There is a new feature in 4.2.x to write objects that are selected with a `Selector` into the EDM and filter (optionally) on the presence of non-zero size. The class template is EDFilterObjectWrapper.

An example is shown here to write jets that satisfy the PFJetIDSelectionFunctor. An example python snippet to use this is

```
from PhysicsTools.SelectorUtils.pfJetIDSelector_cfi import pfJetIDSelector
process.goodPatJetsPFlow = cms.EDFilter("PFJetIDSelectionFunctorFilter",
                                        filterParams = pfJetIDSelector.clone(),
                                        src = cms.InputTag("selectedPatJetsPFlow"),
                                        filter = cms.bool(True)
                                        )
```

The class `T` must satisfy

```
T::T( edm::ParameterSet const &);
T::operator() ( C::value_type const &);
```

where `C` is "vector-like" (it needs a dictionary since it writes a copy of the objects that pass).

-- SalvatoreRoccoRappoccio - 29-Oct-2009

This topic: CMSPublic > SWGuidePATSelectors
Topic revision: r14 - 2011-03-01 - SalvatoreRRappoccio