h **TopCommonDileptonEventSaver.h** 5.98 KiB

```
#ifndef TOPCOMMONDILEPTONEVENTSAVER_H_
#define TOPCOMMONDILEPTONEVENTSAVER_H_

#include "TopAnalysis/EventSaverFlatNtuple.h"
#include "TRandom3.h"
#include "TH2.h"
#include "TopEventSelectionTools/PlotManager.h"
#include "TopCommonDilepton/TTbarNNLORecursiveRew.h"

class TopCommonDileptonEventSaver : public top::EventSaverFlatNtuple {

public:

  ///-- Default - so root can load based on a name --///
  TopCommonDileptonEventSaver();

  ///-- Default - so we can clean up --///
  //~TopCommonDileptonEventSaver();//=0;

  ///-- Exposition of base class initializers (to avoid [-Woverloaded-virtual] hidden warnings --///
  using top::EventSaverFlatNtuple::initialize;

  ///-- Run once at the start of the job --///
  void initialize(std::shared_ptr<top::TopConfig> config, TFile* file, const std::vector<std::string>& extraE

  ///-- For parton level objects --///
  void initializePartonLevelEvent();
  void addCustomPartonLevelVariables();
  void saveTruthEvent(); //naming enforced by AnalysisTop

  ///-- For particle level object --///
  void initializeParticleLevelBranches();
  void addCustomParticleLevelVariables();
  void saveParticleLevelEvent(const top::ParticleLevelEvent& plEvent); //naming enforced by AnalysisTop

  ///-- Run for every event (in every systematic) that needs saving --///
  void initializeRecoLevelBranches();
  void addCustomRecoLevelVariables(const std::vector<std::string>& extraBranches);
  void saveEvent(const top::Event& event); //naming enforced by AnalysisTop

  ///-- Observable calculating functions --///
  float cos_theta_helicity(TLorentzVector top, TLorentzVector parent_t, TLorentzVector ttbar, TLorentzVector
  float cos_theta_transverse(TLorentzVector top, TLorentzVector parent_t, TLorentzVector ttbar, TLorentzVecto
  float cos_theta_raxis(TLorentzVector top, TLorentzVector parent_t, TLorentzVector ttbar, TLorentzVector lep
  float cos_phi(TLorentzVector lpos, TLorentzVector lneg, TLorentzVector top, TLorentzVector tbar, TLorentzVe

  //skip saving nominal branch - useful for splitting ntuples by systematics if need be
  int isBranchStored(top::TreeManager const *treeManager, const std::string &variableName);
  bool m_skipNominal;

  bool m_doNeutrinoWeighter;
  bool m_doEllipseMethod;
  bool m_doSonnenchein;
  bool m_store_weights_reco;
  bool m_store_weights_particle;
  bool m_store_weights_parton;

  TTbarNNLORecursiveRew* m_ttbarNNLO_Reweighter;


 private:

  ///-- Steering --///
  bool is_MC;
  std::shared_ptr<top::TopConfig> m_config;


  ///-- Custom reco level variables --///
  bool m_passSelection;

  std::vector<float> m_lep_pt;
  std::vector<float> m_lep_eta;
  std::vector<float> m_lep_phi;
  std::vector<float> m_lep_e;
  std::vector<int>   m_lep_pdgid;
  bool m_fakeEvent;
  std::vector<int>   m_lep_true_type;
```

```
    std::vector<int>   m_lep_true_origin;
    std::vector<char>  m_lep_true_isPrompt;
    std::vector<char>  m_lep_true_isChargeFl;
    std::vector<int>   m_lep_true_pdgid;
    float m_dilep_m;
    std::vector<int> m_jet_isbtagged_DL1r_85;
    std::vector<float> m_ljet_pt;
    std::vector<float> m_ljet_eta;
    std::vector<float> m_ljet_phi;
    std::vector<float> m_ljet_e;
    std::vector<float> m_bjet_pt;
    std::vector<float> m_bjet_eta;
    std::vector<float> m_bjet_phi;
    std::vector<float> m_bjet_e;
    float m_met_met;
    float m_met_phi;
    float m_t_pt;
    float m_t_eta;
    float m_t_phi;
    float m_t_m;
    float m_tbar_pt;
    float m_tbar_eta;
    float m_tbar_phi;
    float m_tbar_m;
    float m_ttbar_pt;
    float m_ttbar_eta;
    float m_ttbar_phi;
    float m_ttbar_m;
    int   m_top_reco_method;
    float m_cosphi;
    float m_cos_kaxis_p;
    float m_cos_kaxis_m;
    float m_cos_naxis_p;
    float m_cos_naxis_m;
    float m_cos_raxis_p;
    float m_cos_raxis_m;
    std::vector<float> m_generator_weights;


    ///-- Custom particle level variables --///
    std::vector<float> m_particle_lep_pt;
    std::vector<float> m_particle_lep_eta;
    std::vector<float> m_particle_lep_phi;
    std::vector<float> m_particle_lep_m;
    std::vector<float> m_particle_lep_pdgid;
    std::vector<float> m_particle_jet_pt;
    std::vector<float> m_particle_jet_eta;
    std::vector<float> m_particle_jet_phi;
    std::vector<float> m_particle_jet_m;
    std::vector<float> m_particle_jet_btagged;
    float m_particle_met_ex;
    float m_particle_met_ey;
    float m_particle_met_phi;
    float m_particle_met_met;
    float m_particle_t_pt;
    float m_particle_t_eta;
    float m_particle_t_phi;
    float m_particle_t_m;
    float m_particle_tbar_pt;
    float m_particle_tbar_eta;
    float m_particle_tbar_phi;
    float m_particle_tbar_m;
    float m_particle_ttbar_eta;
    float m_particle_ttbar_phi;
    float m_particle_ttbar_m;
    float m_particle_cosphi;
    float m_particle_cos_kaxis_p;
    float m_particle_cos_kaxis_m;
    float m_particle_cos_naxis_p;
    float m_particle_cos_naxis_m;
    float m_particle_cos_raxis_p;
    float m_particle_cos_raxis_m;
    std::vector<float> m_particle_generator_weights;


    ///-- Custom parton level variables --///
    float m_parton_weight_nnlo;
    float m_parton_l_pt;
    float m_parton_l_eta;
```

```
  float m_parton_l_phi;
  float m_parton_l_m;
  float m_parton_l_pdgid;
  float m_parton_lbar_pt;
  float m_parton_lbar_eta;
  float m_parton_lbar_phi;
  float m_parton_lbar_m;
  float m_parton_lbar_pdgid;
  float m_parton_t_afterFSR_pt;
  float m_parton_t_afterFSR_eta;
  float m_parton_t_afterFSR_phi;
  float m_parton_t_afterFSR_m;
  float m_parton_tbar_afterFSR_pt;
  float m_parton_tbar_afterFSR_eta;
  float m_parton_tbar_afterFSR_phi;
  float m_parton_tbar_afterFSR_m;
  float m_parton_ttbar_pt;
  float m_parton_ttbar_eta;
  float m_parton_ttbar_phi;
  float m_parton_ttbar_m;
  float m_parton_cosphi;
  float m_parton_cos_kaxis_p;
  float m_parton_cos_kaxis_m;
  float m_parton_cos_naxis_p;
  float m_parton_cos_naxis_m;
  float m_parton_cos_raxis_p;
  float m_parton_cos_raxis_m;
  bool  m_parton_is_tau;
  std::vector<float> m_parton_generator_weights;


  //ClassDef(top::TopCommonDileptonEventSaver, 1);
  ClassDef(TopCommonDileptonEventSaver, 0);
  };

  //}

#endif

```