```
////////////////////////////////////////////////////////////////////////////////
// File: SMTAnalyze.cpp
//
// Purpose:   Example of analysis of MC++ data
//            This exampe finds parton and particle jets
//            from t, tb, W+, W- and writes out an Ntuple file ttb.hbk
//
//            The file has separate Ntuples for jets from partons
//            from X and for jets from particles from X:
//            1= partons from t,  2= partons from tb
//            3= partons from W+, 4= partons from W-
//            5-8 same as 1-4 but with jets from particles
//
//            Each Ntuple has:
//            4-momenta from jets (each jet labeled a,b,...)
//            and also em E, full width and emwidth
//            in addition mass of all jets included
//            Because W and t are treated independently Ntuple file
//            contains duplicate information
//
// Created:    28-May-1998  Serban Protopopescu
// History:
//
//              Georg: Took analyze and modified it for smt purposes
//              Doug:
//              In addition to the changes mentioned below I added some more
//              isValid statements.
////////////////////////////////////////////////////////////////////////////////

#include "smt_analyze/SMTAnalyze.hpp"
#include "smt_analyze/unique.hpp"
#include "smt_analyze/si_det_ids.inc"
#include "framework/Registry.hpp"
//  includes for Event
#include "edm/Event.hpp"
//#include "edm/TKey.hpp" in header
#include "edm/IDSelector.hpp"
#include "edm/THandle.hpp"

//  includes for MC data
#include "d0_mcpp/MCKineChunk.hpp"
#include "mcpp/MCparticleList.hpp"
#include "mcpp/StableEmOrHadFilter.hpp"
#include "mcpp/FinalPartonFilter.hpp"
#include "mcpp/DisplacedVtxFilter.hpp"
#include "mcpp/MCjetConeAlgo1.hpp"
#include "mcpp/MCfactoryStd.hpp" // must replace with d0_mcpp/D0MCfactory for
                                 // persistency
// includes for SMT MC DIGI/HITS data
#include "simpp/SimSMTHitChunk.hpp"
#include "smtdata/SmtDataChunk.hpp"
#include "smtdata/SmtData.hpp"
#include "smtdata/SmtStrips.hpp"
#include "smtutil/StripAddress.hpp"



#include "simpp/SimEventInfoChunk.hpp"

#include "simpp/SimSFTHitChunk.hpp"
#include "simpp/SimCPSHitChunk.hpp"
#include "simpp/SimFPSHitChunk.hpp"
#include "simpp/SimMUONHitChunk.hpp"
#include "simpp/SimCAEPChunk.hpp"


//include barrel stuff added Georg

#include "smt_input/SMTBData.hpp"
#include "smt_input/SMTBDataChunk.hpp"
#include "smt_cluster/SMTBarClusData.hpp"
#include "smt_cluster/SMTBarClusDataChunk.hpp"
```



```
#include "smt_input/SMTBInterface.hpp"
#include "smt_input/SMTDInterface.hpp"

#include "smt_input/SMTDData.hpp"
#include "smt_input/SMTDDataChunk.hpp"
#include "smt_cluster/SMTDiskClusData.hpp"
#include "smt_cluster/SMTDiskClusDataChunk.hpp"

#include "smt_cluster/SMTBarrelClusters.hpp"
#include "smt_cluster/SMTDiskClusters.hpp"
#include "smt_cluster/SMTClusData.hpp"
#include "smt_cluster/SMTClust.hpp"

#include "smt_lpos/SMTBarrel1DPos.hpp"
#include "smt_lpos/SMTDisk1DPos.hpp"
#include "smt_lpos/SMTDetClusPos.hpp"
#include "smt_lpos/SMTPosCollect.hpp"

#include "smt_lpos/SMTPosBCollect.hpp"
#include "smt_lpos/SMTPosBCollectChunk.hpp"
#include "smt_hit/SMTGlblBCollect.hpp"
#include "smt_d0hit/SMTGlblBCollectChunk.hpp"

#include "smt_d0hit/SMTGlblDCollectChunk.hpp"
#include "smt_lpos/SMTPosDCollect.hpp"
#include "smt_lpos/SMTPosDCollectChunk.hpp"
#include "smt_hit/SMTGlblDCollect.hpp"

#include "smt_hit/SMTBarrelHits.hpp"
#include "smt_hit/SMTDiskHits.hpp"
#include "smt_hit/SMTHit.hpp"
#include "smt_hit/SMTXyzGlbl.hpp"

#include "rcp/RCP.hpp"


//  includes for ntuples and histograms
#include "HepTuple/HepHBookFileManager.h"
#include "HepTuple/HepNtuple.h"
#include "HepTuple/HepHBook.h"

//add smt_hitmatch
#include "smt_hitmatch/SMTStripRecoHitMatch.hpp"
#include "smt_hitmatch/MatchInterface.hpp"
#include "smt_digimatch/SMTStripMcHitMatch.hpp"
//#include "simpp/SimTkHit.hpp"

using namespace edm;
using namespace std;
using namespace fwk;
using namespace smthit;
using namespace smt_match;

////////////////////////////////////////////////////////////////////////////////

FWK_REGISTRY_IMPL(SMTAnalyze ,SMTAnalyze::version().c_str())

const string SMTAnalyze::version(){
    return "$Id: SMTAnalyze.cpp,v 1.4 1999/05/18 16:48:38 steinbru Exp $";
}



const int SMTAnalyze::_maxjets=10;
const int SMTAnalyze::_maxtraks=100;
const int SMTAnalyze::_maxhits=2000;
const int SMTAnalyze::_maxclust=2000;


SMTAnalyze::SMTAnalyze(Context* context):
  Package(context), Process(context), JobSummary(context),
  _count(0),_njets(0),_mcKey(),_smthitKey(),_smtdigiKey(), BClusDataKey(),
  BPosKey(),  BHitKey(),_nhit(0),_nclust(0)
```

```
{

  RCP rcp=packageRCP();  //  get RCP for SMTAnalyze package

  //  setup Hbook manager
  _hbkMgr= new HepHBookFileManager((rcp.getString("hbk_file")).c_str());
  cout<<endl<<"Initializing analysis package"<<endl;
  cout<<" ntuples are in "<<rcp.getString("hbk_file")<<endl;

  //   MC jet algorithm setup
  float conesize=rcp.getFloat("conesize");
  float ETseed=rcp.getFloat("ETseed");
  float ETmin=rcp.getFloat("ETmin");
  _make=new MCfactoryStd;   // must replace with D0MCfactory
  //                           if you want persistent jets
  _jalgo=new MCjetConeAlgo1(conesize,ETseed,ETmin,_make);

  //    Define Histograms and Ntuples

  //    define histograms to plot displaced vertex information
  _vdist=&_hbkMgr->hist1D("vertex distance",50,0.,1.);
  _vxy=&_hbkMgr->hist2D("vertex x y",50, -0.5, 0.5, 50, -0.5, 0.5);

  //    define Ntuples
  //    There are 8 Ntuple sets:
  //
  // (1) tpq: t-quark jets from partons
  // (2) tmq: antit-quark jets from partons
  // (3) wpq: W+ jets from partons
  // (4) wmq: W- jets from partons
  // (5) tpjets: t-quark jets from partons
  // (6) tmjets: antit-quark jets from partons
  // (7) wpjets: W+ jets from partons
  // (8) wmjets: W- jets from partons
  //
  //  Note that, because _hbkMgr returns references and one
  //  cannot have arrays of references, the references are
  //  copied to pointers instead.
  //  _ptuple[0]=&_hbkMgr->ntuple("tpq");
  //  _ptuple[1]=&_hbkMgr->ntuple("tmq");
  //  _ptuple[2]=&_hbkMgr->ntuple("wpq");
  //  _ptuple[3]=&_hbkMgr->ntuple("wmq");
  //  _ptuple[4]=&_hbkMgr->ntuple("tpjets");
  //  _ptuple[5]=&_hbkMgr->ntuple("tmjets");
  //  _ptuple[6]=&_hbkMgr->ntuple("wpjets");
  //  _ptuple[7]=&_hbkMgr->ntuple("wmjets");
  _ptuple[7]=&_hbkMgr->ntuple("bsvxclus");
  _ptuple[8]=&_hbkMgr->ntuple("bsvxhits"); //added by doug
  _ptuple[9]=&_hbkMgr->ntuple("dsvxhits"); //added by doug


  Book_SMTBHit(8); // changed by doug
  Book_MCBHit(8);  // changed by doug


  Book_SMTBDATA(9);
  Book_SMTBClust(7);

  //  Book_SMTDClust(11);

}
SMTAnalyze::~SMTAnalyze()
{
  delete _make;
  delete _jalgo;
  delete _hbkMgr;
}

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
///////////////////////PROCESS EVENT/////////////////////////////
Result SMTAnalyze::processEvent(Event &event)
```



```
{
  _count++;

  //  extract pointer to MC information from MCKineChunk

  cout<< event.collisionID()<<endl;

  THandle<MCKineChunk> mcchunk=_mcKey.find(event);

  if (mcchunk.isValid())

    {

      MCevt* ptevt=mcchunk->ptrMCevt();

      //just a test

      //   THandle<SMTBarClusDataChunk> bcdata = BClusDataKey.find(event);




      list<MCparticle*> listp;
      ptevt->getParticles(listp);   //  get complete list of particles

      //  get vertex of origin and displaced vertices
      //  vertex of origin of 1st particle in list is a primary vertex
      MCvtx* firstv=(listp.front())->vtx();
      //  get displaced vertices
      DisplacedVtxFilter disp;
      list<MCvtx*> listv;
      ptevt->getVertices(&disp,listv);
      //    fill histograms
      fill_hists(firstv, listv);


      //  ids for t, tb, W+ and W-
      int id[4];
      id[0]=6;
      id[1]=-6;
      id[2]=80;
      id[3]=-80;

      //  make jets for descendants of each id in turn
      for(int k=0; k<4; ++k){

        list<MCparticle*> lall;  // this list will contain all descendatns
        list<MCparticle*> lforj; // this list will contain subset for jets

        // find all descendants in listp of first particle with given id
        MCparticleList::findDescendants(listp,id[k],1,lall);

        //  find jets using only partons

        FinalPartonFilter fpfilter;   // select only final parton descendants
        // make container for parton descendants
        MCparticleList partons(&fpfilter, lall);
        partons.getParticles(lforj);  // get final parton list
        //      fill_ntuple(lforj,k);  // find jets and fill ntuples

        //  find jets using only particles that deposit energy in a calorimeter

        StableEmOrHadFilter emhfilter; // select em or had descendants
        // make container for em or had descendants
        MCparticleList emhads(&emhfilter, lall);
        emhads.getParticles(lforj);       // get em + had list
        //      fill_ntuple(lforj,k+4);   // find jets and fill ntuples
      }


    }
  else
```

```
      {
        cout<<"MCKineChunk pointer not valid for this event\n";
      }

    {

    THandle<SimSMTHitChunk> ptrsmtchunk=_smthitKey.find(event);

    if (ptrsmtchunk.isValid())

      {



        ptrsmtchunk->printChunk(cout);

        list<SimTkHitCollection*> tkhclist;
        ptrsmtchunk->getCollectionList(tkhclist);

        list<SimTkHitCollection*>::iterator jhc;
        int nhit=0;
        for(jhc=tkhclist.begin(); jhc !=tkhclist.end(); ++jhc){
          list<SimTkHit*> tkhitlist;
          string HCname;
          (*jhc)->getHCName(HCname);
          //    cout<<"Tracker hit collection name "<<HCname<<endl;
          float radius;
          (*jhc)->getHitList(tkhitlist);
          list<SimTkHit*>::iterator jhit;
          for(jhit=tkhitlist.begin(); jhit !=tkhitlist.end(); ++jhit){
            ++nhit;
            //      cout << " SMT ideal Geant hit # " << nhit << endl;
            vector<int> vindex;
            int itrack;
            (*jhit)->getTrackNumber(itrack);
            //      cout<<" track number "<<itrack<<endl;
            float xg,yg,zg,pxg,pyg,pzg;
            xg=0.;
            yg=0.;
            zg=0.;
            pxg=0.;
            pyg=0.;
            pzg=0.;
            (*jhit)->getGlobalXYZ(xg,yg,zg);
            (*jhit)->getGlobalMomentum(pxg,pyg,pzg);

            radius = sqrt(xg*xg+yg*yg);
            //cout<<" global x,y,z,r  "<<xg<<" "<<yg<<" "<<zg<<" "<<radius<<endl;
            //      cout<<"        px "<<pxg<<" py "<<pyg<<" pz "<<pzg<<endl;

            float xin,yin,zin,xout,yout,zout;
            float steplength,g3code,edep,tof;
            (*jhit)->getLocalXYZin(xin,yin,zin);
            (*jhit)->getLocalXYZout(xout,yout,zout);
            (*jhit)->getStep(steplength);
            (*jhit)->getG3ParticleCode(g3code);
            (*jhit)->getEnergyDeposit(edep);
            (*jhit)->getTOF(tof);
            //cout<<" local (in) x,y,z "<<xin<<" "<<yin<<" "<<zin<<endl;
            //cout<<"       (out) x,y,z "<<xout<<" "<<yout<<" "<<zout<<endl;
            //      cout<<"  step "<<steplength<<" g3code "<<g3code<<" edep "<<edep<<
+" tof "<<tof<<endl;

          }         // end of loop over hits
        }           // end of loop over hit collections

      }
    else
      {
        cout<<"SimSMTHitChunk pointer not valid for this event\n";
      }

    }
```



```


    THandle<SmtDataChunk> ptrsmtchunk=_smtdigiKey.find(event);
    ptrsmtchunk->printChunk(cout);
        //  --- digi information ---
    //  cout<<" "<<endl;
    // cout<<"--- digitized information ---"<<endl;



    if (ptrsmtchunk.isValid())
      {

        SmtData* smtdata = ptrsmtchunk->getSmtData();

        //cout << " --------- Contents of SmtData ------------";
        //cout << smtdata ;
        //cout << " --------- Print all Strips ---------------";

        /*
          list<SmtStrips> smtStrips;

          smtStrips = smtdata->getAllStrips();
          std::list<SmtStrips>::iterator jstr;
          for (jstr=smtStrips.begin(); jstr!=(smtStrips).end(); jstr++){
          cout   << (*jstr) << endl;
          }
          */
        SMTStripMcHitMatch minter;
        vector<int> mctracks;
        vector<SimTkHit*> mchits;

          list<SmtStrips> smtStrips;
          std::list<SmtStrips>::iterator jstr;
          int deltastr=0;
          int stripl=0;
          int view1=0;
          float barrel[_maxclust];
          float layer[_maxclust];
          float ladder[_maxclust];
          float nstrip[_maxclust];
          float delstrip[_maxclust];
          float side[_maxclust];
          float type[_maxclust];
          float pulse[_maxclust];
          float nmchits[_maxclust];
          int nclust;
          int clust_in_progress=0;
          for (int iii = 0; iii<_maxhits; iii++)
            {
              barrel[iii]=0.0;
              layer[iii]=0.0;
              ladder[iii]=0.0;
              nstrip[iii]=0.0;
              type[iii]=0.0;
              delstrip[iii]=0.0;
              pulse[iii]=0.0;
              side[iii]=-1.0;
              nmchits[iii]=0.0;
            }
          int stripNum;
          float pulseHeight, totpulse=0;
          int view;
          int count =0;
          int idet = 1;
          bool change_view;
          vector<int> StrList;
          int ibar, ilay, ilad;
          for (ibar=1; ibar<=6; ibar++){
          for (ilay=1; ilay<=8; ilay++){

          if (ilay <=4) {
            for (ilad=1; ilad<=6; ilad++){
```

```
          //      cout << " debug 1 " << endl;
          smtStrips =smtdata->getModuleStrips(idet,ibar,ilay,ilad);
          //      cout << " debug 2 " << endl;
          int nstr=0;
          if (smtStrips.size()!=0)
          {   jstr=smtStrips.begin();
          StripAddress address = jstr->getAddress();
          int stripNum = address.Strip();
          stripl = stripNum;
          view1 = address.View();
          }
          cout << "det" << idet << " " << ibar <<" ";
          cout<< ilay << " " << ilad << endl;
          for (jstr=smtStrips.begin(); jstr!=(smtStrips).end(); jstr++){
            StripAddress address = jstr->getAddress();
            int adet=address.DetType();
            if (adet!= idet ) {
              cout << " det wrong... " << idet << " " << adet << endl;
              cout << "det " << idet << " " << ibar <<" ";
              cout<< ilay << " " << ilad << endl;
            }
            stripNum  = address.Strip();
            pulseHeight = jstr->getData();
            view = address.View();
            cout << " str# " <<  stripNum << " " <<  pulseHeight;
            cout << " " << view << endl;
            change_view=false;
            if (view !=view1) {
              stripl = stripNum;
              view1 = address.View();
              change_view=true;
            }
            int spid = address.get_StripAddress();

            deltastr=stripNum-stripl;
            if (pulseHeight<0.1) {deltastr=999;}
            if (deltastr < 0) {
              cout << " Negative delta ! " <<  deltastr;
              cout << " " << stripNum << " " << stripl << endl;
            }
            else {
              if (deltastr>1 ) {
                if (nstr>0){
                count++;
                cout << " nstrips " << nstr << "  " << deltastr << endl;
                nstrip[count]=nstr;
                layer[count]=ilay;
                ladder[count]=ilad;
                barrel[count]=ibar;
                delstrip[count]=deltastr-1;
                if (deltastr==999){delstrip[count]=1;}
                side[count]=view;
                pulse[count]=totpulse;
                vector<int> mlist;
                for (int iik=0; iik<StrList.size(); iik++){
                  mctracks = minter.get_mcids(StrList[iik]);
                  cout << " Mc tracks size " << mctracks.size()<< endl;
                  mlist.insert(mlist.begin(),mctracks.begin(),mctracks.end());
                }
                unique(mlist);
                nmchits[count]=nmchits[count]+mlist.size();
                cout << " total Mc tracks " << nmchits[count] << endl;
                }
                StrList.resize(0);
                nstr=0;
                totpulse=0;
                if (deltastr!=999) {
                  nstr=1;
                  StrList.push_back(spid);
                  cout << " Strip list size " << StrList.size() << endl;
                  totpulse=pulseHeight;
                }
                clust_in_progress=0;
```



```
              }
              else {
                if (change_view){
                  count++;
                  cout << " nstrips " << nstr << "  " << deltastr << endl;
                  nstrip[count]=nstr;
                  layer[count]=ilay;
                  ladder[count]=ilad;
                  barrel[count]=ibar;
                  delstrip[count]=-1.;
                  side[count]=view;
                  pulse[count]=totpulse;
                  vector<int> mlist;
                  for (int iik=0; iik<StrList.size(); iik++){
                    mctracks = minter.get_mcids(StrList[iik]);
                    cout << " Mc tracks size " << mctracks.size()<< endl;
                    mlist.insert(mlist.begin(),mctracks.begin(),mctracks.end());
                  }
                  unique(mlist);
                  nmchits[count]=nmchits[count]+mlist.size();
                  cout << " total Mc tracks " << nmchits[count] << endl;
                  StrList.resize(0);
                  cout << " Strip list size " << StrList.size() << endl;
                  nstr=0;
                  totpulse=0;
                  clust_in_progress=0;
                }
                nstr++;
                totpulse=totpulse+pulseHeight;
                //              cout << " deltastrip - 0 or 1 - " << deltastr <<
+" " << nstr;
                //              cout << endl;
                StrList.push_back(spid);
                clust_in_progress=1;
              }
            }
            //              cout << " deltastrip " << deltastr << " " << nstr;
            //              cout << " " << stripNum << " " << stripl << endl;
            stripl=stripNum;
          }
          if (clust_in_progress!=0) {
            count++;
            cout << " nstrips " << nstr << "  " << deltastr << endl;
            nstrip[count]=nstr;
            layer[count]=ilay;
            ladder[count]=ilad;
            barrel[count]=ibar;
            delstrip[count]=-1.;
            side[count]=view;
            pulse[count]=totpulse;
            vector<int> mlist;
            for (int iik=0; iik<StrList.size(); iik++){
              mctracks = minter.get_mcids(StrList[iik]);
              cout << " Mc tracks size " << mctracks.size()<< endl;
              mlist.insert(mlist.begin(),mctracks.begin(),mctracks.end());
            }
            unique(mlist);
            nmchits[count]=nmchits[count]+mlist.size();
            cout << " total Mc tracks " << nmchits[count] << endl;
            StrList.resize(0);
            cout << " Strip list size " << StrList.size() << endl;
            nstr=0;
            totpulse=0;
            clust_in_progress=0;
          }
        }
      }
      else{
        cout << " cluster in progress " << clust_in_progress << endl;
        cout << " Total Number of clusters now " << count << endl;
        for (ilad=1; ilad<=12; ilad++){
          //      cout << " debug 3 " << endl;
          smtStrips =smtdata->getModuleStrips(idet,ibar,ilay,ilad);
```

```
        //    cout << " debug 4 " << endl;
        int nstr=0;
        if (smtStrips.size()!=0)
        {   jstr=smtStrips.begin();
        StripAddress address = jstr->getAddress();
        int stripNum = address.Strip();
        stripl = stripNum;
        view1 = address.View();
        }
        cout << "det" << idet << " " << ibar <<" ";
        cout<< ilay << " " << ilad << endl;
        for (jstr=smtStrips.begin(); jstr!=(smtStrips).end(); jstr++){
          StripAddress address = jstr->getAddress();
          //    cout << " debug 5 " << endl;
          int adet=address.DetType();
          //    cout << " debug 6 " << endl;
          if (adet!= idet ) {
          cout << " det wrong... " << idet << " " << adet << endl;
          cout << "det " << idet << " " << ibar <<" ";
          cout<< ilay << " " << ilad << endl;
          }
          stripNum = address.Strip();
          //    cout << " debug 7 " << endl;
          pulseHeight = jstr->getData();
          //    cout << " debug 8 " << endl;
          view = address.View();
          //    cout << " debug 9 " << endl;
          cout << " str# " <<  stripNum << " " <<  pulseHeight ;
          cout << " " << view << endl;
          change_view=false;
          if (view !=view1) {
            stripl = stripNum;
            view1 = address.View();
            change_view=true;
          }
          int spid = address.get_StripAddress();
          /*
          int spid = address.get_StripAddress();
          cout << " " <<  stripNum << " " << stripl << " " << view;
          cout << " " << hex << address.get_StripAddress() << dec<<endl;

          SMTStripMcHitMatch minter;
          vector<int> mctracks;
          vector<SimTkHit*> mchits;
          mchits = minter.get_hits(spid);
          mctracks = minter.get_mcids(spid);
                  vector<int> smtdmctracks;
          jstr->getTrackNumber(smtdmctracks);
          cout << " MCtracks " << mctracks.size() << " trk " ;
          for (int ijk=0; ijk<mctracks.size(); ijk++){
            cout << mctracks[ijk] << " ";
          }
          cout <<endl;
          cout << " SMTData MCtracks "<<smtdmctracks.size() << "  trk" ;
          for (int ijk=0; ijk<smtdmctracks.size(); ijk++){
            cout << smtdmctracks[ijk] << " ";
          }
          cout <<endl;
          cout << " MCHits ";
          for (int ijk=0; ijk<=mchits.size(); ijk++){
            cout << mchits[ijk] << " ";
          }
          cout <<endl;
          int nmt =  mchits.size();
          cout <<"We  found  "<<nmt<<"mchits for this Barrel strip\n";
          vector<SimTkHit*>::const_iterator im;
          //loop over MC hits
          float  pxg;
          float  pyg;
          float  pzg;
          int itrack, idet;
          int nprim=0;
          for(im=mchits.begin();im!=mchits.end();im++){
```



```
           (*im)->printHit(cout);
           (*im)->getGlobalMomentum(pxg,pyg,pzg);
           (*im)->getTrackNumber(itrack);
           bool okg = (*im)->is_ParticlePrimary();
           if (okg) {nprim++;}
           (*im)->getDetId(idet);
           cout <<"Global coord for hit from track ";
           cout << itrack << " " << idet << " " <<okg << " : "<<pxg<< ", ";
           cout << pyg << ", "<<pzg <<endl;
          }
          if (nprim!=1) {cout << " Nprimary " << nprim << endl;
          }

          */

          deltastr=stripNum-stripl;
          if (pulseHeight<0.1){deltastr=999;}
          if (deltastr < 0) {
            cout << " Negative delta ! " <<  deltastr;
            cout << " " << stripNum << " " << stripl << endl;
          }
          else {
            if (deltastr>1 ) {
              if (nstr>0){
              count++;
              cout << " nstrips " << nstr << "  " << deltastr << endl;
              nstrip[count]=nstr;
              layer[count]=ilay;
              ladder[count]=ilad;
              barrel[count]=ibar;
              delstrip[count]=deltastr-1;
              if (deltastr==999){delstrip[count]=1;}
              side[count]=view;
              pulse[count]=totpulse;
              vector<int> mlist;
              for (int iik=0; iik<StrList.size(); iik++){
                mctracks = minter.get_mcids(StrList[iik]);
                cout << " Mc tracks size " << mctracks.size()<< endl;
                mlist.insert(mlist.begin(),mctracks.begin(),mctracks.end());
              }
              unique(mlist);
              nmchits[count]=nmchits[count]+mlist.size();
              cout << " total Mc tracks " << nmchits[count] << endl;
              }
              nstr=0;
              StrList.resize(0);
              totpulse=0;
              if (deltastr!=999) {
                nstr=1;
                StrList.push_back(spid);
                cout << " Strip list size " << StrList.size() << endl;
                totpulse=pulseHeight;
              }
              clust_in_progress=0;
            }
            else {
              if (change_view){
                count++;
                cout << " nstrips " << nstr << "  " << deltastr << endl;
                nstrip[count]=nstr;
                layer[count]=ilay;
                ladder[count]=ilad;
                barrel[count]=ibar;
                delstrip[count]=-1.;
                side[count]=view;
                pulse[count]=totpulse;
                vector<int> mlist;
                for (int iik=0; iik<StrList.size(); iik++){
                  mctracks = minter.get_mcids(StrList[iik]);
                  cout << " Mc tracks size " << mctracks.size()<< endl;
                  mlist.insert(mlist.begin(),mctracks.begin(),mctracks.end());
                }
                unique(mlist);
```

```
                    nmchits[count]=nmchits[count]+mlist.size();
                    cout << " total Mc tracks " << nmchits[count] << endl;
                    StrList.resize(0);
                    cout << " Strip list size " << StrList.size() << endl;
                    nstr=0;
                    totpulse=0;
                    clust_in_progress=0;
                  }
                  nstr++;
                  totpulse=totpulse+pulseHeight;
                  clust_in_progress=1;
                  //               cout << " deltastrip - 0 or 1 - " << deltastr <<
+" " << nstr;
                  //               cout << endl;
                  StrList.push_back(spid);
                }
              }
              //               cout << " deltastrip " << deltastr << " " << nstr;
              //               cout << " " << stripNum << " " << stripl << endl;
              stripl=stripNum;
            }
            if (clust_in_progress!=0) {
              count++;
              cout << " nstrips " << nstr << "  " << deltastr << endl;
              nstrip[count]=nstr;
              layer[count]=ilay;
              ladder[count]=ilad;
              barrel[count]=ibar;
              delstrip[count]=-1.;
              side[count]=view;
              pulse[count]=totpulse;
              vector<int> mlist;
              for (int iik=0; iik<StrList.size(); iik++){
                mctracks = minter.get_mcids(StrList[iik]);
                cout << " Mc tracks size " << mctracks.size()<< endl;
                mlist.insert(mlist.begin(),mctracks.begin(),mctracks.end());
              }
              unique(mlist);
              nmchits[count]=nmchits[count]+mlist.size();
              cout << " total Mc tracks " << nmchits[count] << endl;
              StrList.resize(0);
              cout << " Strip list size " << StrList.size() << endl;
              nstr=0;
              totpulse=0;
              clust_in_progress=0;
            }
          }
        }
        }
        }
  /*
        idet=2;
        int idisk, iwedge;
        for (idisk=1; idisk<=12; idisk++){
        for (iwedge=1; iwedge<=12; iwedge++){
        smtStrips =smtdata->getModuleStrips(idet,idisk,iwedge);
        //      cout <<"Fdisk/wedge"<<idisk<<","<<iwedge<<endl;
        for (jstr=smtStrips.begin(); jstr!=(smtStrips).end(); jstr++){
          cout  << (*jstr) << endl;
        }
        }
        }
        idet=3;
        for (idisk=1; idisk<=4; idisk++){
        for (iwedge=1; iwedge<=24; iwedge++){
        smtStrips =smtdata->getModuleStrips(idet,idisk,iwedge);
        //      cout <<"Hdisk/wedge"<<idisk<<","<<iwedge<<endl;
        for (jstr=smtStrips.begin(); jstr!=(smtStrips).end(); jstr++){
          cout  << (*jstr) << endl;
        }
        }
        }
  */
```



```
        if (count!=0) {
          cout << " BDATA  ntuple " << count << endl;
          int ntpl = 9;
        _ptuple[ntpl]->capture("BDATA::nclust",count);
     //  array size
        _ptuple[ntpl]->capture("BDATA::layer",layer);  //  first arra
        _ptuple[ntpl]->capture("BDATA::ladder",ladder);  //  first arra
        _ptuple[ntpl]->capture("BDATA::barrel",barrel);  //  first arra
        _ptuple[ntpl]->capture("BDATA::side",side);  //  first arra
        _ptuple[ntpl]->capture("BDATA::nstrip",nstrip);  //  first arra
        _ptuple[ntpl]->capture("BDATA::delstrip",delstrip);  //  first arra
        _ptuple[ntpl]->capture("BDATA::pulse",pulse);  //  first arra
        _ptuple[ntpl]->capture("BDATA::nmchits",nmchits);  //  first arra

        _ptuple[ntpl]->captureBlock("BDATA");
        }
    }
  else
    {
     cout<<"SmtDataChunk pointer not valid for this event\n";
    }



  //georg adds:

  Fill_SMTBarrel(event,8); // changed by doug
  Fill_MCBHit(event,8);      // changed by doug

  //  Fill_SMTDisk(event,9); // changed by doug
  //  Fill_MCDHit(event,9);      // changed by doug

  Fill_SMTBClust(event,7);
  //  Fill_SMTDClust(event,11);

  // changed by doug
  // fix(?) to entry problem. Removed these two lines from Fill_SMTBarrel
  // and Fill_MCHit so that they get call once for this one ntuple.

   _ptuple[8]->storeCapturedData();   // fill ntuple
   _ptuple[8]->clearData();           // write to disk and clear

   _ptuple[9]->storeCapturedData();   // fill ntuple
   _ptuple[9]->clearData();           // write to disk and clear

   _ptuple[7]->storeCapturedData();   // fill ntuple
   _ptuple[7]->clearData();           // write to disk and clear


   //   _ptuple[11]->storeCapturedData();   // fill ntuple
   //   _ptuple[11]->clearData();           // write to disk and clear


  return Result::success;
}

void SMTAnalyze::fill_hists(MCvtx* firstv, list<MCvtx*> &listv){
  //  get x, y, z of primary vertex
  float xprim=firstv->x();
  float yprim=firstv->y();
  float zprim=firstv->z();
  list<MCvtx*>::iterator jv;
  //  loop over all displaced vertices
  for( jv=listv.begin(); jv != listv.end(); ++jv){
    //  get x, y, z of 2ndary vertex
    float x2=(*jv)->x();
    float y2=(*jv)->y();
    float z2=(*jv)->z();
    //  calculate distance from primary vertex
    float vdist=sqrt((x2-xprim)*(x2-xprim)+(y2-yprim)*(y2-yprim)+
                     (z2-zprim)*(z2-zprim));
    //  accumulate histograms
    _vdist->accumulate(vdist);
```

```
    _vxy->accumulate(x2,y2);
  }
}

//******************************
// fill_ntuple
//**************************

void SMTAnalyze::fill_ntuple(list<MCparticle*> &listp, int ntpl)
{
  list<MCjet*> listj;  // this list will contain jets found by jet algorithm
  _jalgo->makeJets(listp,listj);  // find jets using listp

  // arrays for ntuples

  float px[_maxjets],py[_maxjets],pz[_maxjets],E[_maxjets];
  float pt[_maxjets],phi[_maxjets],eta[_maxjets];
  float emE[_maxjets],width[_maxjets],emwidth[_maxjets];
  //  zero the arrays
  for(int i=0; i<_maxjets; ++i){
    px[i]=0;
    py[i]=0;
    pz[i]=0;
    pt[i]=0;
    E[i]=0;
    phi[i]=0;
    eta[i]=0;
    emE[i]=0;
    width[i]=0;
    emwidth[i]=0;
  }
  // iterate over jets to fill arrays

  list<MCjet*>::iterator jj;
  L4vec sum;   // vector for sum over jets
  int nj=0;    // jet counter

  for ( jj=listj.begin(); jj != listj.end() && nj<_maxjets; ++jj ) {
    L4vec vec4=(*jj)->vec4();
    px[nj]=vec4.px();
    py[nj]=vec4.py();
    pz[nj]=vec4.pz();
    E[nj]=vec4.E();
    pt[nj]=(*jj)->pT();
    width[nj]=(*jj)->width();
    emE[nj]=(*jj)->emE();
    emwidth[nj]=(*jj)->emwidth();
    sum+=vec4;
    nj++;
  }
  _njets=nj;
  float mass=sum.mass();  // calculate mass from jets

  // fill ntuples

  _ptuple[ntpl]->capture("JETS::njets",_njets);     //  array size
  _ptuple[ntpl]->capture("JETS::px",&(px[0]));  //  first array
  _ptuple[ntpl]->capture("JETS::py",&(py[0]));
  _ptuple[ntpl]->capture("JETS::pz",&(pz[0]));
  _ptuple[ntpl]->capture("JETS::E",&(E[0]));
  _ptuple[ntpl]->capture("JETS::pt",&(pt[0]));
  _ptuple[ntpl]->capture("JETS::phi",&(phi[0]));
  _ptuple[ntpl]->capture("JETS::eta",&(eta[0]));
  _ptuple[ntpl]->capture("JETS::emE",&(emE[0]));
  _ptuple[ntpl]->capture("JETS::width",&(width[0]));
  _ptuple[ntpl]->capture("JETS::emwidth",&(emwidth[0])); // last array
  _ptuple[ntpl]->column("JETS::mass",mass,(float)999.9); // single value
  _ptuple[ntpl]->captureBlock("JETS");
  _ptuple[ntpl]->storeCapturedData();   // fill ntuple
  _ptuple[ntpl]->clearData();           // write to disk and clear

}
```



```
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
///////////JOB SUMMARY/////////////////////////////////////////////
Result SMTAnalyze::jobSummary(){
  out()<<endl<<" Processed "<<_count<<" events"<<endl;
  return Result::success;
}



// georgs stuff
/////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////
/////////////////////BOOK SMT BARREL HITS//////////////////////////
void SMTAnalyze::Book_SMTBHit(int unit)

{


  cout <<"in  SMTAnalyze::Book_SMTBHit()"<<endl;

      int _maxdets=10;

      _ptuple[unit]->columnAt("HITS::nhits",&_nhit,(int)0);
      _ptuple[unit]->span("HITS::nhits",0,_maxhits);
      // _ptuple[unit]->columnArray("HITS::nhits",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::detnum",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::hitnum",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::xgsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::ygsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::zgsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::xlsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::zlsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::detid",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::layer",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::hitphi",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::barrel",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::nstrip1",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::nstrip2",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::dedx1",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::dedx2",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::theta",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::trphi",_maxhits,(float)999.9);

      // Changed by Doug. Had the incorrect use of index("nhits")
      // Not sure what dimension does, but left it in.

      _ptuple[unit]->dimension(_maxhits);
      _ptuple[unit]->index("detnum","nhits");
      _ptuple[unit]->index("hitnum","nhits");
      _ptuple[unit]->index("xgsmth","nhits");
      _ptuple[unit]->index("ygsmth","nhits");
      _ptuple[unit]->index("zgsmth","nhits");
      _ptuple[unit]->index("xlsmth","nhits");
      _ptuple[unit]->index("zlsmth","nhits");
      _ptuple[unit]->index("detid","nhits");
      _ptuple[unit]->index("hitphi","nhits");
      _ptuple[unit]->index("layer","nhits");
      _ptuple[unit]->index("barrel","nhits");
      _ptuple[unit]->index("nstrip1","nhits");
      _ptuple[unit]->index("nstrip2","nhits");
      _ptuple[unit]->index("dedx1","nhits");
      _ptuple[unit]->index("dedx2","nhits");
      _ptuple[unit]->index("theta","nhits"); //added Georg 0517
      _ptuple[unit]->index("trphi","nhits"); //added Georg 0517


      _ptuple[unit]->clearDataBlock("HITS");

}
```

```
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////////////BOKK SMT BARREL CLUSTERS///////////////////////////
void SMTAnalyze::Book_SMTBClust(int unit)

{

    cout <<"in  SMTAnalyze::Book_SMTBClust()"<<endl;


      _ptuple[unit]->columnAt("CLUST::nclust",&_nclust,(int)0);
      _ptuple[unit]->span("CLUST::nclust",0,_maxclust);

      _ptuple[unit]->columnArray("CLUST::layer",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::ladder",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::barrel",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::nstrip",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::type",_maxclust,(float)999.9);


      // Changed by Doug. Had the incorrect use of index("nhits")
      // Not sure what dimension does, but left it in.

      _ptuple[unit]->dimension(_maxclust);
      _ptuple[unit]->index("layer","nclust");
      _ptuple[unit]->index("ladder","nclust");
      _ptuple[unit]->index("barrel","nclust");
      _ptuple[unit]->index("nstrip","nclust");
      _ptuple[unit]->index("type","nclust");


      _ptuple[unit]->clearDataBlock("CLUST");


}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////////////BOKK SMT BARREL DATA///////////////////////////
void SMTAnalyze::Book_SMTBDATA(int unit)

{

    cout <<"in  SMTAnalyze::Book_SMTBClust()"<<endl;

    cout << " unit = " << unit << endl;
      _ptuple[unit]->columnAt("BDATA::nclust",&_nclust,(int)0);
      _ptuple[unit]->span("BDATA::nclust",0,_maxclust);

      _ptuple[unit]->columnArray("BDATA::layer",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::ladder",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::barrel",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::side",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::nstrip",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::delstrip",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::pulse",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("BDATA::nmchits",_maxclust,(float)999.9);

      _ptuple[unit]->dimension(_maxclust);
      _ptuple[unit]->index("layer","nclust");
      _ptuple[unit]->index("ladder","nclust");
      _ptuple[unit]->index("barrel","nclust");
      _ptuple[unit]->index("side","nclust");
      _ptuple[unit]->index("nstrip","nclust");
      _ptuple[unit]->index("delstrip","nclust");
      _ptuple[unit]->index("pulse","nclust");
      _ptuple[unit]->index("nmchits","nclust");


      _ptuple[unit]->clearDataBlock("BDATA");
```



```
}



////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
////////////////BOOK SMT DISK  CLUSTER////////////////////
void SMTAnalyze::Book_SMTDClust(int unit)

{


   cout <<"in  SMTAnalyze::Book_SMTDClust()"<<endl;


      _ptuple[unit]->columnAt("CLUST::nclust",&_nclust,(int)0);
      _ptuple[unit]->span("CLUST::nclust",0,_maxclust);

      _ptuple[unit]->columnArray("CLUST::disktyp",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::disk",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::wedge",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::nstrip",_maxclust,(float)999.9);
      _ptuple[unit]->columnArray("CLUST::type",_maxclust,(float)999.9);



      // Changed by Doug. Had the incorrect use of index("nhits")
      // Not sure what dimension does, but left it in.

      _ptuple[unit]->dimension(_maxclust);
      _ptuple[unit]->index("disktyp","nclust");
      _ptuple[unit]->index("disk","nclust");
      _ptuple[unit]->index("wedge","nclust");
      _ptuple[unit]->index("nstrip","nclust");
      _ptuple[unit]->index("type","nclust");



      _ptuple[unit]->clearDataBlock("CLUST");


}



////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
////////////////BOOK SMT DISK  HIT////////////////////

void SMTAnalyze::Book_SMTDHit(int unit)

{


    cout <<"in  SMTAnalyze::Book_SMTDHit()"<<endl;

      int _maxdets=10;


      _ptuple[unit]->columnAt("HITS::nhits",&_nhit,(int)0);
      _ptuple[unit]->span("HITS::nhits",0,_maxhits);
      // _ptuple[unit]->columnArray("HITS::nhits",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::detnum",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::hitnum",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::xgsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::ygsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::zgsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::xlsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::zlsmth",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::detid",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::dsktyp",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::wedge",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::disk",_maxhits,(float)999.9);
```

```
      _ptuple[unit]->columnArray("HITS::nstrip1",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::nstrip2",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::dedx1",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::dedx2",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::theta",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("HITS::trphi",_maxhits,(float)999.9);

      // Changed by Doug. Had the incorrect use of index("nhits")
      // Not sure what dimension does, but left it in.

      _ptuple[unit]->dimension(_maxhits);
      _ptuple[unit]->index("detnum","nhits");
      _ptuple[unit]->index("hitnum","nhits");
      _ptuple[unit]->index("xgsmth","nhits");
      _ptuple[unit]->index("ygsmth","nhits");
      _ptuple[unit]->index("zgsmth","nhits");
      _ptuple[unit]->index("xlsmth","nhits");
      _ptuple[unit]->index("zlsmth","nhits");
      _ptuple[unit]->index("detid","nhits");
      _ptuple[unit]->index("dsktyp","nhits");
      _ptuple[unit]->index("wedge","nhits");
      _ptuple[unit]->index("disk","nhits");
      _ptuple[unit]->index("nstrip1","nhits");
      _ptuple[unit]->index("nstrip2","nhits");
      _ptuple[unit]->index("dedx1","nhits");
      _ptuple[unit]->index("dedx2","nhits");
      _ptuple[unit]->index("theta","nhits");
      _ptuple[unit]->index("trphi","nhits");
      _ptuple[unit]->clearDataBlock("HITS");


}



////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/////////////////BOOK MC BARREL HIT/////////////////////////
void SMTAnalyze::Book_MCBHit(int unit)

{


   cout <<"in  SMTAnalyze::Book_MCBHit()"<<endl;

      int _maxdets=10;

      _ptuple[unit]->columnAt("MCHITS::nhitssim",&_nhit,(int)0);
      _ptuple[unit]->span("MCHITS::nhitssim",0,_maxhits);
      // _ptuple[unit]->columnArray("MCHITS::nhitssim",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::xgsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::ygsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::zgsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::xlsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::zlsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::gbarrel",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::glayer",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::gladder",_maxhits,(float)999.9);



      // Changed by Doug. Had the incorrect use of index("nhits")
      // Not sure what dimension does, but left it in.

      _ptuple[unit]->dimension(_maxhits);
      _ptuple[unit]->index("xgsimh","nhitssim");
      _ptuple[unit]->index("ygsimh","nhitssim");
      _ptuple[unit]->index("zgsimh","nhitssim");
      _ptuple[unit]->index("xlsimh","nhitssim");
      _ptuple[unit]->index("zlsimh","nhitssim");
      _ptuple[unit]->index("gbarrel","nhitssim");
      _ptuple[unit]->index("glayer","nhitssim");
      _ptuple[unit]->index("gladder","nhitssim");
```



```

      _ptuple[unit]->clearDataBlock("MCHITS");

}


////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/////////////////BOOK MC DISC HIT/////////////////////

void SMTAnalyze::Book_MCDHit(int unit)

{


  //  cout <<"in  SMTAnalyze::Book_MCHit()"<<endl;

      int _maxdets=10;

      _ptuple[unit]->columnAt("MCHITS::nhitssim",&_nhit,(int)0);
      _ptuple[unit]->span("MCHITS::nhitssim",0,_maxhits);
      // _ptuple[unit]->columnArray("MCHITS::nhitssim",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::xgsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::ygsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::zgsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::xlsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::zlsimh",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::gdisktyp",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::gdisk",_maxhits,(float)999.9);
      _ptuple[unit]->columnArray("MCHITS::gwedge",_maxhits,(float)999.9);




      // Changed by Doug. Had the incorrect use of index("nhits")
      // Not sure what dimension does, but left it in.

      _ptuple[unit]->dimension(_maxhits);
      _ptuple[unit]->index("xgsimh","nhitssim");
      _ptuple[unit]->index("ygsimh","nhitssim");
      _ptuple[unit]->index("zgsimh","nhitssim");
      _ptuple[unit]->index("xlsimh","nhitssim");
      _ptuple[unit]->index("zlsimh","nhitssim");
      _ptuple[unit]->index("gdisktyp","nhitssim");
      _ptuple[unit]->index("gdisk","nhitssim");
      _ptuple[unit]->index("gwedge","nhitssim");

      _ptuple[unit]->clearDataBlock("MCHITS");

}



////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/////////////////FILL SMT HIT///////////////////////////////

void SMTAnalyze::Fill_SMTHit()
{

}

void SMTAnalyze::Fill_SMTBClust(Event &evt, int unit)
{

   float barrel[_maxclust];
   float layer[_maxclust];
   float ladder[_maxclust];
   float nstrip[_maxclust];
   float type[_maxclust];

   int nclust;
```

```
    for (int iii = 0; iii<_maxhits; iii++)
      {
        barrel[iii]=0.0;
        layer[iii]=0.0;
        ladder[iii]=0.0;
        nstrip[iii]=0.0;
        type[iii]=0.0;
      }

  edm::THandle<SMTBarClusDataChunk> bcdata =
    BClusDataKey.find(evt);

      int count =0;

            if (bcdata.isValid())
              {

               nclust = bcdata->data().nClus();

               for (int i = 1; i<=nclust; i++)
                  {
                   barrel[count] =
                     bcdata->data().unpack_barrel_number(
                                                    bcdata->
                                                    data().cluster(i).
                                                    faddress());
                   layer[count] =
                     bcdata->data().unpack_layer_number(
                                                   bcdata->
                                                   data().cluster(i).
                                                   faddress());
                   ladder[count] =
                     bcdata->data().unpack_phi_number(
                                                 bcdata->
                                                 data().cluster(i).
                                                 faddress());
                   type[count] =
                     bcdata->data().unpack_view_number(
                                                  bcdata->
                                                  data().cluster(i).
                                                  faddress());

                   nstrip[count] = bcdata->data().cluster(i).nStrip();


                   ++count;

                 if (count>_maxclust) break;

                  }



              }
            else
              {
                cout<< "Handle to SMTBarClusDataChunk not valid\n";
              }


        int ntpl = unit;

        _ptuple[ntpl]->capture("CLUST::nclust",count);     //  array size
        _ptuple[ntpl]->capture("CLUST::layer",layer);  //  first arra
        _ptuple[ntpl]->capture("CLUST::ladder",ladder);  //  first arra
        _ptuple[ntpl]->capture("CLUST::barrel",barrel);  //  first arra
        _ptuple[ntpl]->capture("CLUST::nstrip",nstrip);  //  first arra
        _ptuple[ntpl]->capture("CLUST::type",type);  //  first arra

        _ptuple[ntpl]->captureBlock("CLUST");


}
```



```
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/////////////////FILL SMT DISK  CLUSTER/////////////////////

void SMTAnalyze::Fill_SMTDClust(Event &evt, int unit)
{
  // fill in later.
}


////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
/////////////////FILL SMT BARREL////////////////////////////

void SMTAnalyze::Fill_SMTBarrel(Event &evt, int unit)
{


  //  cout << "in  SMTAnalyze::Fill_SMTBarrel"<<endl;


  edm::THandle<SMTBarClusDataChunk> bcdata =
    BClusDataKey.find(evt);


  //  if (bcdata.isValid())
  //    {
      //  cout << "For " << evt.collisionID();
      //cout << " We have the following barrel clusters\n\n";
      // bcdata->data().print_data('s');
    //    const SMTBarClusData& bclusDat =  bcdata->data();
  //    }
  //  else
  //    {
  //      cout<< "Handle to SMTBarClusDataChunk not valid\n";
  //    }



    edm::THandle<SMTPosBCollectChunk> bpdata =
      BPosKey.find(evt);

    if (bpdata.isValid())
      {
        //      cout << "For " << evt.collisionID();
        //cout << " We have the following barrel 1D positions\n\n";
      }
    else
      {
        cout<< "Handle to SMTPosBCollectChunk not valid\n";
      }




    float detnumber[_maxhits];
    float hitnumber[_maxhits];
    float xgsmth[_maxhits];
    float ygsmth[_maxhits];
    float zgsmth[_maxhits];
    float xlsmth[_maxhits];
    float zlsmth[_maxhits];
    float detid[_maxhits];
    float layer[_maxhits];
    float phi[_maxhits];
    float barrel[_maxhits];
    float dedx1[_maxhits];
    float dedx2[_maxhits];
    float theta[_maxhits];
    float trphi[_maxhits];

    float nstrip1[_maxhits];
```

```
     float nstrip2[_maxhits];

     for (int iii = 0; iii<_maxhits; iii++)
       {
         detnumber[iii] = 0.0;
         hitnumber[iii] = 0.0;
         xgsmth[iii] = 0.0;
         ygsmth[iii] = 0.0;
         zgsmth[iii] = 0.0;
         xlsmth[iii] = 0.0;
         zlsmth[iii] = 0.0;
         detid[iii] = 0.0;
         layer[iii] = 0.0;
         phi[iii] = 0.0;
         barrel[iii] = 0.0;
         dedx1[iii] = 0.0;
         dedx2[iii] = 0.0;
         theta[iii] = 0.0;
         trphi[iii] = 0.0;
         nstrip1[iii]=0.0;
         nstrip2[iii]=0.0;
       }


     int index = 0;

     edm::THandle<SMTGlblBCollectChunk> bhdata =
       BHitKey.find(evt);

     if (bhdata.isValid())
       {
         //      cout << "For " <<  evt.collisionID();
         //      cout << " We have the following barrel hits\n\n";
         //bhdata->data().print_data('s');
         //cout<<"Number of dets in ntuple"<<bhdata->data().nDet()<<endl;//gst

         int ndets =  bhdata->data().nDet();



         for (int detnum = 1; detnum<ndets+1 ; detnum++){

           // SMTXyzGlbl myglobal =  bhdata->data().hitData(detnum);
           const SMTXyzGlbl& myglobal =  bhdata->data().hitData(detnum); //slava
           int nhits = myglobal.nHits(); // number of hits on detector


           for (int hitid = 1; hitid<nhits+1 ; hitid++){
             //  SMTHit  my_smthit=  myglobal.hits(hitid);
             const SMTHit& my_smthit=  myglobal.hits(hitid);//slava

             std::vector<float>  hitvector = my_smthit.siPos();
             std::vector<float> dedxvector = my_smthit.dedxV();
             std::vector<int> clusnumvector = my_smthit.clusNum();

             //  cout<<"for detnum="<<detnum<<"hitid="<<hitid<<"We have 3-d pos:"<<e
+ndl;


             detnumber[index]=(float)detnum;
             hitnumber[index]=(float)hitid;
             xgsmth[index]=hitvector[0];
             ygsmth[index]=hitvector[1];
             zgsmth[index]=hitvector[2];
             xlsmth[index]=hitvector[3];
             zlsmth[index]=hitvector[4];
             detid[index] = (float)myglobal.detID();
             dedx1[index]=dedxvector[0];
             dedx2[index]=dedxvector[1];

             if (bcdata.isValid())
               {
           nstrip1[index] = bcdata->data().cluster_for_ID(clusnumvector[0]).
```



```
             nStrip();
           nstrip2[index] = bcdata->data().cluster_for_ID(clusnumvector[1]).
             nStrip();
               }
             else
               {
                 cout<< "Handle to SMTBarClusDataChunk not valid\n";
               }


             layer[index]=(float)(bhdata->data().layer(detid[index]));
             phi[index]=(float)(bhdata->data().phi(detid[index]));
             barrel[index]=(float)(bhdata->data().barrel(detid[index]));




             //add smt_hitmatch stuff here


             MatchInterface minter;
             //  SMTStripRecoHitMatch smatch;

             int ind= bhdata->get_index(&my_smthit,barrel[index],layer[index],
                                        phi[index]);

             //get ids to Monte Carlo Hits
             McHitList mc2list = minter.get_mchits_real_diff(&my_smthit) ;

             int nmt =  mc2list.size();
             cout <<"We  found  "<<nmt<<"mchits for this Barrel hit\n";

             McHitList::const_iterator im;

             //loop over MC hits

             float  pxg;
             float  pyg;
             float  pzg;

             for(im=mc2list.begin();im!=mc2list.end();im++){

              (*im)->getGlobalMomentum(pxg,pyg,pzg);

              cout<<"Global track coord"<<pxg<<endl;
             }


             //Now find angle
             theta[index] = find_theta(pxg,pyg,pzg);
             trphi[index] = find_phi(pxg,pyg,pzg);
             //cout<<"theta"<<theta<<endl;


             // end if smt_hitmatch



             index++;

             if (index>=_maxhits) break;

             //cout<<"index="<<index<<endl;
             //  for (int i=0; i < hitvector.size(); i++){
             // cout<<"hitvector="<<hitvector[i]<<endl;
             //}

           }
             if (index>=_maxhits) break;
         }
```

```
        int ntpl = unit;


        _ptuple[ntpl]->capture("HITS::nhits",index);           //  array size
        _ptuple[ntpl]->capture("HITS::detnum",&(detnumber[0]));//first array
        _ptuple[ntpl]->capture("HITS::hitnum",&(hitnumber[0]));
        _ptuple[ntpl]->capture("HITS::xgsmth",&(xgsmth[0]));
        _ptuple[ntpl]->capture("HITS::ygsmth",&(ygsmth[0]));
        _ptuple[ntpl]->capture("HITS::zgsmth",&(zgsmth[0]));
        _ptuple[ntpl]->capture("HITS::xlsmth",&(xlsmth[0]));
        _ptuple[ntpl]->capture("HITS::zlsmth",&(zlsmth[0]));
        _ptuple[ntpl]->capture("HITS::detid",&(detid[0]));
        _ptuple[ntpl]->capture("HITS::layer",&(layer[0]));
        _ptuple[ntpl]->capture("HITS::hitphi",&(phi[0]));
        _ptuple[ntpl]->capture("HITS::barrel",&(barrel[0]));
        _ptuple[ntpl]->capture("HITS::nstrip1",&(nstrip1[0]));
        _ptuple[ntpl]->capture("HITS::nstrip2",&(nstrip2[0]));
        _ptuple[ntpl]->capture("HITS::dedx1",&(dedx1[0]));
        _ptuple[ntpl]->capture("HITS::dedx2",&(dedx2[0]));
        _ptuple[ntpl]->capture("HITS::theta",&(theta[0]));
        _ptuple[ntpl]->capture("HITS::trphi",&(trphi[0]));


        _ptuple[ntpl]->captureBlock("HITS");

      }
    else
      {
        cout<< "Handle to SMTGlblBCollectChunk not valid\n";
      }


}



/////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////
////////////////////FILL SMT DISK///////////////////////////////////

void SMTAnalyze::Fill_SMTDisk(Event &evt, int unit)
{

  //cout << "in  SMTAnalyze::Fill_SMTDisk"<<endl;

  edm::THandle<SMTDiskClusDataChunk> dcdata =
    DClusDataKey.find(evt);


  //  if (dcdata.isValid())
  //    {
  //      cout << "For " << evt.collisionID();
  //cout << " We have the following disk clusters\n\n";
  // dcdata->data().print_data('s');
  //    }
  //    const SMTDiskClusData& dclusDat =  dcdata->data();

  //  else
  //    {
  //      cout<< "Handle to SMTDiskClusDataChunk not valid\n";
  //    }



    edm::THandle<SMTPosDCollectChunk> dpdata =
      DPosKey.find(evt);

    if (dpdata.isValid())
      {
        //      cout << "For " << evt.collisionID();
        //      cout << " We have the following disk 1D positions\n\n";
      }
    else
```



```
      {
        cout<< "Handle to SMTPosDCollectChunk not valid\n";
      }




    float detnumber[_maxhits];
    float hitnumber[_maxhits];
    float xgsmth[_maxhits];
    float ygsmth[_maxhits];
    float zgsmth[_maxhits];
    float xlsmth[_maxhits];
    float zlsmth[_maxhits];
    float detid[_maxhits];
    float dsktyp[_maxhits];
    float wedge[_maxhits];
    float disk[_maxhits];
    float dedx1[_maxhits];
    float dedx2[_maxhits];
    float theta[_maxhits];
    float trphi[_maxhits];

    float nstrip1[_maxhits];
    float nstrip2[_maxhits];

    for (int iii = 0; iii<_maxhits; iii++)
      {
        detnumber[iii] = 0.0;
        hitnumber[iii] = 0.0;
        xgsmth[iii] = 0.0;
        ygsmth[iii] = 0.0;
        zgsmth[iii] = 0.0;
        xlsmth[iii] = 0.0;
        zlsmth[iii] = 0.0;
        detid[iii] = 0.0;
        dsktyp[iii] = 0.0;
        wedge[iii] = 0.0;
        disk[iii] = 0.0;
        dedx1[iii] = 0.0;
        dedx2[iii] = 0.0;
        theta[iii] = 0.0;
        trphi[iii] = 0.0;
        nstrip1[iii]=0.0;
        nstrip2[iii]=0.0;
      }


    int index = 0;

    edm::THandle<SMTGlblDCollectChunk> dhdata =
      DHitKey.find(evt);

    if (dhdata.isValid())
      {
        //      cout << "For " <<  evt.collisionID();
        //      cout << " We have the following disk hits\n\n";
        //dhdata->data().print_data('s');
        //cout<<"Number of dets in ntuple"<<dhdata->data().nDet()<<endl;//gst

        int ndets =  dhdata->data().nDet();



        for (int detnum = 1; detnum<ndets+1 ; detnum++){

          SMTXyzGlbl myglobal =  dhdata->data().hitData(detnum);
          int nhits = myglobal.nHits(); // number of hits on detector

          for (int hitid = 1; hitid<nhits+1 ; hitid++){
            SMTHit  my_smthit=  myglobal.hits(hitid);
            std::vector<float>  hitvector = my_smthit.siPos();
            std::vector<float> dedxvector = my_smthit.dedxV();
```

```
          std::vector<int> clusnumvector = my_smthit.clusNum();

          //  cout<<"for detnum="<<detnum<<"hitid="<<hitid<<"We have 3-d pos:"<<e
+ndl;

          detnumber[index]=(float)detnum;
          hitnumber[index]=(float)hitid;
          xgsmth[index]=hitvector[0];
          ygsmth[index]=hitvector[1];
          zgsmth[index]=hitvector[2];
          xlsmth[index]=hitvector[3];
          zlsmth[index]=hitvector[4];
          detid[index] = (float)myglobal.detID();

          dedx1[index]=dedxvector[0];
          dedx2[index]=dedxvector[1];

          if (dcdata.isValid())
            {
        nstrip1[index] = dcdata->data().cluster_for_ID(clusnumvector[0]).
          nStrip();
        nstrip2[index] = dcdata->data().cluster_for_ID(clusnumvector[1]).
          nStrip();
            }
          else
            {
              cout<< "Handle to SMTDiskClusDataChunk not valid\n";
            }



          dsktyp[index]=(float)(dhdata->data().disk_type(detid[index]));
          wedge[index]=(float)(dhdata->data().wedge(detid[index]));
          disk[index]=(float)(dhdata->data().disk(detid[index]));



          //add smt_hitmatch stuff here


          MatchInterface minter;
          //  SMTStripRecoHitMatch smatch;



          //   int ind= dcdata->get_index(&my_smthit,disk[index],
          //                          wedge[index]);



          //get ids to Monte Carlo Hits
          //the following causes assertion
          //    McHitList mc2list = minter.get_mchits_real_diff(&my_smthit) ;
          McHitList mc2list ;

          int nmt =  mc2list.size();
          cout <<"We  found  "<<nmt<<"mchits for this Disc hit\n";

          McHitList::const_iterator im;

          //loop over MC hits

          float  pxg;
          float  pyg;
          float  pzg;

          for(im=mc2list.begin();im!=mc2list.end();im++){

           (*im)->getGlobalMomentum(pxg,pyg,pzg);

           cout<<"Global track coord"<<pxg<<endl;
          }
```



```
            //Now find angle
            theta[index] = find_theta(pxg,pyg,pzg);
            trphi[index] = find_phi(pxg,pyg,pzg);
            //cout<<"theta"<<theta<<endl;


            // end if smt_hitmatch

            index++;

            if (index>=_maxhits) break;


            //cout<<"index="<<index<<endl;


            //  for (int i=0; i < hitvector.size(); i++){
            // cout<<"hitvector="<<hitvector[i]<<endl;
            //}

          }

            if (index>=_maxhits) break;
        }



        int ntpl = unit;


        _ptuple[ntpl]->capture("HITS::nhits",index);     //  array size
        _ptuple[ntpl]->capture("HITS::detnum",&(detnumber[0]));  //first array
        _ptuple[ntpl]->capture("HITS::hitnum",&(hitnumber[0]));
        _ptuple[ntpl]->capture("HITS::xgsmth",&(xgsmth[0]));
        _ptuple[ntpl]->capture("HITS::ygsmth",&(ygsmth[0]));
        _ptuple[ntpl]->capture("HITS::zgsmth",&(zgsmth[0]));
        _ptuple[ntpl]->capture("HITS::xlsmth",&(xlsmth[0]));
        _ptuple[ntpl]->capture("HITS::zlsmth",&(zlsmth[0]));
        _ptuple[ntpl]->capture("HITS::detid",&(detid[0]));
        _ptuple[ntpl]->capture("HITS::dsktyp",&(dsktyp[0]));
        _ptuple[ntpl]->capture("HITS::wedge",&(wedge[0]));
        _ptuple[ntpl]->capture("HITS::disk",&(disk[0]));
        _ptuple[ntpl]->capture("HITS::nstrip1",&(nstrip1[0]));
        _ptuple[ntpl]->capture("HITS::nstrip2",&(nstrip2[0]));
        _ptuple[ntpl]->capture("HITS::dedx1",&(dedx1[0]));
        _ptuple[ntpl]->capture("HITS::dedx2",&(dedx2[0]));
        _ptuple[ntpl]->capture("HITS::theta",&(theta[0]));
        _ptuple[ntpl]->capture("HITS::trphi",&(trphi[0]));

        _ptuple[ntpl]->captureBlock("HITS");



      }
    else
      {
        cout<< "Handle to SMTGlblDCollectChunk not valid\n";
      }


}

/////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////
////////////////////FILL MC BARREL HIT///////////////////////////////

void SMTAnalyze::Fill_MCBHit(Event &evt, int unit)
{

  // cout << "in  SMTAnalyze::Fill_Mchits"<<endl;
```

```
  THandle<SimSMTHitChunk> ptrsmtchunk=_smthitKey.find(evt);
  if (ptrsmtchunk.isValid()){



    float xgsimh[_maxhits];
    float ygsimh[_maxhits];
    float zgsimh[_maxhits];
    float xlsimh[_maxhits];
    float zlsimh[_maxhits];
    float gbarrel[_maxhits];
    float gladder[_maxhits];
    float glayer[_maxhits];

    for (int iii = 0; iii<_maxhits; iii++)
      {

     xgsimh[iii]= 0.0;
     ygsimh[iii]= 0.0;
     zgsimh[iii]= 0.0;
     xlsimh[iii]= 0.0;
     zlsimh[iii]= 0.0;
     gbarrel[iii] = 0.0;
     glayer[iii] = 0.0;
     gladder[iii] = 0.0;
     }

    int index = 0;



    ptrsmtchunk->printChunk(cout);

    list<SimTkHitCollection*> tkhclist;
    ptrsmtchunk->getCollectionList(tkhclist);

    list<SimTkHitCollection*>::iterator jhc;

    int nhit=0;

    for(jhc=tkhclist.begin(); jhc !=tkhclist.end(); ++jhc){

      list<SimTkHit*> tkhitlist;
      string HCname;
      (*jhc)->getHCName(HCname);


      //    cout<<"Tracker hit collection name "<<HCname<<endl;


      (*jhc)->getHitList(tkhitlist);
      list<SimTkHit*>::iterator jhit;

      for(jhit=tkhitlist.begin(); jhit !=tkhitlist.end(); ++jhit){
        ++nhit;

        vector<int>  volIndex;

        (*jhit)->getVolumeIndex(volIndex);

        int  myindex[20];

        for (int i=0; i < volIndex.size(); i++){

          //      cout<<"Volume Index in MCHIT="<<volIndex[i]<<endl;

        myindex[i]=volIndex[i];


        } // for loop
```



```
          if (myindex[0]==1)
            {
              gbarrel[index] = float(myindex[1]);
              glayer[index]  = float(myindex[2]);
              gladder[index] = float(myindex[3]);



              //      cout << " SMT ideal Geant hit # " << nhit << endl;

              int itrack;
              (*jhit)->getTrackNumber(itrack);
              //      cout<<" track number "<<itrack<<endl;
              float xg,yg,zg,pxg,pyg,pzg;
              xg=0.;
              yg=0.;
              zg=0.;
              pxg=0.;
              pyg=0.;
              pzg=0.;
              (*jhit)->getGlobalXYZ(xgsimh[index],ygsimh[index],zgsimh[index]);
              (*jhit)->getGlobalMomentum(pxg,pyg,pzg);


              float xin,yin,zin,xout,yout,zout;
              float steplength,g3code,edep,tof;
              (*jhit)->getLocalXYZin(xin,yin,zin);
              (*jhit)->getLocalXYZout(xout,yout,zout);

              xlsimh[index]=(xout+xin)/2.0; // correction by doug
              zlsimh[index]=(zout+zin)/2.0; // correction by doug

              (*jhit)->getStep(steplength);
              (*jhit)->getG3ParticleCode(g3code);
              (*jhit)->getEnergyDeposit(edep);
              (*jhit)->getTOF(tof);
              //cout<<" local (in) x,y,z "<<xin<<" "<<yin<<" "<<zin<<endl;
              //cout<<"       (out) x,y,z "<<xout<<" "<<yout<<" "<<zout<<endl;
              //      cout<<"  step "<<steplength<<" g3code "<<g3code<<" edep "<<edep
+<<" tof "<<tof<<endl;




              index++;



            } // if barrel data


              if (index>=_maxhits) break;

          }         // end of loop over hits

              if (index>=_maxhits) break;

        }           // end of loop over hit collections



        int ntpl= unit;
       _ptuple[ntpl]->capture("MCHITS::nhitssim",index);     //  array size
       _ptuple[ntpl]->capture("MCHITS::xgsimh",&(xgsimh[0]));
       _ptuple[ntpl]->capture("MCHITS::ygsimh",&(ygsimh[0]));
       _ptuple[ntpl]->capture("MCHITS::zgsimh",&(zgsimh[0]));
       _ptuple[ntpl]->capture("MCHITS::xlsimh",&(xlsimh[0]));
       _ptuple[ntpl]->capture("MCHITS::zlsimh",&(zlsimh[0]));
       _ptuple[ntpl]->capture("MCHITS::gbarrel",&(gbarrel[0]));
       _ptuple[ntpl]->capture("MCHITS::glayer",&(glayer[0]));
       _ptuple[ntpl]->capture("MCHITS::gladder",&(gladder[0]));
```

```
    _ptuple[ntpl]->captureBlock("MCHITS");


   }// if


   else
     {
       cout<< "Handle to SimSMTHitChunk not valid\n";
     }
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////FILL MC DISC HIT////////////////////////////////

void SMTAnalyze::Fill_MCDHit(Event &evt, int unit)
{


  //  cout << "in  SMTAnalyze::Fill_Mchits"<<endl;


  THandle<SimSMTHitChunk> ptrsmtchunk=_smthitKey.find(evt);
  if (ptrsmtchunk.isValid()){



    float xgsimh[_maxhits];
    float ygsimh[_maxhits];
    float zgsimh[_maxhits];
    float xlsimh[_maxhits];
    float zlsimh[_maxhits];
    float gdisktyp[_maxhits];
    float gdisk[_maxhits];
    float gwedge[_maxhits];

    for (int iii = 0; iii<_maxhits; iii++)
      {

        xgsimh[iii]= 0.0;
        ygsimh[iii]= 0.0;
        zgsimh[iii]= 0.0;
        xlsimh[iii]= 0.0;
        zlsimh[iii]= 0.0;
        gdisktyp[iii] = 0.0;
        gdisk[iii] = 0.0;
        gwedge[iii] = 0.0;
      }

    int index = 0;



    ptrsmtchunk->printChunk(cout);

    list<SimTkHitCollection*> tkhclist;
    ptrsmtchunk->getCollectionList(tkhclist);

    list<SimTkHitCollection*>::iterator jhc;
    int nhit=0;


    for(jhc=tkhclist.begin(); jhc !=tkhclist.end(); ++jhc){

      list<SimTkHit*> tkhitlist;
      string HCname;

      (*jhc)->getHCName(HCname);

      //         cout<<"Tracker hit collection name "<<HCname<<endl;

      (*jhc)->getHitList(tkhitlist);
```



```
        list<SimTkHit*>::iterator jhit;

        for(jhit=tkhitlist.begin(); jhit !=tkhitlist.end(); ++jhit){
          ++nhit;


          vector<int>  volIndex;

          (*jhit)->getVolumeIndex(volIndex);



          int  myindex[20];

          for (int i=0; i < volIndex.size(); i++){

            //      cout<<"Volume Index in MCHIT="<<volIndex[i]<<endl;

            myindex[i]=volIndex[i];


          }


          if (myindex[0]==2||myindex[0]==3)
            {

              if (myindex[0]==2)
                {
                 gdisktyp[index]=1.0;
                 gdisk[index] = myindex[1];
                 gwedge[index] = myindex[3];
                }
              else if (myindex[0]==3)
                {
                 gdisktyp[index]=2.0;
                 gdisk[index] = myindex[1];
                 gwedge[index] = myindex[3];
                }




              //      cout << " SMT ideal Geant hit # " << nhit << endl;

              int itrack;

              (*jhit)->getTrackNumber(itrack);

              //      cout<<" track number "<<itrack<<endl;

              float xg,yg,zg,pxg,pyg,pzg;
              xg=0.;
              yg=0.;
              zg=0.;
              pxg=0.;
              pyg=0.;
              pzg=0.;
              (*jhit)->getGlobalXYZ(xgsimh[index],ygsimh[index],zgsimh[index]);
              (*jhit)->getGlobalMomentum(pxg,pyg,pzg);

              float xin,yin,zin,xout,yout,zout;
              float steplength,g3code,edep,tof;
              (*jhit)->getLocalXYZin(xin,yin,zin);
              (*jhit)->getLocalXYZout(xout,yout,zout);
              xlsimh[index]=(xout+xin)/2.0; // correction by doug
              zlsimh[index]=(zout+zin)/2.0; // correction by doug

              (*jhit)->getStep(steplength);
              (*jhit)->getG3ParticleCode(g3code);
              (*jhit)->getEnergyDeposit(edep);
              (*jhit)->getTOF(tof);
              //cout<<" local (in) x,y,z "<<xin<<" "<<yin<<" "<<zin<<endl;
```

```
            //cout<<"      (out) x,y,z "<<xout<<" "<<yout<<" "<<zout<<endl;
            //      cout<<"  step "<<steplength<<" g3code "<<g3code<<" edep "<<edep
+<<" tof "<<tof<<endl;



            index++;

          } // if disk data

            if (index>=_maxhits) break;

      }         // end of loop over hits

            if (index>=_maxhits) break;

    }           // end of loop over hit collections



    int ntpl= unit;
   _ptuple[ntpl]->capture("MCHITS::nhitssim",index);     //  array size
   _ptuple[ntpl]->capture("MCHITS::xgsimh",&(xgsimh[0]));
   _ptuple[ntpl]->capture("MCHITS::ygsimh",&(ygsimh[0]));
   _ptuple[ntpl]->capture("MCHITS::zgsimh",&(zgsimh[0]));
   _ptuple[ntpl]->capture("MCHITS::xlsimh",&(xlsimh[0]));
   _ptuple[ntpl]->capture("MCHITS::zlsimh",&(zlsimh[0]));
   _ptuple[ntpl]->capture("MCHITS::gdisktyp",&(gdisktyp[0]));
   _ptuple[ntpl]->capture("MCHITS::gdisk",&(gdisk[0]));
   _ptuple[ntpl]->capture("MCHITS::gwedge",&(gwedge[0]));


   _ptuple[ntpl]->captureBlock("MCHITS");


  }// if


  else
    {
      cout<< "Handle to SimSMTHitChunk not valid\n";
    }
}



////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////////////////////CALCULATE THETA///////////////////////////////

float SMTAnalyze::find_theta(float px,float py,float pz)
{
  float pt = pow(px,2) + pow(py,2);
  pt = sqrt(pt);
  float theta=0.;
  float pi=3.141592653358;

    if (pz != 0. ){
      theta=atan(pt/pz);
    }

    if (theta < 0. ){
      theta=pi+theta;
    }



  return theta;
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////////////////////CALCULATE PHI////////////////////////////////
```



```
float SMTAnalyze::find_phi(float px,float py,float pz)
{
  float phi=0.;
  float pi=3.141592653358;

  if (px != 0. ){
    phi=atan(py/px);
  }

  if (px < 0. ){
    phi=pi+phi;
  }

  if (px > 0. && py < 0. ){
    phi=2*pi+phi;
  }


  return phi;
}
```