EUROPEAN LABORATORY FOR PARTICLE PHYSICS (CERN)

ALEPH/2000-093
SOFTWR/2000-004
December 21, 2000

Comparison between an event selection program performed with *ALPHA* and *ALPHA++*

Christophe Delaere (*delaere@fynu.ucl.ac.be*)
Vincent Lemaitre (*Vincent.lemaitre@cern.ch*)
Valeria Perez Reale (*Maria.Valeria.Perez.Reale@cern.ch*)
Olivier van der Aa (*vanderaa@fynu.ucl.ac.be*)

Abstract

New concrete classes have been developed for the *ALPHA++* project of the *ALEPH* collaboration in order to be able to use electron, muon, tau and jet objects. To check the reliability of this new framework, the hadronic selection of the single W analysis was performed with *ALPHA++* and was found to be identical to the standard *ALPHA* selection result.

1 Introduction

The *ALEPH* Physics Analysis Package *ALPHA* is intended to simplify Fortran programs for physics analysis. *ALPHA* also provides easy access to physical variables (e.g., momentum, energy), such that the user can write physics analysis programs without detailed knowledge of the *ALEPH* data structure (tabular BOS banks). An extensive set of utility routines (e.g., kinematics, event shape, secondary vertex finding, b-tagging, etc.) are available as part of the *ALPHA* package.

Evolution of programming languages, processors and compilers push us to evolve to an object oriented analysis framework. This is the main idea behind *ALPHA++*. It is indeed of first necessity to prevent the *ALEPH* data set to disappear due to a lack of compiler to access it. In addition, it is an interesting framework to become familiar with the new analysis tools that will be used at the LHC.

As explained in the *ALPHA++* requirement list[1] The main goals of this project are:

1. Convert the *ALEPH* data (preferably LEP2 data) from the BOS bank style into persistent objects and write them to an Objectivity database
2. Rewrite a mini version of the *ALEPH* analysis package *ALPHA* in an object oriented computing language (C++), based on the Objectivity database
3. Compare standard and OO performance with regard to efficient access of the data
4. Test the software engineered by the *RD45* and the *ANAPHE/LHC++* project, which evaluate data storage and analysis option for the LHC experiments.
5. Provide some input/experience for the LEP archive project
6. Give an opportunity to learn OO analysis, programming and design

In this package, a driver and an analysis part can be identified. The first part contains the initialization, run and termination routines and handles the conversion from BOS data bank to the Objectivity database. The second part has basically the same structure as *ALPHA*. It consists of two main objects: the tracks and the vertices. All the concrete objects (e.g. photons, tracks, jets, etc.) are classes deriving from the two aforementioned classes. Another class called AlphaBank, acts as a container for all the objects so that the user is provided with pointers to the AlphaBank object, which then contains appropriate methods to access all the other objects. The algorithms are, when possible, methods of the AlephVector Class. AlephVector inherit from the standard STL vector, so that it can be sorted, has a non-fixed size and knows its own size.

Our contribution on the project is mainly the development of a set of track-related classes, and of a set of selection algorithms needed to perform an analysis. We have implemented:

1. The muon class and the muon-filtering algorithm.
2. The electron class and the electron-filtering algorithm.
3. The tau class and the tau production/filtering algorithm.

4. The standard jet finding algorithms (Jade and Durham metrics)

To check the reliability of the new package, we have performed a detailed comparison of the *ALPHA* and *ALPHA++* versions of the hadronic selection of single W analysis [2]. The single W analysis routine has been developed in translating the existing selection code from FORTRAN to the *ALPHA++* framework (C++)[3]. This selection requires the identification of electron and tau candidates and make use of jet algorithms. In addition to the particle identification feature the program also fills a PAW ntuples file of some discriminating variables, such as lepton energy, missing momentum or jet and tau related quantities.

The structure of the analysis routine is quite representative of the general structure of an *ALPHA++* user routine. First, the user gets vectors of pointers to the basic objects he wants to study. He then applies filtering algorithms, sometimes provided by *ALPHA++*, such as for the lepton filter, sometimes made by himself because more specifics. In a third step he can use the remaining objects to produce jets, calculate topological variables, and so on. Finally, an ntuple file is produced.

The direct comparison, on an event by event basis, of the ntuples produced by the two versions shows that the result may be considered as exactly equivalent. The only difference is assimilated with the machine floating number representation (a relative difference of the order of 10^{-7}).

2 General improvements

2.1 The assignment operators

No more redefinition of the assignment operator is made and the default operator is chosen instead. The aim of this operator is to copy an object member by member. A redefinition is sometime useful for classes containing strings. Indeed, in this case, the default operator copies the pointer to the string and a change to one of the object will automatically affect the copy.

2.2 The QvecLink Class

QvecLink is an intermediate class between QvecBase and some concrete classes (Altrack, AlEflw, AlGamp). It contains all the methods which requires a pointer to QVEC. Thus, the classes which require the Qvec data member derive from QvecLink, the others derive directly from QvecBase.

2.3 The Lock functionality

The lock has been re-implemented. Now it is possible to :

1. Lock an object: Lock().

2. UnLock an object: unLock().

3. Lock or unLock an object and, recursively, all the objects contained : (un)Lock(1)

The lock feature is complementary to looperase() (Which acts on any alephVector). The point is that in some cases, the lock feature remains more effective. This is illustrated with two examples taken from the single W analysis.

- To perform a good quality charged track selection: Some track need to be removed from the analysis but not only in the pure AlTrack class. Doing just altp.looperase, where altp is the track pointers vector, will require to do the same on the vector of electron candidates, on the vector of muons candidates, etc. In addition locking the track still enable you to check the track quality later. It is used in the leptonic algorithms, where a locked track is seen as a bad track and is therefore excluded from being associated with a lepton candidate.

- To (temporary) ignore all the AlEflw objects of a jet: The single W selection requires to ignore all the tracks belonging to a tau. Using the looperase, one has to loop on all AlEflw, to compare them with each AlEflw of the tau and looperase the matching items. Even if it can be partially simplified using the STL findif() method, it is easier to do it with a lock : just do tau.lock(1) ; to lock recursively all the AlEflw objects in the tau-jet.

It is clear that such a lock can not do everything : it is a single flag. But the complementarity of the lock with the looperase gives more flexibility to the user to filter the various objects: to select objects using looperase when looping on a single AlephVector and using the lock facility to remove objects in a set of several vectors simultaneously. One can imagine, for instance, to use the lock in a first selection step and to looperase all the locked items in a second stage of the selection program.

2.4 In AlephVector...

The AlephVector class is a template class inheriting from the Standard Template Library (STL) vector class and contains, among other methods, the analysis algorithms. In addition to the jet algorithms and the lepton related algorithms, treated in another section, we have added some methods:

- The method Pointers() returns a vector of pointers to the objects of the original AlephVector.

- The method GetSum() returns the sum LorentzVector.

- The method GetMiss() returns the missing momentum.

You can now sort a vector of objects. Just use the STL sort(vector.begin(),vector.end()). The criterium is [0..3] for a component of the associated LorentzVector, 4 for Pt. It is chosen by QvecBase::SortCriterium. Default is 3(energy).

3 The jets algorithms.

The Durham and Jade jet algorithms have been implemented. The design of the jet related classes is similar to the one used in the ORCA framework of the CMS collaboration. Some improvements have been made in order to be able to force a fixed number of jets, and to provide the list of critical Y_{cut} values for which the event topology changes from n to $n+1$ jet (DurhanYn).

The algorithm is performed using two internal classes (DurhamProtojet and JadeProtojet) where the recombination scheme and the metric are defined. In order to produce jets, the following definitions need to be written:

```
AlephVector<Eflws*> eflws = bank.EflwsPV() ;
AlephVector <AlJet> J1 = eflws.DurhamJets(ycut, scheme, Evis) ;
AlephVector<AlJet> J2 = eflws.JadeJets(ycut, scheme, Evis) ;
```

Where bank is the name of a given alphabank. If $ycut = -n$, with n an integer, exactly n jets are generated. Scheme is the recombination scheme: 0 for $E0$ scheme or 1 for E scheme. Evis is the visible energy. If it is 0, the total energy of the objects in the vector is used.

DurhamYn(Scheme, Evis) has also been modified to use the new internal structure.

4 Leptonic selection.

Concrete classes for handling leptons have been implemented inheriting from the QvecBase class and therefore from AlObject as well : AlMuon, AlElec and AlTau. The Electron and Muon classes contain a pointer to the *ALPHA* banks (EIDT or MUID). There are method to access corresponding *ALPHA* BOS values and a pointer to the associated track. AlphaBank contains a vector of vector to those objects. For the tau, it is quite different because there is no bank containing tau candidates. The original vector is created by the user (see below for more details).

It is possible to separately select muons, electrons or taus. In all cases, the scheme is the same (see Fig. 1):

Get the candidates:

The first step is to get the primary objects (a vector of candidates). For the electrons (muons), it is the vector of all the objects with a related entry in the EIDT (MUID) bank. For the taus, it is a user made vector of taus. Indeed, if the muons/electrons candidates are generated form the relevant banks at the AlphaBank level, the taus are generated on demand by the user Using the AtauJn method.

Thus, the list of electron, muon and tau pointers are obtained as follows:

- `AlephVector<AlMuon*> muons = bank.muonPV() ;`
- `AlephVector<AlElec*> elecs = bank.elecPV() ;`
- ```
  AlephVector<AlEflw*> eflws = bank.eflwPV() ;
  AlephVector<AlTau> taus  = eflws.ATauJn(Ycut,scheme,metric,Evis) ;
  AlephVector<AlTau*> tausP = taus.Pointers() ;
  ```
  ```

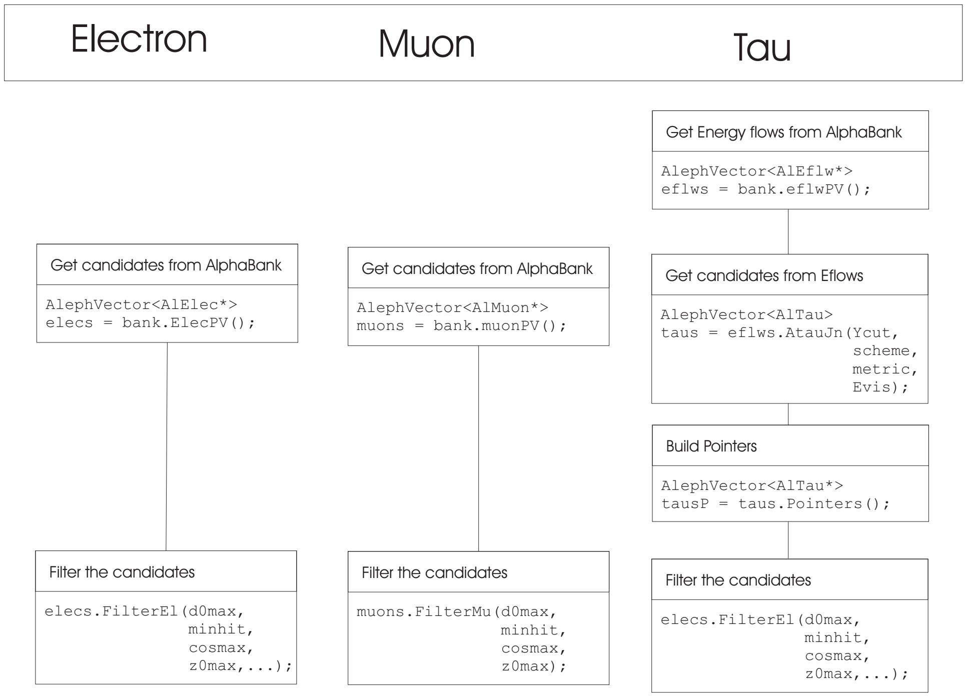

Figure 1: Selection procedure for electrons, muons and taus.

Note the additional step for the taus where the user has to provide the vector of pointers from the produced vector of objects provided by the AtauJn() method.

The standard parameters for ATauJn are :

Parameter	Values	Description
Ycut	$\simeq 0.001$	Cut used by the jet algo.
Scheme	0	Recombination scheme: 0 for E0 or 1 for E
Metric	1	1 for Jade, 0 for Durham.
Evis	QELEP	here set to the total center of mass energy If set to 0, the visible energy is used.

The standard way to use it is therefore: ATauJn(ycut,0,1,QELEP), with $ycut = 5.8^2/\text{QELEP}^2$.

Filter the candidates:

The second step is to filter the candidates with more precise cuts which are analysis dependent.

```
muons.FilterMu(d0max,pmin,minhit,cosmax,z0max) ;
elecs.FilterEl(almup,d0max,pmin,minhit,cosmax,z0max,nwirem,
               r3cut,r2cut0l, r2cut1l,r2cut2l,r2cuth,r5cutl,r5cuth) ;
tausP.FilterTau(pmin,d0max,minhit,cosmax,z0max) ;
```

standard parameters are the *ALPHA* ones :

Parameter	Values	Description
d0max	0.5 or 2 (for τ)	distance of closest approach to the beam.
Pmin	1	minimum momentum of the lepton
Minhit	4	minimum number of hits in the TPC
Cosmax	0.95	maximum cosine of the track-beam angle
Z0max	10	distance of closest approach to the beam in z.
Almup		vector of muons. (see below)
Nwirem	40	number of useful wires in the dE/dx calculation.
R3cut	1000	upper limit for the r3 parameter (output of dE/dx)
R2cut0l	-3	lower limit for r2 in a clean region of the calorimeter
R2cut1l	-7	lower limit in a crack
R2cut2l	-5	lower limit in overlap
R2cuth	1000	upper limit of r2.
R5cutl	-0.5	lower limit of r5. It is used out of clean regions
R5cuth	1000	upper limit of r5. It is used out of clean regions

A vector of muons has to be given (maybe empty) for the electron filtering. It is used to guaranty that none of the electrons has already been selected as a muon. If an empty vector is chosen, the user has to check this himself.

5 Collateral changes in the code

Some minor changes have been performed in other classes in order to access to the required informations from the new BOS banks used. These added lines of codes are generally a literal extension of the existing approach. Such changes have been done in alphacopy or alphabanks. A class diagram of the new version of alpha++ is shown in Fig. 2. Only the object part is shown. It has been reverse ingeneered with Rational Rose ; the new classes are shadowed.

6 Hadronic selection of the Single *W* analysis

The Hadronic selection of the Single W analysis has been performed in translating an existing selection code[3] from *ALPHA* to *ALPHA++*. In this note , we just focus on the main technical and/or physical parts of the user routine. The full code can be found in the appendices.

```
void AlephExManager::UserEvent(AlphaBanks& bb)
{
```

The first important thing to do is to get the vectors of objects from the AlphaBank, which is given as argument of the userevent routine (by reference).

```
// get vectors of pointers to the objects used in the analysis
  AlephVector<AlTrack *> altp=bb.TrackPV();
```

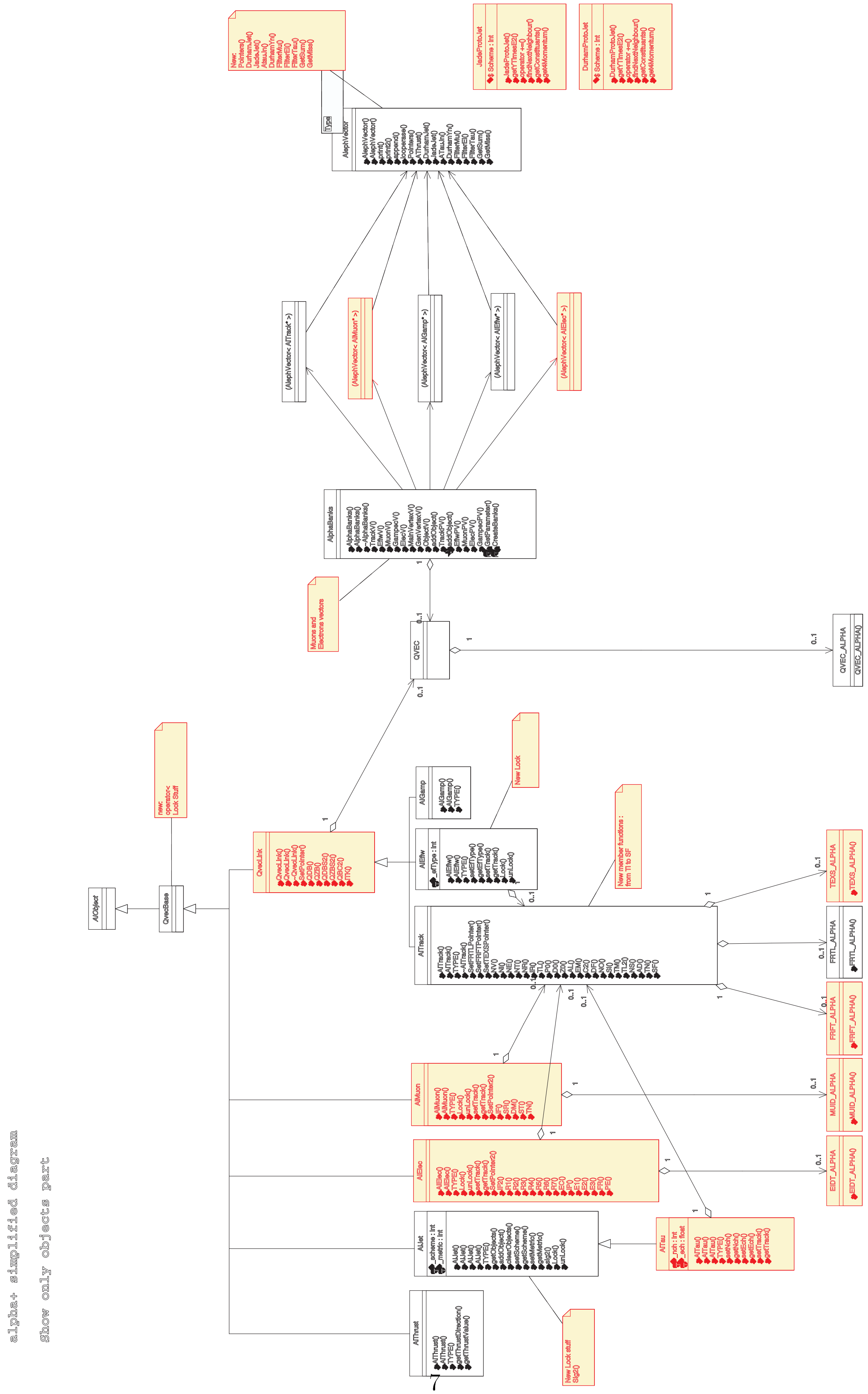

Figure 2: UML Diagram of the new *ALPHA++* version.

```
  AlephVector<AlMuon *> almup=bb.MuonPV();
  AlephVector<AlElec *> alelp=bb.ElecPV();
  AlephVector<AlEflw *> alefp=bb.EflwPV();
```

In this analysis, a good track is defined as having at least four TPC hits and a value of $|\cos\theta| < 0.95$. Moreover, it must originate within a cylinder of $10cm$ length and $2cm$ radius centered at the nominal interaction point and parallel to the beam (particles created in the hadronic decay should come from the main interaction point). Only charged tracks are selected where the angle theta to the beam direction is not too small because measurements in the detector are less precise at small angles. All charged tracks that don't fulfill these conditions are rejected applying the looperase() AlephVector method. As explained before, we also lock such tracks to avoid using them for leptons id.

```
// Good quality charged Tracks selection
  int Nchsel=0;
  float Echsel=0.;
  for (Ich=altp.begin(); Ich<altp.end(); Ich++)
    {
      int Ntpc = (*Ich)->NT();
      float costhch = (*Ich)->QCT();
      float d0 = (*Ich) -> QDB();
      float z0 = (*Ich) -> QZB();

      if ((Ntpc<4)||(costhch>0.95)||(d0>2.)||(z0>10.)
  ||(costhch<-.95)||(d0<-2)||(z0<-10))
{
  (*Ich)->Lock();         // Lock the tracks (for leptons algos)
  altp.looperase(Ich);    // Erases elements in the altp vector
}
      else
{
  Nchsel++;
  Echsel += (*Ich)->QE();
}
    }
```

Next, a selection of events according to the good charged tracks is performed. These cuts are applied to reject backgrounds from gamma-gamma interactions. The requirement is that the total number of charged tracks that have passed the track selection must be larger than five and that their total energy must be at least 5 GeV.

```
// now select the event according to the good charged tracks
  bool GoodEvent = false;
  if ((Nchsel >= 5) && (Echsel >= 5.))
    {
      // now we have a good event
```

```
      Ngood++;
      GoodEvent = true;
    }
  else
    cout << "bad event" << endl;
```

The electrons, muons and taus are then selected. As explained in section 4, the muons and the electrons are already obtained from the AlphaBank at the very beginning of the user analysis. Therefore, they just need to be filtered. The taus need to be both generated and filtered.

```
  //filter muons. parameters are (d0max,pmin,minhit,cosmax,z0max)
  almup.FilterMu(.5,9.43,4,.95,10);
  //filter elecs.parameters are
  //            (almup,d0max,pmin,minhit,cosmax,z0max,
  //             nwirem,r3cut,r2cut0l,r2cut1l,r2cut2l,
  //             r2cuth,r5cutl,r5cuth)
  alelp.FilterEl(almup,.5,(0.05*QELEP),4,.95,10,
 0,1000,-3,-7,-5,
 1000,-0.5,1000);
  //generate raw tau candidates. Parameters are (Ycut,Scheme,Metric,Qelep)
  AlephVector<AlTau> JJ = alefp.ATauJn(Yjade, 1,1,QELEP);
  AlephVector<AlTau*> altaup = JJ.Pointers();
  //filter taus. Parameters are (psummin,d0max,minhit,cosmax,z0max)
  altaup.FilterTau(psummin,2.,4,.95,10.);
```

Using the GetSum() and GetMiss() methods, the total invariant mass and the missing momentum are calculated. These methods return a Lorentz Vector, such that the final result is obtained using the standard *ALPHA*-like methods implemented in QvecBase.

```
  QvecBase Psum = alefp.GetSum();
  float esum = Psum.QE();
  float psum = Psum.QP();
  float msum = Psum.QM();
  QvecBase Pmiss = alefp.GetMiss(QELEP);
  float cthmiss=Pmiss.QCT();
```

Tagged two photons events are rejected by requiring that no energy of neutral object be detected within a cone of 12 degrees around the beam (e12). Thus, all ECAL hadrons/residuals, HCAL, LCAL and SICAL elements will contribute to e12. The energy in a 30 degrees wedge around the transverse missing momentum (ewedge) and the total energy in a cone of 12 degrees (econe) around the beam are also calculated.

```
  // wedge energy in 30 degree
  float ewedge=0;
  if (Pmiss.QP() >0)
```

```
    {
      for(Ifl = alefp.begin(); Ifl < alefp.end(); Ifl++)
if((acos(min(max(cos(Pmiss.QPH()-((*Ifl)->QPH())),-1.),1.)))*QQRADP<30)
  ewedge+= (*Ifl)->QE();
    }
  else
    GoodEvent=false;
  //energy in 12 deg cone centred aound beam pipe
  float e12_av=0;
  float econe =0;
  for(Ifl = alefp.begin(); Ifl < alefp.end(); Ifl++)
    {
      int type = (*Ifl)->getEfType();
      float costheta = fabs((*Ifl)->QCT());
      const double c12=0.9781476;
      float nint=(*Ifl)->QCH();
      if((type == 7) || (type == 8) || ((nint ==0) && (costheta>0.9781476)))
e12_av += (*Ifl)->QE();
      if(costheta > c12)
econe += (*Ifl)-> QE();
    }
```

The next step consist to identify the electron and/or muon with an energy greater than 0.05 x QELEP, the most anti-parallel to the missing momentum. The loop is explicitly performed over the remaining AlElec and AlMuon objects, respectively.

```
// Select the most anti-parallel (to Pmiss)
// identified electron or  muon
  int imutag=0;
  int ieltag=0;
  float emumax=0;
  float eelmax=0;
  float cosjmiss;
  AlMuon selmuon;
  AlElec selelec;
  AlTrack* theElTrack;
  AlTrack* theMuTrack;
  float cosjmin=1;
  for (Imu=almup.begin(); Imu<almup.end(); Imu++)
    if(Pmiss.QP() > 0)
      {
cosjmiss=(*Imu)->QCOSA(Pmiss);
if(cosjmiss <= cosjmin)
  {
    cosjmin=cosjmiss;
    emumax= (*Imu)->QE();
```

```
    selmuon =(**Imu);
    theMuTrack = (**Imu).getTrack();
    imutag =1;
  }
      }
  cosjmin=1;
  for (Iel=alelp.begin(); Iel<alelp.end(); Iel++)
    if(Pmiss.QP() > 0)
      {
cosjmiss=(*Iel)->QCOSA(Pmiss);
if(cosjmiss <= cosjmin)
  {
    cosjmin=cosjmiss;
    eelmax= (*Iel)->QE();
    selelec =(**Iel);
    theElTrack = (**Iel).getTrack();
    ieltag =1;
  }
      }
```

The selection of the most antiparallel tau is performed requiring that none of the tau tracks is already identified as an electron or a muon. The identification is done using the associated track information. Given that the track pointers are pointing to the BOS area in memory, the pointers can be directly compared.

```
// Select the most anti-parallel (to Pmiss)
// identified tau with energy > 0.05*ELEP
// Do not select Tau which are already identified as electron or muon...
  int itautag=0;
  float etaumax=0;
  AlTau* seltau;
  TauI  Itau;
  cosjmin=1;
  for (Itau=altaup.begin(); Itau<altaup.end(); Itau++)
    {
      int nch=0;
      bool keep=true;
      nch=(*Itau)->getNch();
      if(imutag>0 || ieltag>0)
for (ObI Ifl=(*Itau)->getObjects().begin();Ifl<(*Itau)->getObjects().end();Ifl++)
  if((((AlEflw*)(*Ifl))->getTrack())==theElTrack ||
     (((AlEflw*)(*Ifl))->getTrack())==theMuTrack) keep=false;
      if(Pmiss.QP()>0 && (keep))
{
  cosjmiss= (*Itau)->QCOSA(Pmiss);
  if(cosjmiss < cosjmin)
```

```
    {
      cosjmin=cosjmiss;
      etaumax=(*Itau)->QE();
      seltau=(*Itau);
      itautag=nch;
    }
}
    }
```

At this stage, the calculation is only completed for good events. Two jets are made using the Durham PE algorithm from the energy flows. The DurhamJet routine produces exactly 2 jets (Ycut=-2). Each jet contains a vector its associated Objects. The acolinearity and acoplanarity of those two jets are calculated and the DurhamYn routine is also applied.

```
  if(GoodEvent){
    // Compute acoplanarity and acolinearity
    //     ( Make 2 jets using Durham PE for a fixed number of jet,
    //       enter a negative values of Ycut = -number of jets!    )
    int jetr;
    float Ycut = -2.;// force 2 jets
    int scheme = 1;  // E-scheme
    AlephVector<AlJet> Jn = alefp.DurhamJet(Ycut, scheme, 0);
    jetr = Jn.size();
    float acopl=180.;
    float acolin=90.;
    if(jetr == 2)
      {
acopl= acos(min(max(cos(Jn[0].QPH()-Jn[1].QPH()),-1.),1.))*QQRADP;
acolin= acos(min(max(Jn[1].QCOSA(Jn[0]),(float)-1),(float)1))*QQRADP;
      }
// Calculate Yi variables
    vector<float> Yn = alefp.DurhamYn(scheme,0.);
```

If a tau lepton has been found, acolinearity, acoplanarity and the total invariant mass are computed without the energy flow objects which belong to the tau candidate. The way those quantities are calculated is similar to the previous ones. Just note than we lock recursively the tracks belonging to the tau with Lock(1).

```
// If there is a tau, compute msumww acoplww acolww without the tau
    float msumww  = 0.;
    float acoplww = 180.;
    float acolww  = 90.;
    AlephVector<AlJet> Jn2;
    if(itautag>0)
      {
```

```
// lock all tracks belonging to the Tau.(use the recursive lock)
seltau->Lock(1);
// and calculate sumww and acollinww
Jn2 = alefp.DurhamJet(Ycut, scheme, 0);
// first set the Sort Criterium ( component of the 4-vector )
// the default value is 3, so that we may forget this line
        QvecBase::SortCriterium = 3;
// sort the vector using the component 3 (e) of A4V
        sort(Jn2.begin(),Jn2.end());
// compute the invariant mass of the 2 jets
msumww = Jn2[0].QM2(Jn2[1]);
jetr = Jn2.size();
// compute acoplanarity and acolinearity
if(jetr == 2)
  {
    acoplww= acos(min(max(cos(Jn2[0].QPH()-Jn2[1].QPH()),-1.),1.))*QQRADP;
    acolww= acos(min(max(Jn2[1].QCOSA(Jn2[0]),(float)-1),(float)1))*QQRADP;
  }
seltau->unLock(1);
      }
```

Finally, the useful variables are copied and the ntuple is filled.

```
// Prepare output...
    myvars[0]  = Nchsel;              //nch
    myvars[1]  = Echsel;              //ech
...
    myvars[25] = Yn[0];               //y12
    myvars[26] = Yn[1];               //y23
    myvars[27] = Yn[2];               //y34
    myvars[28] = msumww;              //msumww
    myvars[29] = acoplww;             //acoplww
    myvars[30] = acolww;              //acolww
    // Fill ntuple
    HFN(1000,myvars);
  }
  else
    cerr << "bad event" << endl;
// delete temporary array
  delete [] myvars;
}
```

7 Results of the *ALPHA*-*ALPHA++* comparison.

So far, only rough estimations of the matching between *ALPHA* and *ALPHA++* using basic tools had been performed. The goal of this work was to compare the result of a full analysis, using all the existing algorithms. In order to compare the results, the two codes were executed on the same dataset (WW CC03 Monte Carlo sample) and each reconstructed quantity was then compared on an event by event basis.

In order to illustrate the degree of compatibility of the results, the Fig. 3 shows the distributions of the relative difference between the total invariant mass reconstructed with *ALPHA* and *ALPHA++* (top histogram). The bottom histogram of Fig. 3 shows the relative difference between the reconstructed most energetic jet energy obtained after having locked the energy flow objects which belong to the tau candidate.

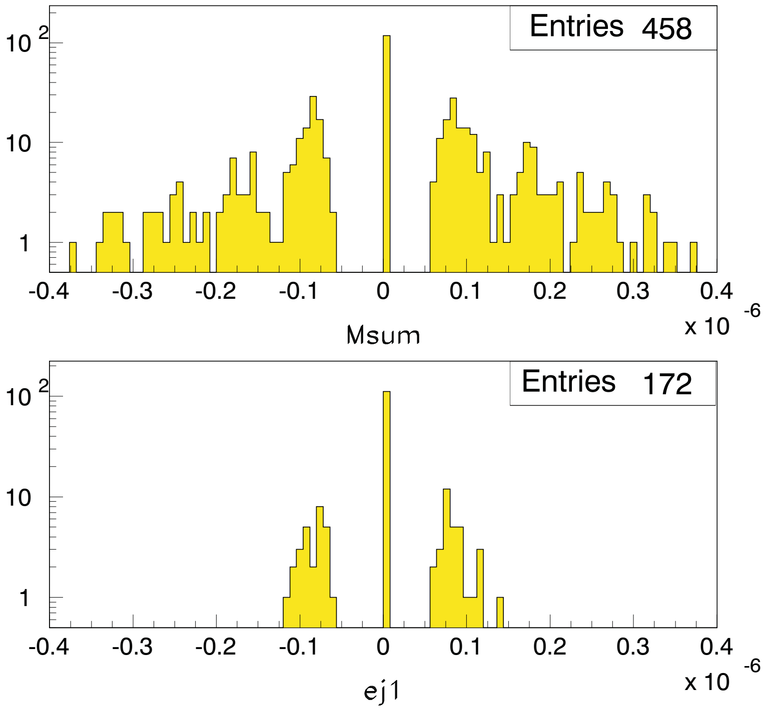

Figure 3: Relative difference between *ALPHA* and *ALPHA++* for the total invariant mass and the most energetic jet energy. The latter is formed from events where the energy flow objects which belong to the tau candidate have been locked (and therefore, not used in the jet algorithm).

The results from *ALPHA* and *ALPHA++* may be considered as exactly equivalent. The only difference can be assimilated with the machine floating number representation (a relative difference of the order of 10^{-7}). The structures which appear in this plot are understood by the rounding feature which cannot be avoided in the difference and which are a function of the value of the plotted quantity. The number of entries in each histogram represent the number of events for which the plotted quantity make sense. Thus, the bottom histogram of Fig. 3 has 172 entries which correspond to the number of times a tau has been identified in the Monte-Carlo sample used.

For completeness, Fig. 4 shows the relative difference for the energy of the electron, muon and tau candidates.

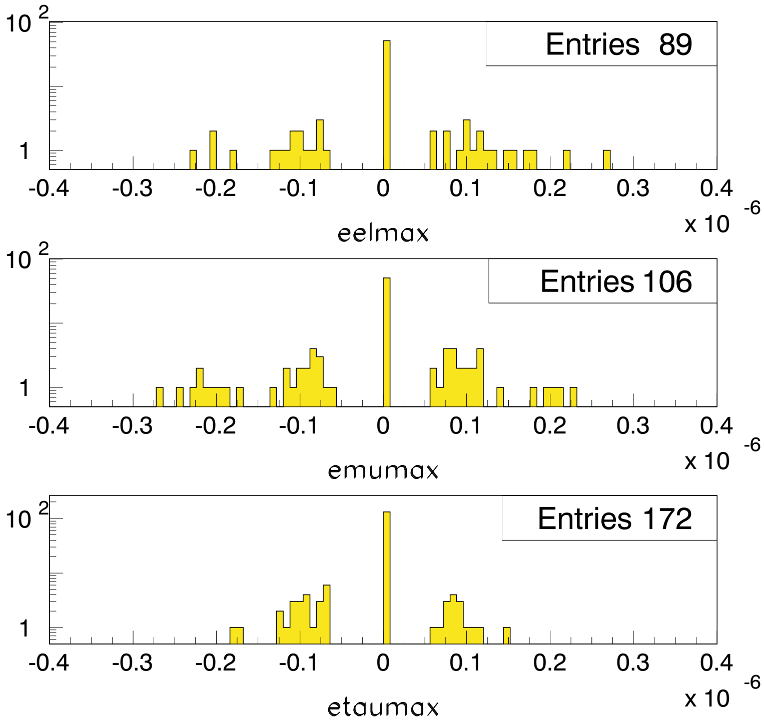

Figure 4: Relative difference between *ALPHA* and *ALPHA++* for the energy of the electron,muon and tau candidates.

References

[1] http://alephwww.cern.ch/ disserto/objty/

[2] ALEPH Collaboration, Single W Production at Energies up to $\sqrt{s} = 202GeV$ and Search for Anomalous Triple Gauge Boson Couplings, *ALEPH*/00-054, Moriond2000

[3] Dan Wallin, Further developments and testing of *ALPHA++*, Summer Student Report.

Appendices

A Single W

```
/////////////////////////////////////////////////////////////////////
// User analysis routines for hadronic selection
//
// Author : V. Lemaitre, C. Delaere V. Perez Reale
//
// Date: 2000-09-10
//
/////////////////////////////////////////////////////////////////////


#include <fstream>
#include "AlephExManager.h"
#include "hbook.h"
#include "AlephVector.h"
#include <vector>

extern ofstream fout;

///////////////////////////////////////////////////////////////////////
//  User Init
///////////////////////////////////////////////////////////////////////

static int Ngood = 0;

void AlephExManager::UserInit()
{
  // book an ntuple
  int ntags = 31;
  char mytags[31][8] = {"nch    ","ech    ","e12f   ","econe  ","msum   ",
"cthmiss","ewedge ","ptmiss ","mmiss  ","imutag ",
"ieltag ","itautag","emumax ","eelmax ","etaumax",
"acopl  ","acolin ","pxj1   ","pyj1   ","pzj1   ",
"ej1    ","pxj2   ","pyj2   ","pzj2   ","ej2    ",
                        "y12    ","y23    ","y34    ",
                        "msumww ","acoplww","acolww "};

  HBOOKN(1000,"myNtuple",ntags," ",5000,mytags);
}

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


///////////////////////////////////////////////////////////////////////
//  User Event
///////////////////////////////////////////////////////////////////////
void AlephExManager::UserEvent(AlphaBanks& bb)
{
// some definitions
  typedef AlephVector<AlTrack *>::iterator TI;
  typedef AlephVector<AlMuon *>::iterator MuI;
  typedef AlephVector<AlElec *>::iterator ElI;
  typedef AlephVector<AlEflw *>::iterator FlI;
  typedef AlphaBanks::AlObjectVec::iterator ObI;
  typedef AlephVector<AlTau *>::iterator TauI;
  float QQRADP=180./M_PI;
  float QELEP=188.6;
  TI Ich;
  MuI Imu;
  ElI Iel;
  FlI Ifl;

// get vectors of pointers to the objects used in the analysis
  AlephVector<AlTrack *> altp=bb.TrackPV();
  AlephVector<AlMuon *> almup=bb.MuonPV();
  AlephVector<AlElec *> alelp=bb.ElecPV();
  AlephVector<AlEflw *> alefp=bb.EflwPV();

// Ntuple variable initialization
  int nvars = 31;
  float* myvars = new float[nvars];
  for (int i=0;i<nvars; i++) myvars[i]=0.;

// Good quality charged Tracks selection
  int Nchsel=0;
  float Echsel=0.;
  for (Ich=altp.begin(); Ich<altp.end(); Ich++)
    {
      int Ntpc = (*Ich)->NT();
      float costhch = (*Ich)->QCT();
      float d0 = (*Ich) -> QDB();
      float z0 = (*Ich) -> QZB();

      if ((Ntpc<4)||(costhch>0.95)||(d0>2.)||(z0>10.)
  ||(costhch<-.95)||(d0<-2)||(z0<-10))
{
```

```
  (*Ich)->Lock();           // Lock the tracks (for leptons algos)
  altp.looperase(Ich);      // Erases elements in the altp vector
}
      else
{
  Nchsel++;
  Echsel += (*Ich)->QE();
}
    }

// now select the event according to the good charged tracks
  bool GoodEvent = false;
  if ((Nchsel >= 5) && (Echsel >= 5.))
    {
      // now we have a good event
      Ngood++;
      GoodEvent = true;
    }
  else
    cout << "bad event" << endl;

// Initialize leptons loops
  float psummin=4.715;
  float Yjade = 0.001;

  //filter muons. parameters are (d0max,pmin,minhit,cosmax,z0max)
  almup.FilterMu(.5,9.43,4,.95,10);

  //filter elecs.parameters are
  //            (almup,d0max,pmin,minhit,cosmax,z0max,
  //             nwirem,r3cut,r2cut0l,r2cut1l,r2cut2l,
  //             r2cuth,r5cutl,r5cuth)
  alelp.FilterEl(almup,.5,(0.05*QELEP),4,.95,10,
 0,1000,-3,-7,-5,
 1000,-0.5,1000);

  //generate raw taus candidates. Parameters are (Ycut,Scheme,Metric,Qelep)
  AlephVector<AlTau> JJ = alefp.ATauJn(Yjade, 1,1,QELEP);
  AlephVector<AlTau*> altaup = JJ.Pointers();
  //filter taus. Parameters are (psummin,d0max,minhit,cosmax,z0max)
  altaup.FilterTau(psummin,2.,4,.95,10.);

// Loop over energy flow objects

  QvecBase Psum = alefp.GetSum();
  float esum = Psum.QE();
  float psum = Psum.QP();
  float msum = Psum.QM();
  QvecBase Pmiss = alefp.GetMiss(QELEP);
  float cthmiss=Pmiss.QCT();
  // wedge energy in 30 degree
  float ewedge=0;
  if (Pmiss.QP() >0)
    {
      for(Ifl = alefp.begin(); Ifl < alefp.end(); Ifl++)
if((acos(min(max(cos(Pmiss.QPH()-((*Ifl)->QPH())),-1.),1.)))*QQRADP<30)
  ewedge+= (*Ifl)->QE();
    }
  else
    GoodEvent=false;
  //energy in 12 deg cone centred aound beam pipe
  float e12_av=0;
  float econe =0;
  for(Ifl = alefp.begin(); Ifl < alefp.end(); Ifl++)
    {
      int type = (*Ifl)->getEfType();
      float costheta = fabs((*Ifl)->QCT());
      const double c12=0.9781476;
      float nint=(*Ifl)->QCH();
      if((type == 7) || (type == 8) || ((nint ==0) && (costheta>0.9781476)))
e12_av += (*Ifl)->QE();
      if(costheta > c12)
econe += (*Ifl)-> QE();
    }

// Select the most anti-parallel (to Pmiss)
// identified electron or  muon
  int imutag=0;
  int ieltag=0;
  float emumax=0;
  float eelmax=0;
  float cosjmiss;
  AlMuon selmuon;
  AlElec selelec;
  AlTrack* theElTrack;
  AlTrack* theMuTrack;
  float cosjmin=1;
  for (Imu=almup.begin(); Imu<almup.end(); Imu++)
    if(Pmiss.QP() > 0)
      {
cosjmiss=(*Imu)->QCOSA(Pmiss);
if(cosjmiss <= cosjmin)
  {
```

```
    cosjmin=cosjmiss;
    emumax= (*Imu)->QE();
    selmuon =(**Imu);
    theMuTrack = (**Imu).getTrack();
    imutag =1;
  }
      }
  cosjmin=1;
  for (Iel=alelp.begin(); Iel<alelp.end(); Iel++)
    if(Pmiss.QP() > 0)
      {
cosjmiss=(*Iel)->QCOSA(Pmiss);
if(cosjmiss <= cosjmin)
  {
    cosjmin=cosjmiss;
    eelmax= (*Iel)->QE();
    selelec =(**Iel);
    theElTrack = (**Iel).getTrack();
    ieltag =1;
  }
      }

// Select the most anti-parallel (to Pmiss)
// identified tau with energy > 0.05*ELEP
// Do not select Tau which are already identified as electron or muon...
  int itautag=0;
  float etaumax=0;
  AlTau* seltau;
  TauI  Itau;
  cosjmin=1;
  for (Itau=altaup.begin(); Itau<altaup.end(); Itau++)
    {
      int nch=0;
      bool keep=true;
      nch=(*Itau)->getNch();
      if(imutag>0 || ieltag>0)
for (ObI Ifl=(*Itau)->getObjects().begin();Ifl<(*Itau)->getObjects().end();Ifl++)
  if((((AlEflw*)(*Ifl))->getTrack())==theElTrack ||
     (((AlEflw*)(*Ifl))->getTrack())==theMuTrack) keep=false;
      if(Pmiss.QP()>0 && (keep))
{
  cosjmiss= (*Itau)->QCOSA(Pmiss);
  if(cosjmiss < cosjmin)
    {
      cosjmin=cosjmiss;
      etaumax=(*Itau)->QE();
      seltau=(*Itau);
      itautag=nch;
    }
}
    }

  // normalize parameters
  float e12f;
  if (QELEP > 0)
    {
      eelmax=eelmax/QELEP;
      emumax=emumax/QELEP;
      etaumax=etaumax/QELEP;
      ewedge=ewedge/QELEP;
      econe = econe/QELEP;
      psum = psum/QELEP;
      e12f= e12_av/QELEP;
    }

  if(GoodEvent){
    // Compute acoplanarity and acolinearity
    //     ( Make 2 jets using Durham PE for a fixed number of jet,
    //       enter a negative values of Ycut = -number of jets!    )
    int jetr;
    float Ycut = -2.;// force 2 jets
    int scheme = 1;  // E-scheme
    AlephVector<AlJet> Jn = alefp.DurhamJet(Ycut, scheme, 0);
    jetr = Jn.size();
    float acopl=180.;
    float acolin=90.;
    if(jetr == 2)
      {
acopl= acos(min(max(cos(Jn[0].QPH()-Jn[1].QPH()),-1.),1.))*QQRADP;
acolin= acos(min(max(Jn[1].QCOSA(Jn[0]),(float)-1),(float)1))*QQRADP;
      }
// Calculate Yi variables
    vector<float> Yn = alefp.DurhamYn(scheme,0.);

// If there is a tau, compute msumww acoplww acolww without the tau
    float msumww  = 0.;
    float acoplww = 180.;
    float acolww  = 90.;
    AlephVector<AlJet> Jn2;
    if(itautag>0)
      {
// lock all tracks belonging to the Tau.(use the recursive lock)
seltau->Lock(1);
```

```
// and calculate sumww and acollinww
Jn2 = alefp.DurhamJet(Ycut, scheme, 0);
// first set the Sort Criterium ( component of the 4-vector )
// the default value is 3, so that we may forget this line
        QvecBase::SortCriterium = 3;
// sort the vector using the component 3 (e) of A4V
        sort(Jn2.begin(),Jn2.end());
// compute the invariant mass of the 2 jets
msumww = Jn2[0].QM2(Jn2[1]);
jetr = Jn2.size();
// compute acoplanarity and acolinearity
if(jetr == 2)
  {
    acoplww= acos(min(max(cos(Jn2[0].QPH()-Jn2[1].QPH()),-1.),1.))*QQRADP;
    acolww= acos(min(max(Jn2[1].QCOSA(Jn2[0]),(float)-1),(float)1))*QQRADP;
  }
seltau->unLock(1);
      }

// Prepare output...

    myvars[0]  = Nchsel;              //nch
    myvars[1]  = Echsel;              //ech
    myvars[2]  = e12f;                //forward E
    myvars[3]  = econe;               //E in 12deg cone
    myvars[4]  = msum;                //invariant mass
    myvars[5]  = cthmiss;             //cthmiss
    myvars[6]  = ewedge;              //wedge Energy
    myvars[7]  = Pmiss.QPT();         //ptmiss
    myvars[8]  = Pmiss.QM();          //mmiss
    myvars[9]  = imutag;              //muon tag
    myvars[10] = ieltag;              //electron tag
    myvars[11] = (itautag!=0);        //tau tag
    myvars[12] = emumax;              //max E of mu
    myvars[13] = eelmax;              //max E of e-
    myvars[14] = etaumax;             //max E of Tau
    myvars[15] = acopl;               //acoplanarity
    myvars[16] = acolin;              //acolinearity
    if(itautag>0)
      {
myvars[17] = Jn2[1].QX();     //pxj1
myvars[18] = Jn2[1].QY();     //pyj1
myvars[19] = Jn2[1].QZ();     //pzj1
myvars[20] = Jn2[1].QE();     //ej1
myvars[21] = Jn2[0].QX();     //pxj2
myvars[22] = Jn2[0].QY();     //pyj2
myvars[23] = Jn2[0].QZ();     //pzj2
myvars[24] = Jn2[0].QE();     //ej2
      }
    else
      {
myvars[17] = 0;               //pxj1
myvars[18] = 0;               //pyj1
myvars[19] = 0;               //pzj1
myvars[20] = 0;               //ej1
myvars[21] = 0;               //pxj2
myvars[22] = 0;               //pyj2
myvars[23] = 0;               //pzj2
myvars[24] = 0;               //ej2
      }
    myvars[25] = Yn[0];               //y12
    myvars[26] = Yn[1];               //y23
    myvars[27] = Yn[2];               //y34
    myvars[28] = msumww;              //msumww
    myvars[29] = acoplww;             //acoplww
    myvars[30] = acolww;              //acolww

    // Fill ntuple
    HFN(1000,myvars);
  }
  else
    cerr << "bad event" << endl;

// delete temporary array
  delete [] myvars;

}

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



//////////////////////////////////////////////////////////////////////////
//  User Term
//////////////////////////////////////////////////////////////////////////
void AlephExManager::UserTerm()
{
  fout << endl << " USER TERM : Ngood = " << Ngood << endl;
}
```

B AlephVector

```
////////////////////////////////////////////////////////
// AlephVector.h
// Subclass of vector.h
//
// Author: Dan Wallin (with contributions from M. Hoerndl)
// modified by:  C. Delaere, V. Lemaitre, O. van der Aa :
//                 - added Pointers method
//                 - added Leptons filter algos
//                 - added GetSum and GetMiss
//                 - new Jet Algos (form Orca)
//
////////////////////////////////////////////////////////

#ifndef __AlephVector__
#define __AlephVector__

#include <iostream.h>
#include <vector>
#include <algorithm>
#include "AlObject.h"
#include "QvecBase.h"
#include "AlThrust.h"
#include "AlJet.h"
#include "AlTau.h"
#include "AlEflw.h"
#include "AlElec.h"
#include "AlMuon.h"
#include "CLHEP/Matrix/Matrix.h"
#include "CLHEP/Matrix/Vector.h"

// Prevents the compiler from generating several copies of
// methods during the precompiling stage, which leads to longer
// compile time.
#if defined(__GNUC__)
#pragma interface
#endif

// some additional possible naming
#define AlephSelection AlephVector;

template <class Type>
class AlephVector: public vector<Type>{

public:

  // Default Constructor
  AlephVector::AlephVector():vector<Type>(){}

  // Another Constructor
  AlephVector::AlephVector(int i):vector<Type>(i){}

  // Print all elements (debug tools)
  void print();
  void print2();

  // Add two vectors to each other
  void append(AlephVector<Type>);

  // Erase an element from the vector, should be used in loops
  void looperase(AlephVector<Type>::iterator &it){
    AlephVector::iterator tmp=it;
    it--;
    erase(tmp);
  }

  // Create a vector of pointers from a vector of objects
  AlephVector<Type *> Pointers()
    {
      AlephVector<Type *> out;
      for (unsigned int i=0;i<size();i++) out.push_back(&((*this)[i]));
      return out;
    }

  // The Thrust algorithm
  AlThrust AThrust();

  // DurhamJet (jet clustering)
  AlephVector<AlJet> DurhamJet(float theYcut, int Scheme=1, float Qelep=0.);
  // JadeJet (jet clustering)
  AlephVector<AlJet> JadeJet(float theYcut, int Scheme=1, float Qelep=0.);

  // ATauJn (Tau candidates generator)
  AlephVector<AlTau> ATauJn(float Ycut, int Scheme=0, int Metric=1, float Qelep=0);

  // DurhamYn
  vector<float> DurhamYn(int, float Qelep=0.);

  // Filter Leptons
  void FilterMu(float d0max=.5,float pmin=1,int minhit=4, float cosmax=.95,float z0max=10);
  void FilterEl(const AlephVector<AlMuon *>& almup,
```

```
float d0max=.5,float pmin=1,int minhit=4, float cosmax=.95,float z0max=10,
int nwirem=40,float r3cut=1000,float r2cut0l=-3,
float r2cut1l=-7,float r2cut2l=-5,float r2cuth=1000,
float r5cutl=-0.5,float r5cuth=1000);
  void FilterTau(float psummin=1, float d0max=2, int minhit=4,
 float cosmax=.95, float z0max=10);

  QvecBase GetSum();
  QvecBase GetMiss(float QELEP=0.);

};

#if defined(__GNUC__)
#pragma implementation
#endif
// Implementations...

// Debugging print function
template <class Type>
void AlephVector<Type>::print(){
  iterator i;
  int ii=1;
  for (i=begin();i<end();i++){
    cout << ii << " My vector: " << *i << " " << (**i).QX() << endl;
    ii++;
  }
}

// Debugging print function
template <class Type>
void AlephVector<Type>::print2(){
  iterator i;
  int ii=1;
  for (i=begin();i<end();i++){
    cout << ii << " My vector: " << i << " " << *i << endl;
    ii++;
  }
}

// Appends a vector of a certain kind to another vector
template <class Type>
void AlephVector<Type>::append(AlephVector<Type> cv)
{
  for(iterator i=cv.begin();i<cv.end();i++) push_back(*i);
}

// Sum all the elements
template <class Type>
QvecBase AlephVector<Type>::GetSum()
{
  QvecBase Psum;
  HepLorentzVector P_4sum(0.,0.,0.,0.);
  float chtot=0.;
  iterator it;
  for (it=begin();it<end();it++)
    if (!((*it)->isLocked()))
      {
if ((*it)->TYPE() == ALTHRUST )
  {
    cout << "WARNING Type is AlThrust: type not supported !" << endl;
    exit(1);
  }
P_4sum += (*it)->A4V();
chtot += (*it)->QCH();
      }
  Psum.setA4V(P_4sum);
  Psum.setQCH(chtot);
  return Psum;
}

// Find the missing momentum
template <class Type>
QvecBase AlephVector<Type>::GetMiss(float QELEP)
{
  if(QELEP==0.) cout << "WARNING, Pmiss calculated with QELEP=0 !" << endl;
  QvecBase Pmiss;
  HepLorentzVector P_4miss(0.,0.,0.,QELEP);
  float chtot=0.;
  iterator it;
  for (it=begin();it<end();it++)
    if (!((*it)->isLocked()))
      {
if ((*it)->TYPE() == ALTHRUST )
  {
    cout << "WARNING Type is AlThrust: type not supported !" << endl;
    exit(1);
  }
P_4miss -= (*it)->A4V();
chtot -= (*it)->QCH();
      }
  P_4miss.setE(max(P_4miss.e(),P_4miss.rho()));
  Pmiss.setA4V(P_4miss);
  Pmiss.setQCH(chtot);
```

```
  return Pmiss;
}

// AlThrust algorithm
#include "AlephAlgoThrust.h"

// Jet clustering
#include "AlephDurhamAlgo.h"
#include "AlephJadeAlgo.h"

// Leptons filtering
#include "AlephAlgoLeptons.h"

#endif


////////////////////////////////////////////////////////
//
// Implementation of Class AlephVector
//
// Author: Dan Wallin (with contributions from M. Hoerndl)
// modified by:  C. Delaere, V. Lemaitre, O. van der Aa :
//                 - added Pointers method
//                 - added Leptons filter algos
//                 - added GetSum and GetMiss
//                 - new Jet Algos (form Orca)
//
////////////////////////////////////////////////////////

#include "AlephVector.h"

float  max(float  i, float j)  { return i > j ? i : j; }
float  max(double i, float j)  { return i > j ? i : j; }
double max(double i, double j) { return i > j ? i : j; }
float  min(float  i, float j)  { return i < j ? i : j; }
float  min(double i, float j)  { return i < j ? i : j; }
double min(double i, double j) { return i < j ? i : j; }
```

C QvecBase

```
/////////////////////////////////////////////////////////
//
// CLASS QvecBase : is independent of Fortran memory
//                  and serves as base for tracks,
//                  eflows etc.
//
// Author : G. Bagliesi
// modified by C. Delaere: - modified lock (recursive)
//                         - added operator <
//                           with a variable SortCriterium
//
/////////////////////////////////////////////////////////


#ifndef _QVECBASE_H_
#define _QVECBASE_H_

#include "AlObject.h"
#include "AlphaCopy.h"

// the class, inherits from AlObject
class QvecBase : public AlObject {

public:

  // constructor
  QvecBase();

  //copy constructor
  QvecBase(const QvecBase & origin);

  // destructor
  ~QvecBase(){}

  // Compare two objects...
  bool operator<(const QvecBase& other);

  // set the Lorentz Vector
  virtual void setA4V(float x, float y, float z, float e);
  virtual void setA4V(HepLorentzVector vec){_A4V = vec;}

  // return the Lorentzvector
  virtual HepLorentzVector A4V();

  // set the charge
  virtual void setQCH(float qch){_qch = qch;}

  // get the type
  virtual ALEPHTYPE TYPE(){return OBJECT;}

  // copies of the ALPHA utility functions (see ALPHA manual)
  virtual float sign(float k);
  virtual float QP();
  virtual float QX();
  virtual float QY();
  virtual float QZ();
  virtual float QE();
  virtual float QM();
  virtual float QCH();
  virtual float QCT();
  virtual float QPH();
  virtual float QPT();
  virtual float QBETA();
  virtual float QGAMMA();
  virtual float QMSQ2(QvecBase j);
  virtual float QM2(QvecBase j);
  virtual float QDMSQ(QvecBase j);
  virtual float QPPAR(QvecBase j);
  virtual float QPPER(QvecBase j);
  virtual float QDOT3(QvecBase j);  //3 dim cross product
  virtual float QDOT4(QvecBase j);  //4 dim cross product
  virtual float QCOSA(QvecBase j);
  virtual float QDECA2(QvecBase j);
  virtual float QDECAN(QvecBase j);

  // add two objects
  virtual QvecBase add(QvecBase j);

  // get decay angle
  virtual HepDouble DECAY_ANG(QvecBase j);

  // the locking
  virtual void Lock(bool recurse=false );   // lock the track
  virtual void unLock(bool recurse=false ); // unlock the track
  virtual int  isLocked();   // =0 if not locked, else locked

  // sort criterium
  // [0,1,2,3] = [px,py,pz,e]
  // default is 3 = e
  static int SortCriterium;
```

```
protected:
  HepLorentzVector _A4V; //this is a data member containing the instance
                         // of the four vector

  int _locked;           // lock status
  float  _qch;           // charge

private:
  static int NERR;
};

#endif


////////////////////////////////////////////////////////////
//
// Implementation of
// CLASS QvecBase : is independent of Fortran memory
//                  and serves as base for tracks,
//                  eflows etc.
//
// Author : G. Bagliesi
// modified by C. Delaere: - modified lock (recursive)
//                         - added operator <
//                           with a variable SortCriterium
//
////////////////////////////////////////////////////////////

#include <vector>
#include <stdlib.h>
#include <math.h>
#include "QvecBase.h"


int QvecBase::NERR=0;

// constructor
QvecBase::QvecBase()
{
  _locked = 0;
  setA4V(0.,0.,0.,0.);
  _qch    =  0;
}

//copy constructor
QvecBase::QvecBase(const QvecBase& origin)
{
 _A4V    = origin._A4V;
 _locked = origin._locked;
 _qch    = origin._qch;
}

// compare 2 objects...
// the criterium is fixed by the static member [SortCriterium]
// [0,1,2,3] = [px,py,pz,e]
// 4 = Pt
// default is 3 = e
bool QvecBase::operator<(const QvecBase& other)
  {
    if(SortCriterium>3)
      return ( QPT() < other.QPT());
    else
      return (_A4V[SortCriterium] < other._A4V[SortCriterium]);
  }

float QvecBase::sign(float k) {return k/fabs(k);}

// return the Lorentz vector
HepLorentzVector QvecBase::A4V() { return _A4V; }

// set the Lorentz vector
void QvecBase::setA4V(float x, float y, float z, float e)
 {
   _A4V.setPx(x);
   _A4V.setPy(y);
   _A4V.setPz(z);
   _A4V.setE(e);
 }


// add two objects
QvecBase QvecBase::add(const QvecBase j)
 {
  HepLorentzVector A4Temp = A4V() + j.A4V();
  QvecBase Temp;
  Temp.setA4V(A4Temp);
  Temp.setQCH(QCH() + j.QCH());
  return Temp;
 }

// appreviations for standard ALPHA functions
float QvecBase::QCH(){return _qch;}
float QvecBase::QX() {return A4V().px();}
float QvecBase::QY() {return A4V().py();}
```

```
float QvecBase::QZ() {return A4V().pz();}
float QvecBase::QP() {return A4V().rho();}
float QvecBase::QE() {return A4V().e();}
float QvecBase::QM() {return A4V().m();}
float QvecBase::QCT() { return A4V().cosTheta();}
float QvecBase::QPH() { return A4V().phi(); }
float QvecBase::QPT() { return A4V().perp();}
float QvecBase::QBETA() { return A4V().rho()/A4V().e();}
float QvecBase::QGAMMA() { return 1./sqrt((1.-QBETA())*(1.+QBETA())); }


float QvecBase::QMSQ2(QvecBase j)
{
  HepLorentzVector A4Temp = A4V() + j.A4V();
  return A4Temp.m2();
}

float QvecBase::QM2(QvecBase j)
{
  HepLorentzVector A4Temp = A4V() + j.A4V();
  return A4Temp.m();
}

float QvecBase::QDMSQ(QvecBase j)
{
  HepLorentzVector A4Temp = A4V() - j.A4V();
  return A4Temp.m2();
}

float QvecBase::QPPAR(QvecBase j) { return QDOT3(j)/j.A4V().rho();}
float QvecBase::QPPER(QvecBase j) { return A4V().perp(j.A4V()); }
float QvecBase::QDOT3(QvecBase j) { return Hep3Vector(A4V()).dot(j.A4V());}
float QvecBase::QDOT4(QvecBase j) { return A4V().dot(j.A4V());}

float QvecBase::QCOSA(QvecBase j)
{
  return cos(A4V().angle(Hep3Vector(j.A4V())));
}

float QvecBase::QDECA2(QvecBase j)
  {
    double P_1[4], P_2[4];
    P_1[0] = (double) A4V().px();
    P_1[1] = (double) A4V().py();
    P_1[2] = (double) A4V().pz();
    P_1[3] = (double) A4V().e();
    double PMAG = sqrt(pow(P_1[0],2) + pow(P_1[1],2) + pow(P_1[2],2));
    P_2[0] = (double) j.A4V().px();
    P_2[1] = (double) j.A4V().py();
    P_2[2] = (double) j.A4V().pz();
    P_2[3] = (double) j.A4V().e();
    double PP = sqrt( pow((P_1[0]+P_2[0]),2) + pow((P_1[1]+P_2[1]),2)
                              + pow((P_1[2]+P_2[2]),2));
    double BETA = PP / ( P_1[3]+P_2[3] );
    double PPAR = ( P_1[0] *(P_1[0]+P_2[0]) + P_1[1] *(P_1[1]+P_2[1])
                              + P_1[2] *(P_1[2]+P_2[2]) )/PP;
    PP = PPAR - BETA * P_1[3];
    return (float) (PP / sqrt(pow(PP,2.) + (PMAG-PPAR)*(PMAG+PPAR)*(1.-BETA)*(1.+BETA)));
  }

HepDouble QvecBase::DECAY_ANG(QvecBase j)
 {
  cout << "before addition" << endl;
  QvecBase temp= (*this).add(j);
  cout << "after addition" << endl;
  HepLorentzVector v = temp.A4V();
  cout << "v written" << endl;
  HepLorentzVector v1 = A4V();
  cout << "v = " << v << " v1 = " << v1 << endl;
  v1.boost(-v.boostVector());
  cout << "boosted v1 = " << v1 << endl;
  return cos(v.angle(Hep3Vector(v1)));
 }



float QvecBase::QDECAN(QvecBase j)
  {
    float PP =A4V().rho();
    float BETA = PP / A4V().e();
    float PPAR = (A4V().px() * j.A4V().px() + A4V().py() * j.A4V().py() + A4V().pz() * j.A4V().pz()) / PP;
    PP = PPAR - BETA * j.A4V().e();
    float Q_DECAN = pow(PP,2) + (j.A4V().rho() - PPAR) * (j.A4V().rho() + PPAR) *
                                              (1. - BETA) * (1. + BETA);
     if (Q_DECAN > 0.)
      { return PP / sqrt (Q_DECAN); }
     else
       {NERR++;
if (NERR <= 10)
            { cout << endl << "_QDECAN_ 2nd particle is not a decay product of 1st" << endl; }
        return 1.;
       }
  }
```

```
// the locking
void  QvecBase::Lock(bool recurse)     { _locked = 1 ; }
void  QvecBase::unLock(bool recurse)   { _locked = 0 ; }
int   QvecBase::isLocked() { return _locked; }


// default Sort Criterium
int   QvecBase::SortCriterium = 3;
```

D QvecLink

```
////////////////////////////////////////////////////////
//
// CLASS QvecLink : holds pointer to QVEC, is independent
//                  of Fortran memory and serves as base
//                  for tracks, eflows and gampecs.
//
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
////////////////////////////////////////////////////////


#ifndef _QVECLINK_H_
#define _QVECLINK_H_

#include "QvecBase.h"
#include "AlphaCopy.h"


class QvecLink : public QvecBase  {

public:

  // constructor
  QvecLink();

  //copy constructor
  QvecLink(const QvecLink & origin);

  // destructor
  virtual ~QvecLink(){}

  // set the pointer to the QVEC object
  virtual void SetPointer(QVEC* qvec);

  virtual float QDB();
  virtual float QZB();
  virtual float QDBS2();
  virtual float QZBS2();
  virtual float QBC2();
  virtual int   ITK();

private:

  // pointer to QVEC
  QVEC* qvec;
//  HepLorentzVector _A4V;
};

#endif


////////////////////////////////////////////////////////
//
// Implementation of
// CLASS QvecLink : holds pointer to QVEC, is independent
//                  of Fortran memory and serves as base
//                  for tracks, eflows and gampecs.
//
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
////////////////////////////////////////////////////////

#include "QvecLink.h"

// constructor
QvecLink::QvecLink()
{
  qvec = NULL;
}

//copy constructor
QvecLink::QvecLink(const QvecLink& origin):QvecBase(origin)
{
  qvec = origin.qvec;
}

// set the pointer to the memory
void  QvecLink::SetPointer(QVEC* pointer)
{
  qvec = pointer;
  setA4V(qvec->QX(), qvec->QY(), qvec->QZ(), qvec->QE());
  _qch = qvec->QCH();
}

float QvecLink::QDB()   { return qvec->DB(); }
float QvecLink::QZB()   { return qvec->ZB(); }
float QvecLink::QDBS2() { return qvec->SD(); }
float QvecLink::QZBS2() { return qvec->SZ(); }
float QvecLink::QBC2()  { return qvec->CB(); }
int   QvecLink::ITK()   { return qvec->TN(); }
```

E AlEflw

```
///////////////////////////////////////////////////////////
//
// CLASS AlEflw:
// Concrete class for Eflow Objects, inherits from QvecBase
//
// Author :  G. Bagliesi
// modified by C. Delaere: now derive from QvecLink
//                         added lock and unlock
//                         fixed assignement operator
//
////////////////////////////////////////////////////////////


#ifndef _ALEFLW_H_
#define _ALEFLW_H_

#include "QvecLink.h"
#include "AlTrack.h"

class AlEflw : public QvecLink {

public:

  // constructor
  AlEflw();

  // copy constructor
  AlEflw(const AlEflw&);

  // get the type
  ALEPHTYPE TYPE(){return EFLOW;}

  // set the Eflow type
  void setEfType(const int& type);

  // get the Eflow type
  int  getEfType();

  // set the pointer to the assoc. track
  void setTrack(AlTrack* theTrack);

  // get the pointer to the assoc. track
  AlTrack* getTrack();

  // specialised lock
  void Lock(bool recurse = 0);     // lock the Eflw
  void unLock(bool recurse = 0);   // unlock the Eflw

private:

  // the Eflow type
  int _efType;
  // pointer to associated track if type <= 3
  AlTrack* _theAssocTrack;
};

#endif


///////////////////////////////////////////////////////////
//
// CLASS AlEflw:
// Concrete class for Eflow Objects, inherits from QvecBase
//
// Author :  G. Bagliesi
// modified by C. Delaere: now derive from QvecLink
//                         added lock and unlock
//                         fixed assignement operator
//
////////////////////////////////////////////////////////////

#include "AlEflw.h"

// default constructor :
AlEflw::AlEflw()  {}

// copy constructor :
AlEflw::AlEflw(const AlEflw& oldE):QvecLink(oldE) {}

// set the Eflow type
void AlEflw::setEfType(const int& type)
{
  _efType = type;
}


// get the Eflow type
int AlEflw::getEfType()
{
  return _efType;
}
```

```
// set the assoc. track
void AlEflw::setTrack(AlTrack* theTrack)
{
  _theAssocTrack = theTrack;
}

// get the assoc. track
AlTrack* AlEflw::getTrack()
{
  return _theAssocTrack;
}

// specialised lock
void AlEflw::Lock(bool recurse)
{
  if (recurse && (_theAssocTrack != NULL) ) _theAssocTrack->Lock(1);
  _locked = 1;
}
void AlEflw::unLock(bool recurse)
{
  if (recurse && (_theAssocTrack != NULL) ) _theAssocTrack->unLock(1);
  _locked = 0;
}
```

F AlElec

```
///////////////////////////////////////////////////////////
//
// CLASS AlElec:
// Concrete class for AlElec Objects, inherits from "QvecBase"
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
////////////////////////////////////////////////////////////


#ifndef _ALELEC_H_
#define _ALELEC_H_

#include "QvecBase.h"
#include "AlTrack.h"

class AlElec : public QvecBase {

public:

  // constructor
  AlElec();

  // copy constructor
  AlElec(const AlElec&);

  // get the type
  ALEPHTYPE TYPE(){return ELEC;}

// * Overwrite of general QvecBase Methods

  void Lock(bool recurse = 0);     // lock the track
  void unLock(bool recurse = 0);   // unlock the track

// * AlElec related methods

  // set the pointer to the assoc. track
  void setTrack(AlTrack* theTrack);

  // get the pointer to the assoc. track
  AlTrack* getTrack();

  // set the pointer to EIDT_ALPHA
  void SetPointer2(EIDT_ALPHA* pointer){eidt_alpha = pointer;}

  // copy of ALPHA functions for EIDT
  int   IF2();
  float R1();
  float R2();
  float R3();
  float R4();
  float R5();
  float R6();
  float R7();
  float EC();
  int   IP();
  float E1();
  float E2();
  float E3();
  int   FR();
  int   PE();

private:
  // pointer to associated track
  AlTrack* _theAssocTrack;
  // pointer to the BOS bank
  EIDT_ALPHA* eidt_alpha;

};

#endif



///////////////////////////////////////////////////////////
//
// CLASS AlElec:
// Concrete class for AlElec Objects, inherits from "QvecBase"
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
////////////////////////////////////////////////////////////


#include "AlElec.h"

// default constructor
AlElec::AlElec()
{
  _theAssocTrack = NULL;
  eidt_alpha = NULL;
```

```
}

// copy constructor
AlElec::AlElec(const AlElec& oldEl):QvecBase(oldEl)
{
  eidt_alpha = oldEl.eidt_alpha;
  _theAssocTrack = oldEl._theAssocTrack;
}

// *Normal QvecBase methods (overwritten here)

// specialised lock

void AlElec::Lock(bool recurse)
{
  if (recurse && (_theAssocTrack != NULL) ) _theAssocTrack->Lock(1);
  _locked = 1;
}
void AlElec::unLock(bool recurse)
{
  if (recurse && (_theAssocTrack != NULL) ) _theAssocTrack->unLock(1);
  _locked = 0;
}

// *Special methods for the AlElec objects only

// set the assoc. track
void AlElec::setTrack(AlTrack* theTrack)
{
  _theAssocTrack = theTrack;
 _A4V    =  _theAssocTrack->A4V();
 _qch    =  _theAssocTrack->QCH();
}

// get the assoc. track
AlTrack* AlElec::getTrack()
{
  return _theAssocTrack;
}

int   AlElec::IF2() { if (eidt_alpha == NULL)
                        return -1;
                      else
                        return eidt_alpha->Iflag; }
float AlElec::R1() { if (eidt_alpha == NULL)
                        return -1;
                     else
                        return eidt_alpha->R1esti; }
float AlElec::R2() { if (eidt_alpha == NULL)
                        return -1.;
                     else
                        return eidt_alpha->R2esti; }
float AlElec::R3() { if (eidt_alpha == NULL)
                        return -1.;
                     else
                        return eidt_alpha->R3esti; }
float AlElec::R4() { if (eidt_alpha == NULL)
                        return -1.;
                     else
                        return eidt_alpha->R4esti; }
float AlElec::R5() { if (eidt_alpha == NULL)
                        return -1.;
                     else
                        return eidt_alpha->R5esti; }
float AlElec::R6() { if (eidt_alpha == NULL)
                        return -1.;
                     else
                        return eidt_alpha->R6esti; }
float AlElec::R7() { if (eidt_alpha == NULL)
                        return -1.;
                     else
                        return eidt_alpha->R7esti; }
float AlElec::EC() { if (eidt_alpha == NULL)
                        return  0.;
                     else
                        return eidt_alpha->ECorr; }
int   AlElec::IP() { if (eidt_alpha == NULL)
                        return -1;
                     else
                        return eidt_alpha->IPpoth; }
float AlElec::E1() { if (eidt_alpha == NULL)
                        return  0.;
                     else
                        return eidt_alpha->E2; }
float AlElec::E2() { if (eidt_alpha == NULL)
                        return  0.;
                     else
                        return eidt_alpha->E1; }
float AlElec::E3() { if (eidt_alpha == NULL)
                        return  0.;
                     else
                        return eidt_alpha->E3; }
int   AlElec::FR() { if (eidt_alpha == NULL)
```

```
                        return -1;
                     else
                        return eidt_alpha->FRFT; }
int AlElec::PE() { if (eidt_alpha == NULL)
                        return -1;
                     else
                        return eidt_alpha->PECO; }
```

G AlJet

```
//////////////////////////////////////////////////////////
//
// CLASS AlJet
// Concrete class for Jet Objects, inherits from QvecBase
//
// Author :  M. Hoerndl
// modified by C. Delaere:
//          - added lock, unlock, clearObjects, sig2,
//            conversion constructors from protojets
//          - fixed assignement operator
//
///////////////////////////////////////////////////////////


#ifndef _ALJET_H_
#define _ALJET_H_

#include <vector>
#include "QvecBase.h"
#include "DurhamProtoJet.h"
#include "JadeProtoJet.h"


class AlJet: public QvecBase {
 public:

  // default costructor
  AlJet();

  // copy constructor
  AlJet(const AlJet& oldAj);

  // conversion constructors
  AlJet(const DurhamProtoJet& protojet);
  AlJet(const JadeProtoJet&   protojet);

  ALEPHTYPE TYPE(){return ALJET;}

  vector<AlObject*>& getObjects();

  void addObject(AlObject* a);

  void clearObjects();

  void setScheme(int s);

  int  getScheme();

  void setMetric(int m);

  int  getMetric();

  double sig2() { return (4*pow( 0.2862-0.01205* (45 > QP() ? QP() : 45 )
 +0.000135*pow((45 > QP() ? QP() : 45 ),2),2));}

  // specialised lock
  void Lock(bool recurse = 0);     // lock the jet
  void unLock(bool recurse = 0);   // unlock the jet

 private:
  // objects of which jet was built
  vector<AlObject*> _objects;
  // recombination scheme
  int _scheme;
  // metric for M
  int _metric;
 };

#endif


//////////////////////////////////////////////////////////
//
// Implementation of
// CLASS AlJet
// Concrete class for Jet Objects, inherits from QvecBase
//
// Author :  M. Hoerndl
// modified by C. Delaere:
//          - added lock, unlock, clearObjects, sig2,
//            conversion constructors from protojets
//          - fixed assignement operator
//
///////////////////////////////////////////////////////////

#include "AlJet.h"

// default constructor : has to do something,
// since it is not created out of qvec
AlJet::AlJet()
{
  _objects.resize(0);
```

```
  _scheme = 0;
  _metric = 0;
}


// copy constructor : NOTE: THE VECTOR OF POINTERS
// TO ITS OBJECTS IS JUST COPIED!! COULD BE DANGEROUS...

AlJet::AlJet(const AlJet& oldAl):QvecBase(oldAl)
{
  _objects.resize(oldAl._objects.size());
  for (unsigned int i = 0; i < oldAl._objects.size(); i++)
    {
      _objects[i] = oldAl._objects[i];
    }
  _scheme = oldAl._scheme;
  _metric = oldAl._metric;
}

AlJet::AlJet(const DurhamProtoJet& protojet)
{
  float charge=0;
  setA4V(protojet.get4Momentum());
  setScheme(DurhamProtoJet::Scheme);
  setMetric(0);
  const vector<const QvecBase *> aCandConstituants = protojet.getConstituants();
  vector<const QvecBase *>::iterator aCandIter;
  //loop on constituants to fill Objects vector of AlJet
  for(aCandIter=aCandConstituants.begin();
      aCandIter!=aCandConstituants.end();
      aCandIter++)
    {
      charge += (*aCandIter)->QCH();
      addObject(*aCandIter);
    }
  setQCH(charge);
}

AlJet::AlJet(const JadeProtoJet& protojet)
{
  float charge=0;
  setA4V(protojet.get4Momentum());
  setScheme(JadeProtoJet::Scheme);
  setMetric(1);
  const vector<const QvecBase *> aCandConstituants = protojet.getConstituants();
  vector<const QvecBase *>::iterator aCandIter;
  //loop on constituants to fill Objects vector of AlJet
  for(aCandIter=aCandConstituants.begin();
      aCandIter!=aCandConstituants.end();
      aCandIter++)
    {
      charge += (*aCandIter)->QCH();
      addObject(*aCandIter);
    }
  setQCH(charge);
}


// specialised lock
void AlJet::Lock(bool recurse)
{
  if (recurse)
    for (vector<AlObject*>::iterator i = _objects.begin();i<_objects.end();i++)
      (*i)->Lock(1);
  _locked = 1;
}
void AlJet::unLock(bool recurse)
{
  if (recurse)
    for (vector<AlObject*>::iterator i = _objects.begin();i<_objects.end();i++)
      (*i)->unLock(1);
  _locked = 0;
}

vector<AlObject*>& AlJet::getObjects()
{ return _objects; }

void AlJet::addObject(AlObject* a)
{ _objects.push_back(a); }

void AlJet::clearObjects()
{ _objects.clear(); }

void AlJet::setScheme(int s)
{ _scheme = s; }

int  AlJet::getScheme()
{ return _scheme; }

void AlJet::setMetric(int m)
{ _metric = m; }

int  AlJet::getMetric()
```

```
{ return _metric; }
```

H AlMuon

```
//////////////////////////////////////////////////////////
//
// CLASS AlMuon:
// Concrete class for AlMuon Objects, inherits from "QvecBase"
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
///////////////////////////////////////////////////////////


#ifndef _ALMUON_H_
#define _ALMUON_H_

#include "QvecBase.h"
#include "AlTrack.h"

class AlMuon : public QvecBase {

public:

  // constructor
  AlMuon();

  // copy constructor
  AlMuon(const AlMuon&);

  // get the type
  ALEPHTYPE TYPE(){return MUON;}

// * Overwrite of general QvecBase Methods

  void Lock(bool recurse = 0);     // lock the track
  void unLock(bool recurse = 0);     // unlock the track


// * AlMuon related methods

  // set the pointer to the assoc. track
  void setTrack(AlTrack* theTrack);

  // get the pointer to the assoc. track
  AlTrack* getTrack();

  // set the pointer to MUID_ALPHA
  void SetPointer2(MUID_ALPHA* pointer){muid_alpha = pointer;}

  // copy of ALPHA functions for MUID
  int   IF();
  float SR();
  float DM();
  int   ST();
  int   TN();

private:
  // pointer to associated track
  AlTrack* _theAssocTrack;
  // pointer to BOs bank
  MUID_ALPHA* muid_alpha;

};

#endif


//////////////////////////////////////////////////////////
//
// Implementation of
// CLASS AlMuon:
// Concrete class for AlMuon Objects, inherits from "QvecBase"
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
///////////////////////////////////////////////////////////

#include "AlMuon.h"

// default constructor : does nothing
AlMuon::AlMuon()
{
  _theAssocTrack = NULL;
  muid_alpha = NULL;
}

// copy constructor :
AlMuon::AlMuon(const AlMuon& oldMU):QvecBase(oldMU)
{
  muid_alpha = oldMU.muid_alpha;
  _theAssocTrack = oldMU._theAssocTrack;
}

// *Normal QvecBase methods (overwritten here)
```

```
void AlMuon::Lock(bool recurse)
{
  if (recurse && (_theAssocTrack != NULL) )  _theAssocTrack->Lock(1);
  _locked = 1;
}
void AlMuon::unLock(bool recurse)
{
  if (recurse && (_theAssocTrack != NULL) )  _theAssocTrack->unLock(1);
  _locked = 0;
}

// *Special methods for the AlMuon objects only

// set the assoc. track
void AlMuon::setTrack(AlTrack* theTrack)
{
  _theAssocTrack = theTrack;
  _A4V    = _theAssocTrack->A4V();
  _qch    = _theAssocTrack->QCH();
}

// get the assoc. track
AlTrack* AlMuon::getTrack()
{
  return _theAssocTrack;
}

int   AlMuon::IF() { if (muid_alpha == NULL)
                        return -1;
                      else
                        return muid_alpha->IdFlag; }
float AlMuon::SR() { if (muid_alpha == NULL)
                        return -1;
                      else
                        return muid_alpha->SumResid; }
float AlMuon::DM() { if (muid_alpha == NULL)
                        return -1;
                      else
                        return muid_alpha->DistMuch; }
int   AlMuon::ST() { if (muid_alpha == NULL)
                        return -1;
                      else
                        return muid_alpha->ShadTrack; }
int   AlMuon::TN() { if (muid_alpha == NULL)
                        return -1;
                      else
                        return muid_alpha->TrackNumber; }
```

I AlTau

```
///////////////////////////////////////////////////////////
//
// CLASS AlTau
// Concrete class for Tau Objects, inherits from QvecBase
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
////////////////////////////////////////////////////////////


#ifndef _ALTAU_H_
#define _ALTAU_H_

#include <vector>
#include "AlJet.h"
#include "AlTrack.h"


class AlTau: public AlJet {
 public:

  // default constructor
  AlTau();

  // copy constructor
  AlTau(const AlTau& oldAt);

  // copy constructor
  AlTau(const AlJet& oldAt);

  ALEPHTYPE TYPE(){return ALTAU;}

  void  setNch(int nch);

  int   getNch();

  void  setEch(float ech);

  float getEch();

  // set the pointer to the assoc. track
  void setTrack(AlTrack* theTrack);

  // get the pointer to the assoc. track
  AlTrack* getTrack();

 private:
  // number of charged particules
  int _nch;
  // sum of charged tracks energies
  float _ech;
  // the associated track
  AlTrack* _theAssocTrack;
 };

#endif


///////////////////////////////////////////////////////////
//
// Implementation of
// CLASS AlTau
// Concrete class for Tau Objects, inherits from QvecBase
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
////////////////////////////////////////////////////////////

#include "AlTau.h"

// default constructor : has to do something,
// since it is not created out of qvec

AlTau::AlTau()
{
  _theAssocTrack = NULL;
  _ech = 0.;
  _nch = 0;
}

// copy constructor : NOTE: THE VECTOR OF POINTERS
// TO ITS OBJECTS IS JUST COPIED!! COULD BE DANGEROUS...

AlTau::AlTau(const AlTau& oldAt):AlJet(oldAt)
{
  _theAssocTrack = oldAt._theAssocTrack;
  _ech = oldAt._ech;
  _nch = oldAt._nch;
}

AlTau::AlTau(const AlJet& oldAt):AlJet(oldAt)
{
```

```
  _theAssocTrack = NULL;
  _ech = 0.;
  _nch = 0;
}

void AlTau::setNch(int nch)
{ _nch = nch; }
int  AlTau::getNch()
{ return _nch; }

void AlTau::setEch(float ech)
{ _ech = ech; }
float  AlTau::getEch()
{ return _ech; }

// set the assoc. track
void AlTau::setTrack(AlTrack* theTrack)
{
  _theAssocTrack = theTrack;
}

// get the assoc. track
AlTrack* AlTau::getTrack()
{
  return _theAssocTrack;
}
```

J AlTrack

```
/////////////////////////////////////////////////////////
//
// CLASS AlTrack:
// Concrete class for Tracks, inherits from QvecBase
// NOTE: STILL DEPENDS ALSO ON ALPHA BANKS d:-(
//
// Author :  G. Bagliesi
// modified by:  C. Delaere, V. Lemaitre, O. van der Aa :
//                - now inherit form QvecLink
//                - link to new banks: FRFT and TEXS
//                - fixed assignement operator
//
//////////////////////////////////////////////////////////


#ifndef _ALTRACK_H_
#define _ALTRACK_H_

#include "AlphaCopy.h"
#include "QvecLink.h"


class AlTrack : public QvecLink {

public:

  // constructor
  AlTrack();

  // copy constructor
  AlTrack(const AlTrack& oldAt);

  // destructor
  ~AlTrack();

  // get the type
  ALEPHTYPE TYPE(){return TRACK;}

  // set the pointer to ...._ALPHA
  void SetFRTLPointer(FRTL_ALPHA* pointer){frtl_alpha = pointer;}
  void SetFRFTPointer(FRFT_ALPHA* pointer){frft_alpha = pointer;}
  void SetTEXSPointer(TEXS_ALPHA* pointer){texs_alpha = pointer;}


  // copy of ALPHA functions for FRTL
  int   NV();
  int   NI();
  int   NE();
  int   NT();
  int   NR();
  // copy of ALPHA functions for FRFT
  float IR();
  float TL();
  float P0();
  float D0();
  float Z0();
  float AL();
  float EM(int x, int y);
  float C2();
  int   DF();
  int   NO();
  // copy of ALPHA functions for TEXS
  int   SI();
  float TM();
  float TL2();
  int   NS();
  float AD();
  int   TN();
  int   SF();

private:

  FRTL_ALPHA* frtl_alpha;   // pointer to the ALPHA fortran memory
  FRFT_ALPHA* frft_alpha;   // pointer to the ALPHA fortran memory
  TEXS_ALPHA* texs_alpha;   // pointer to the ALPHA fortran memory

};

#endif


/////////////////////////////////////////////////////////
//
// Implementation of
// CLASS AlTrack:
// Concrete class for Tracks, inherits from QvecBase
// NOTE: STILL DEPENDS ALSO ON ALPHA BANKS d:-(
//
// Author :  G. Bagliesi
// modified by:  C. Delaere, V. Lemaitre, O. van der Aa :
//                - now inherit form QvecLink
//                - link to new banks: FRFT and TEXS
```

```
//                  - fixed assignement operator
//
////////////////////////////////////////////////////////////

#include "AlTrack.h"

// NOTE: IF A NEW TRACK IS CREATED OUTSIDE THE CONTEXT
// OF ALPHABANKS, IT WILL NOT KNOW ITS FRTL_ALPHA POINTER,
// SO THE FUNCTIONS BELOW WILL NOT WORK
//
// NEW NOTE:
// CHANGES MADE TO COPY CONSTRUCTOR AND ASSIGNMENTOPERATOR
// SHOULD MAKE IT POSSIBLE TO CREATE NEW TRACKS OUTSIDE THE
// ALPHABANKS.

// default constructor :
AlTrack::AlTrack()
{
   frtl_alpha = NULL;
   frft_alpha = NULL;
   texs_alpha = NULL;
}

// NOTE: IF A NEW TRACK IS CREATED OUTSIDE THE CONTEXT
// OF ALPHABANKS VIA A COPY, THE POINTER TO FRTL COULD BE OBSOLETE...

// copy constructor :
AlTrack::AlTrack(const AlTrack& oldAt):QvecLink(oldAt)
{
  frtl_alpha = oldAt.frtl_alpha;
  frft_alpha = oldAt.frft_alpha;
  texs_alpha = oldAt.texs_alpha;
}

// destructor
AlTrack::~AlTrack(){}

int   AlTrack::NV() { if (frtl_alpha == NULL)
                        return -1;
                      else
                        return frtl_alpha->NarcV; }
int   AlTrack::NI() { if (frtl_alpha == NULL)
                        return -1;
                      else
                        return frtl_alpha->NarcI; }
int   AlTrack::NT() { if (frtl_alpha == NULL)
                        return -1;
                      else
                        return frtl_alpha->NarcT; }
int   AlTrack::NE() { if (frtl_alpha == NULL)
                        return -1;
                      else
                        return frtl_alpha->NrEsti; }
int   AlTrack::NR() { if (frtl_alpha == NULL)
                        return -1;
                      else
                        return frtl_alpha->NrEstt; }

float AlTrack::IR() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->InverseRadi; }
float AlTrack::TL() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->TanLambda; }
float AlTrack::P0() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->Phi0; }
float AlTrack::D0() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->D0; }

float AlTrack::Z0() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->Z0; }
float AlTrack::AL() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->Alpha; }
float AlTrack::EM(int x, int y) { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->EcovarM[(x*(x-1))/2+y-1];}
float AlTrack::C2() { if (frft_alpha == NULL)
                        return -1;
                      else
                        return frft_alpha->Chis2; }
int   AlTrack::DF() { if (frft_alpha == NULL)
                        return -1;
```

```
                     else
                       return frft_alpha->numDegFree; }
int   AlTrack::NO() { if (frft_alpha == NULL)
                       return -1;
                     else
                       return frft_alpha->nopt; }

int   AlTrack::SI() { if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->SegmentId; }
float AlTrack::TM() { if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->TruncatedMean; }
float AlTrack::TL2(){ if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->Tracklength; }
int   AlTrack::NS() { if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->NumberSamples; }
float AlTrack::AD() { if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->AverageDrift; }
int   AlTrack::TN() { if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->TrackNumber; }
int   AlTrack::SF() { if (texs_alpha == NULL)
                       return -1;
                     else
                       return texs_alpha->SatFlag; }
```

K AlephAlgoLeptons

```
/////////////////////////////////////////////////////////
//
// Aleph Algos for Leptons:
// Leptons filtering and Tau Production
//
// Author : C. Delaere, V. Lemaitre, O. van der Aa
//
//////////////////////////////////////////////////////////

#include <map>

// Muons filtering

template <class Type>
void AlephVector<Type>::FilterMu( float d0max,float pmin,int minhit,
  float cosmax,float z0max)
{
  iterator imu;
  for (imu=begin();imu<end();imu++)
    // ignore locked objects
    if (!((*imu)->isLocked()))
      {
// check if the type is correct...
if ((*imu)->TYPE() != MUON )
  {
    cout << "WARNING Not AlMuon: type not supported !" << endl;
    exit(1);
  }
// good tracks selection
bool xtrackdp=true;
if ((*imu)->getTrack()->isLocked())            xtrackdp=false;
if ((*imu)->QP() < pmin)                       xtrackdp=false;
if ((*imu)->getTrack()->NT() < minhit)         xtrackdp=false;
if (fabs((*imu)->QCT()) >= cosmax)             xtrackdp=false;
if (fabs((*imu)->getTrack()->QZB()) >= z0max)  xtrackdp=false;
if (fabs((*imu)->getTrack()->QDB()) >= d0max)  xtrackdp=false;
// final selection
if ((!xtrackdp) || ((*imu)->IF() <=  0)) (*this).looperase(imu);
      }
}

// Electrons filtering

template <class Type>
void AlephVector<Type>::FilterEl( const AlephVector<AlMuon *>& almup,
  float d0max,float pmin,int minhit,
  float cosmax,float z0max,
  int nwirem,float r3cut,float r2cut0l,
  float r2cut1l,float r2cut2l,float r2cuth,
  float r5cutl,float r5cuth)
{
  // We've to check that the electron candidate is NOT a muon
  // to loop only once, we first fill a map
  // with all the needed information
  map<int,int> tr;
  for (unsigned int i=0; i < almup.size(); i++)
    if (!((almup[i])->isLocked()))
      tr[(almup[i]->TN())]=i;
  iterator iel;
  for (iel=begin();iel<end();iel++)
    // ignore locked objects
    if (!((*iel)->isLocked()))
      {
// check the type
if ((*iel)->TYPE() != ELEC)
  {
    cout << "WARNING Not AlElec: type not supported !" << endl;
    exit(1);
  }
// check track quality
bool xtrackdp=true;
if ((*iel)->getTrack()->isLocked())                xtrackdp=false;
if ((*iel)->QP() < pmin)                           xtrackdp=false;
if ((*iel)->getTrack()->NT() < minhit)             xtrackdp=false;
if (fabs((*iel)->QCT()) >= cosmax)                 xtrackdp=false;
if (fabs((*iel)->getTrack()->QZB()) >= z0max)      xtrackdp=false;
if (fabs((*iel)->getTrack()->QDB()) >= d0max)      xtrackdp=false;
// look in the map if it is a muon or not...
if (tr.find((*iel)->FR())!=tr.end()) xtrackdp=false;
// final selection
if (xtrackdp)
  {
    switch ((*iel)->IF2())
      {
      case -1 :                 // no EIDT information available
xtrackdp=false;
break;
      case 0  :                 // o.k.
if (((*iel)->R3() >= r3cut)    ||
    ((*iel)->R2() <= r2cut0l)  ||
    ((*iel)->R2() >= r2cuth)     )     xtrackdp=false;
```

```
break;
      case 1  :                 // in a crack
if ((((*iel)->R3() >= r3cut)   ||
     ((*iel)->R2() <= r2cut1l) ||
     ((*iel)->R2() >= r2cuth))
    &&
    (((*iel)->R5() >= r5cuth)  ||
     ((*iel)->R5() <= r5cutl)  ||
     ((*iel)->getTrack()->NS() < nwirem)) )  xtrackdp=false;
break;
      case 2  :               // in overlap
if ((((*iel)->R3() >= r3cut)   ||
     ((*iel)->R2() <= r2cut2l) ||
     ((*iel)->R2() >= r2cuth))
    &&
    (((*iel)->R5() >= r5cuth)  ||
     ((*iel)->R5() <= r5cutl)  ||
     ((*iel)->getTrack()->NS() < nwirem)) )  xtrackdp=false;
break;
      }
  }
if (!xtrackdp) (*this).looperase(iel);
      }
}

// TauJet clustering
// Produces a raw set of Tau candidates...
// I mean Taujets with one or three Eflws
// at this stage, Ech is the total energy of eflws objects
//               the Associated track is the track from the last Eflw.
// be carefull: this track may be NULL

template <class Type>
AlephVector<AlTau> AlephVector<Type>::ATauJn(float Ycut, int Scheme,
     int Metric, float Qelep)
{
  // Check the type of object. MUST be Eflw
  for (iterator i=begin();i<end();i++)
    if ((*i)->TYPE() != EFLOW)
      {
cout << "WARNING Not Eflw: type not supported !" << endl;
exit(1);
      }
  // make a lot of mini-jets with JADE algorithm (preferably)
  // we don't have to care about the lock problem 'cause the JetAlgo does it
  AlephVector<AlJet> Jjade;
  if (Metric)
    Jjade = (*this).JadeJet(Ycut, Scheme, Qelep);
  else
    Jjade = (*this).DurhamJet(Ycut, Scheme, Qelep);
  AlephVector<AlTau> P;
  int jetn = Jjade.size();
  // loop on the candidates
  for (int ijet = 0; ijet<jetn; ijet++)
    {
      typedef AlephVector<AlObject*>::iterator ObI;
      AlTrack* theAssTrack = NULL;
      float Echgood=0.;
      int tn=0;
      // loop on objects in the taujet, record #tracks and total E
      for (ObI ifl=Jjade[ijet].getObjects().begin();
   ifl<Jjade[ijet].getObjects().end();
   ifl++)
if (((AlEflw*)(*ifl))->getEfType()<=3)
  {
    if (((AlEflw*)(*ifl))->getTrack() != NULL )
      theAssTrack = ((AlEflw*)(*ifl))->getTrack();
    tn++;
    Echgood+=(((AlEflw*)(*ifl))->QE());
  }
      // check number of objects and save the result
      if ((tn == 1) || (tn == 3))
{
  AlTau tmp = Jjade[ijet];
  tmp.setNch(tn);
  tmp.setEch(Echgood);
  tmp.setTrack(theAssTrack);
  P.push_back(tmp);
}
      }
  return P;
}

// Tau filtering
// analyse a vector of pointers to AlTau objects
// apply standard cuts
// Ech is the energy of charged tracks
// The associated track is the most energetic one, with the same sign

template <class Type>
void AlephVector<Type>::FilterTau(float psummin, float d0max, int minhit,
  float cosmax, float z0max)
{
```

```
  for (iterator ijet = begin(); ijet<end(); ijet++)
    {
      int Ntpc;
      float pp;
      float ee;
      float d0;
      float z0;
      float costheta;
      int Nchgood=0;
      int OneBadTrack=0;
      int lock=0;
      typedef AlephVector<AlObject*>::iterator ObI;
      AlTrack* theAssTrack =(*ijet)->getTrack() ;
      // loop on objects in the jet
      // compute the associated track and get some parameters
      for (ObI ifl=(*ijet)->getObjects().begin();
   ifl<(*ijet)->getObjects().end();
   ifl++)
{
  if (((AlEflw*)(*ifl))->getEfType()<=3)
    {
      AlTrack* theTrack = ((AlEflw*)(*ifl))->getTrack();
      if (theTrack==NULL)
{
  Ntpc = 0;
  d0   = 0;
  z0   = 0;
  lock = 1;
  costheta = 1;
}
      else
{
  Ntpc = theTrack->NT();
  if ((((*ijet)->QCH() * theTrack->QCH() > 0 ) &&
       ( theTrack->QE()>theAssTrack->QE()    )     ) ||
      (( (*ijet)->QCH() == 0 ) && ( theTrack->QE()>theAssTrack->QE())))
    theAssTrack=theTrack;
  d0 = ((AlEflw*)(*ifl)) ->getTrack() -> QDB();
  z0 = ((AlEflw*)(*ifl)) ->getTrack() -> QZB();
  lock = ((AlEflw*)(*ifl)) ->getTrack() -> isLocked();
  costheta = ((AlEflw*)(*ifl)) -> QCT();
}
      // bad tracks rejection
      if ( (fabs(d0)>=d0max)        ||
   (fabs(z0)>=z0max)        ||
   (fabs(costheta)>=cosmax) ||
   (Ntpc<minhit)            ||
   (lock==1)                   )
OneBadTrack++;
    }
}
      // final selection
      if(OneBadTrack>0 || (((*ijet)->getEch()) <= psummin))
looperase(ijet);
      else
(*ijet)->setTrack(theAssTrack);
    }
}
```

L Durham jet

```
#include "DurhamProtoJet.h"

// From Orca
// Written by J.P. Wellisch, 25. Jan 1999
// Adapted to Alpha++ by C. Delaere
//              Added : - DurhamYn
//                      - fixed number of jets

template <class Type>
AlephVector<AlJet> AlephVector<Type>::DurhamJet(float theYCut, int Scheme, float Qelep)
  /*
    Routine to calculate number of jet and jet momenta for a given Ycut and
    a given AlephVector of pointers to AlObjects
    Input:   float theYcut             Ycut value
             int Scheme                Recombination Scheme
    Scheme = 0 : E0 - Scheme
    Scheme = 1 : E - Scheme
    Enter a negative valut of theYCut to obtain -theYCut jets

    Output: AlephVector<AlJet> jets    Vector of Jets
  */
{
  // update the static member of DurhamProtoJet: used to add 2 protojets
  DurhamProtoJet::Scheme = Scheme;
  // the result...
  AlephVector<AlJet> result;
  // the temp vector to work
  vector<DurhamProtoJet> theProtoJets;
  iterator i = begin();
  double totalEnergy = 0.;
  // make a copy of the original vector (as protojets) and compute Evis
  for (i = begin(); i != end(); i++)
    if (!((*i)->isLocked()))
      {
totalEnergy += (*i)->A4V().e() ;
DurhamProtoJet aProtoJet(*(*i));
theProtoJets.push_back(aProtoJet);
      }
  if(Qelep != 0.) totalEnergy=Qelep;
  int Np = theProtoJets.size();
  double totalE2 = totalEnergy*totalEnergy;

  // Check if # of particles is o.k.
  if (Np < 2)
    {
      result.resize(0);
      fout <<
"there is only one particle, therefore Jet Algorithm cannot be performed"
   << endl;
      return result;
    }

  double minY = 0;
  int iYcut = (int) theYCut;
  // Main loop
  while(((minY<theYCut) && (theYCut >= 0.)) ||
(((int)theProtoJets.size() != -iYcut) && (theYCut < 0.)))
  {
    minY = DBL_MAX;
    vector<DurhamProtoJet>::iterator aCandidate = 0;
    vector<DurhamProtoJet>::iterator aNextNeighbour = 0;
    vector<DurhamProtoJet>::iterator i1 = theProtoJets.begin();
    for(i1 = theProtoJets.begin(); i1!= theProtoJets.end(); i1++)
      {
double y = DBL_MAX;
vector<DurhamProtoJet>::iterator aNeighbour = i1->findNextNeighbour(theProtoJets);
if(aNeighbour != 0) y = i1->getYTimesE2(aNeighbour)/totalE2;
if(y<minY)
  {
    minY = y;
    aCandidate = i1;
    aNextNeighbour = aNeighbour;
  }
      }
    if(((minY<theYCut) && (theYCut >= 0.)) ||
       (((int)theProtoJets.size() != -iYcut) && (theYCut < 0.)))
      {
*aCandidate += *aNextNeighbour;
theProtoJets.erase(aNextNeighbour);
      }
  }

  // build the result from the protojets
  // (uses the conversion constructor of AlJet)
  vector<DurhamProtoJet>::iterator aJetList;
  for(aJetList = theProtoJets.begin();
      aJetList != theProtoJets.end();
      aJetList++)
    result.push_back(*aJetList);
  // return an AlephVector of AlJet
```

```
  return result;
}


template <class Type>
vector<float> AlephVector<Type>::DurhamYn(int Scheme, float Qelep)
  /*
    Routine to calculate number of jet and jet momenta for a given Ycut and
    a given AlephVector of pointers to AlObjects
    Input:   int Scheme               Recombination Scheme
                         Scheme = 0 : E0 - Scheme
                         Scheme = 1 : E - Scheme

    Output: Vector<Real> Yn            Vector of critical Ycuts
  */
{
  // update the static member of DurhamProtoJet: used to add 2 protojets
  DurhamProtoJet::Scheme = Scheme;
  // the temp vector to work
  vector<DurhamProtoJet> theProtoJets;
  iterator i = begin();
  double totalEnergy = 0.;
  // make a copy of the original vector (as protojets) and compute Evis
  for (i = begin(); i != end(); i++)
    if (!((*i)->isLocked()))
      {
totalEnergy += (*i)->A4V().e() ;
DurhamProtoJet aProtoJet(*(*i));
theProtoJets.push_back(aProtoJet);
      }
  if(Qelep != 0.) totalEnergy=Qelep;
  int Np = theProtoJets.size();
  // the result...
  vector<float> Yn(Np-1);
  double totalE2 = totalEnergy*totalEnergy;

  // Check if # of particles is o.k.
  if (Np < 2)
    {
      Yn.resize(0);
      fout << "there is only one particle, therefore AJetJn cannot be performed" << endl;
      return Yn;
    }

  double minY = 0;
  // Main loop
  while( theProtoJets.size() != 1)
  {
    minY = DBL_MAX;
    vector<DurhamProtoJet>::iterator aCandidate = 0;
    vector<DurhamProtoJet>::iterator aNextNeighbour = 0;
    vector<DurhamProtoJet>::iterator i1 = theProtoJets.begin();
    for(i1 = theProtoJets.begin(); i1!= theProtoJets.end(); i1++)
      {
double y = DBL_MAX;
vector<DurhamProtoJet>::iterator aNeighbour = i1->findNextNeighbour(theProtoJets);
if(aNeighbour != 0) y = i1->getYTimesE2(aNeighbour)/totalE2;
if(y<minY)
  {
    minY = y;
    aCandidate = i1;
    aNextNeighbour = aNeighbour;
  }
      }
*aCandidate += *aNextNeighbour;
theProtoJets.erase(aNextNeighbour);
Yn[theProtoJets.size()-1]=minY;
  }
  // return a vector of float
  return Yn;
}


#ifndef DurhamProtoJet_h
#define DurhamProtoJet_h

//////////////////////////////////////////////////////////////
//
// CLASS : DurhamProtoJet
// Object used to compute jets with Durham
//
// Written by J.P. Wellisch, 25. Jan 1999
// adapted to Alpha++ by C.Delaere
//
///////////////////////////////////////////////////////////////

#include <math.h>
#include <vector>
#include "QvecBase.h"

class DurhamProtoJet
{
  public:
```

```
  // Constructor (from QvecBase)
  DurhamProtoJet(const QvecBase & alobj);
  // Operator += (depends on the scheme)
  DurhamProtoJet & operator+=(const DurhamProtoJet & aJet);
  // the Metric
  double getYTimesE2(const DurhamProtoJet * aJet);
  // find the Next Neighbour
  vector<DurhamProtoJet>::iterator findNextNeighbour(const vector<DurhamProtoJet> & theProtoJets);
  // get constituants
  const vector<const QvecBase *> getConstituants() const {return theConstituants;}
  // the scheme (set by the algo, used by operator+=)
  static int Scheme;
  // get four momentum vector
  const HepLorentzVector & get4Momentum() const { return theFourMomentum;}

  private:
  HepLorentzVector theFourMomentum;
  vector<const QvecBase *> theConstituants;

};

#endif


/////////////////////////////////////////////////////////
//
// Implementation of
// CLASS : DurhamProtoJet
// Object used to compute jets with Durham
//
// Written by J.P. Wellisch, 25. Jan 1999
// adapted to Alpha++ by C.Delaere
//
//////////////////////////////////////////////////////////

#include "DurhamProtoJet.h"
#include "AlObject.h"

// Constructor (from QvecBase)
DurhamProtoJet::
DurhamProtoJet(const QvecBase & alobj)
{
  if (alobj.A4V().e()==0)
    {
      cout << "Error: bad Eflw: E=0" << endl;
      exit(1);
    }
  theFourMomentum = alobj.A4V();
  theConstituants.push_back(&alobj);
}

// Operator += (depends on the scheme)
DurhamProtoJet & DurhamProtoJet::
operator+=(const DurhamProtoJet & aJet)
{
  if (Scheme)
    {
      // recombination according to E-Scheme.
      theFourMomentum += aJet.get4Momentum();
    }
  else
    {
      // recombination according to E0-Scheme.
      double e = theFourMomentum.e() + aJet.get4Momentum().e();
      double p2i = Hep3Vector(theFourMomentum + aJet.get4Momentum()).mag();
      if (p2i <= pow(10,-14))
{
  cout << " Error: Division by zero! " << endl;
  exit(1);
}
      theFourMomentum.setVect(e/p2i*(theFourMomentum.vect()+aJet.get4Momentum().vect()));
      theFourMomentum.setE(e);
    }

  vector<const QvecBase *> aConstituantList;
  aConstituantList = aJet.getConstituants();
  vector<const QvecBase *>::iterator constitIter;
  for(constitIter=aConstituantList.begin();
      constitIter!=aConstituantList.end();
      constitIter++)
  {
    theConstituants.push_back(*constitIter);
  }

  return *this;
}

// the Metric
double DurhamProtoJet::
getYTimesE2(const DurhamProtoJet * aJet)
{
  double result;
  double e1 = theFourMomentum.e();
```

```
  double e2 = aJet->get4Momentum().e();
  double costh = theFourMomentum.vect() * aJet->get4Momentum().vect();
  costh /= theFourMomentum.vect().mag();
  costh /= aJet->get4Momentum().vect().mag();
  result = e1;
  if(e1>e2) result = e2;
  result *= result;
  result *= 2.*(1-costh);
  return result;
}

// find the Next Neighbour
vector<DurhamProtoJet>::iterator DurhamProtoJet::
findNextNeighbour(const vector<DurhamProtoJet> & theProtoJets)
{
  vector<DurhamProtoJet>::iterator i = theProtoJets.begin();
  double minDist = DBL_MAX;
  vector<DurhamProtoJet>::iterator theNextNeighbour = 0;
  for (i = theProtoJets.begin(); i != theProtoJets.end(); i++)
  {
    if(i!=this)
    {
      double distance = getYTimesE2(i);
      if(distance<minDist)
      {
        minDist = distance;
        theNextNeighbour = i;
      }
    }
  }
  return theNextNeighbour;
}

// the scheme (set by the algo, used by operator+=)
int DurhamProtoJet::Scheme=1;
```

M Jade Jet

```
#include "DurhamProtoJet.h"

// From Orca
// Written by J.P. Wellisch, 25. Jan 1999
// Adapted to Alpha++ by C. Delaere
//              Added :  fixed number of jets

template <class Type>
AlephVector<AlJet> AlephVector<Type>::JadeJet(float theYCut, int Scheme,
      float Qelep)
  /*
    Routine to calculate number of jet and jet momenta for a given Ycut and
    a given AlephVector of pointers to AlObjects
    Input:   float theYcut             Ycut value
             int Scheme                Recombination Scheme
    Scheme = 0 : E0 - Scheme
    Scheme = 1 : E - Scheme
    Enter a negative valut of theYCut to obtain -theYCut jets

    Output: AlephVector<AlJet> jets    Vector of Jets
  */
{
  // update the static member of JadeProtoJet: used to add 2 protojets
  JadeProtoJet::Scheme = Scheme;
  // the result...
  AlephVector<AlJet> result;
  // the temp vector to work
  vector<JadeProtoJet> theProtoJets;
  iterator i = begin();
  double totalEnergy = 0.;
  // make a copy of the original vector (as protojets) and compute Evis
  for (i = begin(); i != end(); i++)
    if (!((*i)->isLocked()))
      {
totalEnergy += (*i)->A4V().e() ;
JadeProtoJet aProtoJet(*(*i));
theProtoJets.push_back(aProtoJet);
      }
  if(Qelep != 0.) totalEnergy=Qelep;
  int Np = theProtoJets.size();
  double totalE2 = totalEnergy*totalEnergy;

  // Check if # of particles is o.k.
  if (Np < 2)
    {
      result.resize(0);
      fout <<
"there is only one particle, therefore AJetJn cannot be performed"
   << endl;
      return result;
    }

  double minY = 0;
  int iYcut = (int) theYCut;
  // Main loop
  while(((minY<theYCut) && (theYCut >= 0.)) ||
(((int)theProtoJets.size() != -iYcut) && (theYCut < 0.)))
  {
    minY = DBL_MAX;
    vector<JadeProtoJet>::iterator aCandidate = 0;
    vector<JadeProtoJet>::iterator aNextNeighbour = 0;
    vector<JadeProtoJet>::iterator i1 = theProtoJets.begin();
    for(i1 = theProtoJets.begin(); i1!= theProtoJets.end(); i1++)
      {
double y = DBL_MAX;
vector<JadeProtoJet>::iterator aNeighbour = i1->findNextNeighbour(theProtoJets);
if(aNeighbour != 0) y = i1->getYTimesE2(aNeighbour)/totalE2;
if(y<minY)
  {
    minY = y;
    aCandidate = i1;
    aNextNeighbour = aNeighbour;
  }
      }
    if(((minY<theYCut) && (theYCut >= 0.)) ||
       (((int)theProtoJets.size() != -iYcut) && (theYCut < 0.)))
      {
*aCandidate += *aNextNeighbour;
theProtoJets.erase(aNextNeighbour);
      }
  }

  // build the result from the protojets
  // (uses the conversion constructor of AlJet)
  vector<JadeProtoJet>::iterator aJetList;
  for(aJetList = theProtoJets.begin();
      aJetList != theProtoJets.end();
      aJetList++)
    result.push_back(*aJetList);
  // return an AlephVector of AlJet
```

```
  return result;
}


#ifndef JadeProtoJet_h
#define JadeProtoJet_h

///////////////////////////////////////////////////////////
//
// CLASS : JadeProtoJet
// Object used to compute jets with Jade
//
// Written by J.P. Wellisch, 25. Jan 1999
// adapted to Alpha++ by C.Delaere
//
////////////////////////////////////////////////////////////

#include <math.h>
#include <vector>
#include "QvecBase.h"

class JadeProtoJet
{
  public:
  // Constructor (from QvecBase)
  JadeProtoJet(const QvecBase & alobj);
  // Operator += (depends on the scheme)
  JadeProtoJet & operator+=(const JadeProtoJet & aJet);
  // the Metric
  double getYTimesE2(const JadeProtoJet * aJet);
  // find the Next Neighbour
  vector<JadeProtoJet>::iterator findNextNeighbour(const vector<JadeProtoJet> & theProtoJets);
  // get constituants
  const vector<const QvecBase *> getConstituants() const {return theConstituants;}
  // the scheme (set by the algo, used by operator+=)
  static int Scheme;
  // get four momentum vector
  const HepLorentzVector & get4Momentum() const { return theFourMomentum;}

  private:
  HepLorentzVector theFourMomentum;
  vector<const QvecBase *> theConstituants;

};

#endif



///////////////////////////////////////////////////////////
//
// Implementation of
// CLASS : JadeProtoJet
// Object used to compute jets with Jade
//
// Written by J.P. Wellisch, 25. Jan 1999
// adapted to Alpha++ by C.Delaere
//
////////////////////////////////////////////////////////////

#include "JadeProtoJet.h"
#include "AlObject.h"

// Constructor (from QvecBase)
JadeProtoJet::
JadeProtoJet(const QvecBase & alobj)
{
  if (alobj.A4V().e()==0)
    {
      cout << "Error: bad Eflw: E=0" << endl;
      exit(1);
    }
  theFourMomentum = alobj.A4V();
  theConstituants.push_back(&alobj);
}

// Operator += (depends on the scheme)
JadeProtoJet & JadeProtoJet::
operator+=(const JadeProtoJet & aJet)
{
  if (Scheme)
    {
      // recombination according to E-Scheme.
      theFourMomentum += aJet.get4Momentum();
    }
  else
    {
      // recombination according to E0-Scheme.
      double e = theFourMomentum.e() + aJet.get4Momentum().e();
      double p2i = Hep3Vector(theFourMomentum + aJet.get4Momentum()).mag();
      if (p2i <= pow(10,-14))
{
  cout << " Error: Division by zero! " << endl;
  exit(1);
```

```
}
      theFourMomentum.setVect(e/p2i*(theFourMomentum.vect()+aJet.get4Momentum().vect()));
      theFourMomentum.setE(e);
    }

  vector<const QvecBase *> aConstituantList;
  aConstituantList = aJet.getConstituants();
  vector<const QvecBase *>::iterator constitIter;
  for(constitIter=aConstituantList.begin();
      constitIter!=aConstituantList.end();
      constitIter++)
  {
    theConstituants.push_back(*constitIter);
  }

  return *this;
}

// the Metric
double JadeProtoJet::
getYTimesE2(const JadeProtoJet * aJet)
{
  double result;
  double e1 = theFourMomentum.e();
  double e2 = aJet->get4Momentum().e();
  double costh = theFourMomentum.vect() * aJet->get4Momentum().vect();
  costh /= theFourMomentum.vect().mag();
  costh /= aJet->get4Momentum().vect().mag();
  result = 2.*e1*e2*(1-costh);
  return result;
}

// find the Next Neighbour
vector<JadeProtoJet>::iterator JadeProtoJet::
findNextNeighbour(const vector<JadeProtoJet> & theProtoJets)
{
  vector<JadeProtoJet>::iterator i = theProtoJets.begin();
  double minDist = DBL_MAX;
  vector<JadeProtoJet>::iterator theNextNeighbour = 0;
  for (i = theProtoJets.begin(); i != theProtoJets.end(); i++)
  {
    if(i!=this)
    {
      double distance = getYTimesE2(i);
      if(distance<minDist)
      {
        minDist = distance;
        theNextNeighbour = i;
      }
    }
  }
  return theNextNeighbour;
}

// the scheme (set by the algo, used by operator+=)
int JadeProtoJet::Scheme=1;
```