```
#ifndef CalCluster_H
#define CalCluster_H

#include <vector>
#include "geometry/Point.h"
#include "geometry/Vector.h"
#include "GaudiKernel/DataObject.h"
#include "GaudiKernel/MsgStream.h"

extern const CLID& CLID_CalClusterCol;



namespace Event
{

/**
*  @class CalCluster
*
*
*  This class stores the results of the reconstruction performed
*  in CalClustersAlg. It contains the reconstructed data for one
*  cluster in the calorimeter.
*
*  \author Alexandre Chekhtman
*  \author Regis Terrier
*  \author Jose Angel Hernando
*
* $Header$
*/


   class CalCluster
   {

   public:

      /// constructor
      /// @param e total energy sum in the cluster
      /// @param p reconstructed cluster position
      CalCluster(double e, Point p);

      ~CalCluster() {}

      /**
```

```
*   initializing a subset
*   of CalCluster data members
*
*   @param eleak energy corrected by last layer correlation method
*   @param eneLayer vector of energy depositions for each layer
*   @param pLayer   average position for each layer
*   @param rmsLayer quadratic position spread for each layer
*   @param rms_long RMS of longitudinal position measurements
*   @param rms_long RMS of transversal position measurements
*   @param caldir particle direction reconstructed from calorimeter
*               data
*   @param calTransvOffset transverse offset of position measured
*                       by calorimeter with respect
*                       to prediction from tracker data
*/
void initialize(double eleak,
   std::vector<double> eneLayer,
   std::vector<Vector> pLayer,
   std::vector<Vector> rmsLayer,
   double rms_long,
   double rms_trans,
   Vector caldir,
   double calTransvOffset)
{
   m_eneLayer = eneLayer;
   m_leakEnergy = eleak;
   m_pLayer = pLayer;
   m_rmsLayer = rmsLayer;
   m_rmslong = rms_long;
   m_rmstrans= rms_trans;
   m_direction = caldir;
   m_transvOffset = calTransvOffset;
}

/**
*   storing the profile fitting results
*
*  @param fit_energy energy obtained by profile fitting
*  @param ki2        chi squared of the fit
*  @param fit_start  position of the shower start
*  @param fit_alpha  alpha parameter - position of the shower maximum
*  @param fit_lambda lambda parameter, describing the shower back tail
*
*/

void initProfile(double fit_energy, double ki2, double fit_start,
```

```
   double fit_alpha, double fit_lambda)
{
   m_fitEnergy = fit_energy;
   m_ProfChisq = ki2;
   m_CsiAlpha = fit_alpha;
   m_CsiLambda = fit_lambda;
   m_start  = fit_start;
}



/// get energy sum
double getEnergySum()       const {return m_energySum;}

/// get energy corrected for leakage with last layer correlation
double getEnergyLeak()      const {return m_leakEnergy;}

/// get energy corrrected by the best chosen method
double getEnergyCorrected()  const {return m_energyCorrected;}

/// get energy for the layer with given number
double getEneLayer(int i) const {return m_eneLayer[i];}

/// get position for the layer with given number
const Vector& getPosLayer(int i) const {return m_pLayer[i];}

/// get vector of energies for all layers
const std::vector<double>& getEneLayer() const {return m_eneLayer;}

/// get vector of positions for all layers
const std::vector<Vector>& getPosLayer() const {return m_pLayer;}

/// get vector of quadratic spreads for all layers
const std::vector<Vector>& getRmsLayer() const {return m_rmsLayer;}

/// get RMS of longitudinal position measurements
double getRmsLong()                 const {return m_rmslong;}

/// get RMS of transverse position measurements
double getRmsTrans()      const {return m_rmstrans;}

/// get transversal offset of cluster position with respect to
/// prediction from tracker data
double getTransvOffset()  const {return m_transvOffset;}

/// get reconstructed position
```

```
    Point getPosition()         const {return m_position;}

    /// get reconstructed direction
    Vector getDirection()       const {return m_direction;}

    /// get energy obtained by profile fitting
    double getFitEnergy()       const {return m_fitEnergy;}

    /// get chi square of profile fitting
    double getProfChisq()       const {return m_ProfChisq;}

    /// get alpha parameter of profile fit
    double getCsiAlpha()        const {return m_CsiAlpha;}

    /// get lambda parameter of profile fit
    double getCsiLambda()       const {return m_CsiLambda;}

    /// get position of shower start
    double getCsiStart()        const {return m_start;}

    /// write some of CalCluster data to the ASCII output file
    /// for debugging purposes
    void writeOut(MsgStream& stream) const;
    friend std::ostream& operator << (std::ostream& s, const CalCluster& obj)
    {
        return obj.fillStream(s);
    };

    /// Fill the ASCII output stream
    virtual std::ostream& fillStream( std::ostream& s ) const;

  protected:

    ///reset CalCluster data
    virtual void ini();

  private:

    //! Total measured energy in the calorimeter
    double m_energySum;

    /// Energy corrected for leakage using correlation method
    /// ( for E> several GeV)
    double m_leakEnergy;

    //! corrected energy not used ( yet )
```

```
    double m_energyCorrected;

    //! Energy per layer in MeV
    std::vector<double> m_eneLayer;

    //! Barycenter position in each layer
    std::vector<Vector> m_pLayer;

    //! RMS of energy deposition in each layer
    std::vector<Vector> m_rmsLayer;

    //! RMS of longitudinal position measurement
    double m_rmslong;

    //! RMS of transverse position measurement
    double m_rmstrans;

    //! Transvers offset of calorimeter position measurement
    double m_transvOffset;

    //! fitted energy ( for E>10 GeV)
    double m_fitEnergy;

    //! Chisquare of the fit ( not a real Chisquare)
    double m_PprofChisq;

    //! Alpha parameter used in the fit
    double m_cCsiAlpha;

    //! Lambda parameter used in the fit
    double m_cCsiLambda;

    //! Fitted starting point of the shower (physical meaning is not clear)
    double m_start;

    /// reconstructed position
    Point m_position;

    /// reconstructed direction
    Vector m_direction;
};


/*!
* @class CalClusterCol
*
```

```
*  TDS class  to store the results of the reconstruction performed
*  in CalClustersAlg. It inherits from DataObject
* (to be a part of Gaudi TDS) and from std::vector of pointers
* to CalCluster objects. Some methods just rename the std::vector
* methods with the same functionality for backward compartibility.
*
*
* @warning there is no clustering in the calorimeter up to now. There is
*  just one CalCluster stored, containing the information for the whole
*  event
*
* @author Jose Angel Hernando
* @author Alexandre Chekhtman
*
* @todo replace this class by typedef to ObjectVector, make corresponding
*       changes in CalClustersAlg
*/

class CalClusterCol : public DataObject, public std::vector<CalCluster*>
{
public:

   CalClusterCol() { clear();}

   /// destructor - deleting the clusters pointed
   /// by the vector elements
   ~CalClusterCol() { delClusters();}


   // GAUDI members to be use by the converters
   static const CLID& classID() {return CLID_CalClusterCol;}
   virtual const CLID& clID() const {return classID();}

   /// add new cluster
   void add(CalCluster* cl) {push_back(cl);}

   /// get the number of clusters in collection
   int num()                const {return size();}

   /// get pointer to the cluster with given number
   CalCluster* getCluster(int i) const {return operator[](i);}

   /// delete all clusters pointed by the vector elements
   void delClusters();

   /// write information for all clusters to the ascii file
```

```
        friend std::ostream& operator << (std::ostream& s, const AcdRecon& obj)
        {
          return obj.fillStream(s);
        };

        /// Fill the ASCII output stream
        virtual std::ostream& fillStream( std::ostream& s ) const;      /// for debugging purposes
        virtual void writeOut(MsgStream& stream) const;

    protected:

        /// does the same function as clear()
        virtual void ini();

    };

}

#endif
```