```
                }

                m_ServeLongCentroids[i] = m_LongCentroids[i];
                m_ServeLatCentroids[i] = m_LatCentroids[i];
            }
        }

        if (Method == "OfAllPts") {
            double LongSum, LatSum;
            for (i = 0; i < nClus; i++) {
                LongSum = 0.0;
                LatSum = 0.0;
                N = m_ClusMembers[i].GetSize();

                for (j = 0; j < N; j++) {
                    LongSum += m_pFinal->GetLong(m_ClusMembers[i].GetAt(j));
                    LatSum += m_pFinal->GetLat(m_ClusMembers[i].GetAt(j));
                }

                m_LongCentroids[i] = LongSum / (double)N;
                m_LatCentroids[i] = LatSum / (double)N;
                m_ServeLongCentroids[i] = m_LongCentroids[i];
                m_ServeLatCentroids[i] = m_LatCentroids[i];
            }
        }

        delete pDist;
        return;
}

void CFinalize::ComputePath(int whichPath, CString measure)
{
        if (whichPath < 1 && whichPath > 2)
            return;

        long i, j, k, n, nClus = m_pFinal->m_CurClust->m_nClusters;
        CString str;
        CDWordArray path, clus;
        CStringArray label;
        CDistance pDist;

        double* X = new double[m_pFinal->m_nPoints];
        double* Y = new double[m_pFinal->m_nPoints];
        CPath* pPath = new CPath;
```

© PNR and Associates, All rights reserved

```
switch (whichPath)
{
case 1:
      for (i = 0; i < nClus; i++)
      {
            // Initialize
            n = 0;
            label.RemoveAll();

            // Create distance matrix for each cluster
            for (j = 0; j < m_ClusMembers[i].GetSize(); j++)
            {
                  X[n] = m_pFinal->GetLong(m_ClusMembers[i].GetAt(j));
                  Y[n] = m_pFinal->GetLat(m_ClusMembers[i].GetAt(j));
                  label.Add(m_ClusLabels[i]);
                  label[n++] += ".";
            }
            pDist.Compute((unsigned long)n, X, Y, measure);

            // Calculate the path for each cluster
            path.RemoveAll();
            path.SetSize(n);
            pPath->Run(pDist, n, path, label);

            // Set member variables for cluster
            for (j = 0; j < n; j++)
            {
                  m_ClusPath[m_ClusMembers[i].GetAt(j)] = path[j];
                  m_ClusPathLabel[m_ClusMembers[i].GetAt(j)] = label[j];
            }

            // Assign cluster distance and maximum point distance
            m_ClusDist[i] = pPath->GetDist();
            m_ClusMaxPtDist[i] = pPath->GetPointMax();
      }
      break;

case 2:
      for (i = 0; i < m_ActualClusters.GetSize(); i++)
      {
            // Initialize
            n = 0;
            clus.RemoveAll();
```

© PNR and Associates, All rights reserved

```
label.RemoveAll();

// Set actual cluster as beginning
X[n] = m_LongCentroids[m_ActualClusters[i]];
Y[n] = m_LatCentroids[m_ActualClusters[i]];
label.Add(m_ClusLabels[m_ActualClusters[i]]);
label[n++] += ".";
clus.Add(m_ActualClusters[i]);

// Create distance matrix for each cluster
for (j = 0; j < m_OutlierClusters.GetSize(); j++)
{
      if (m_ClusLabels[m_OutlierClusters[j]] == m_ClusLabels[m_ActualClusters[i]])
      {
            X[n] = m_LongCentroids[m_OutlierClusters[j]];
            Y[n] = m_LatCentroids[m_OutlierClusters[j]];
            label.Add(m_ClusLabels[m_ActualClusters[i]]);
            label[n++] += ".";
            clus.Add(m_OutlierClusters[j]);
      }
}
pDist.Compute((unsigned long)n, X, Y, measure);

// Calculate the path for each cluster
path.RemoveAll();
path.SetSize(n);
pPath->Run(pDist, n, path, label);

// Set member variables for cluster
for (j = 0; j < n; j++)
{
      m_GrpPath[clus[j]] = path[j];
      m_GrpPathLabel[clus[j]] = label[j];
}

// Assign serving centroid
for (j = 0; j < n; j++)
{
      k = m_GrpPathLabel[clus[j]].GetLength() - 3;
      str = m_GrpPathLabel[clus[j]].Left(k);

      if (str.GetLength() < 3)
      {
            m_ServeLongCentroids[clus[j]] = m_LongCentroids[clus[j]];
```

© PNR and Associates, All rights reserved

```
                        m_ServeLatCentroids[clus[j]] = m_LatCentroids[clus[j]];
                    }
                    else
                    {
                        k = -1;
                        while (str != m_GrpPathLabel[clus[++k]]);
                        m_ServeLongCentroids[clus[j]] = m_LongCentroids[clus[k]];
                        m_ServeLatCentroids[clus[j]] = m_LatCentroids[clus[k]];
                    }
                }

                // Assign cluster distance and maximum point distance
                m_GrpDist[clus[0]] = pPath->GetDist();
                m_GrpMaxPtDist[clus[0]] = pPath->GetPointMax();
            }
            break;
        }

        delete X;
        delete Y;
        delete pPath;
}


/*
void CFinalize::ComputePerimeter()
{
        int j, ic, cluster;
        int nClus = m_pFinal->m_CurClust->m_nClusters;
        long i, nCols, row, col;
        CDWordArray p;
        Point pt;
        CPointArray pXY;
        CPerim* pPerim = new CPerim;

        // Calculate perimiter of final cluster solution
        for (ic = 0; ic < nClus; ic++)
        {
            pt.x = 0;
            pt.y = 0;
            pXY.RemoveAll();

            cluster = m_pFinal->m_CurClust->GetCluster(ic);
            CDWordArray *ClustList = m_pFinal->m_CurClust->GetClustList(cluster);
```

© PNR and Associates, All rights reserved

```
            for (i = 0; i < ClustList->GetSize(); i++)
            {
                  nCols = m_pFinal->m_ClGrid.GetNCols();
                  row = (long)floor(ClustList->GetAt(i) / nCols);
                  col = ClustList->GetAt(i) - row * nCols;

                  pt.x = col;
                  pt.y = row;
                  pXY.Add(pt);
            }

            pPerim->Compute(pXY, p);

            for (j = 0; j < p.GetSize(); j++)
            {
                  long pos2 = ClustList->GetAt(p[j]);
                  for (i = 0; i < m_pFinal->m_nPoints; i++)
                  {
                        if (m_pFinal->m_CurClust->m_pCluster[i] == cluster)
                        {
                              long pos = m_pFinal->m_ClGrid.GetPCol(i) +
                                          m_pFinal->m_ClGrid.GetPRow(i) *
                                            m_pFinal->m_ClGrid.GetNCols();

                              if (pos2 == pos)
                              {
                                    m_Perimeter[i] = j+1;
//                                  m_Vertices[ic].Add(m_pFinal->m_pLong[i]);
//                                  m_Vertices[ic].Add(m_pFinal->m_pLat[i]);
                                    break;
                              }
                        }
                  }
            }
      }

      delete pPerim;
}
*/


// GRDCLUST.H header files
```

© PNR and Associates, All rights reserved

```
#ifndef GRDCLUST__H
#define GRDCLUST__H

#include "Global.h"
class COutput;

class CData;

class CClustGrid
{
public:
      CClustGrid();
      ~CClustGrid();

      CClustGrid(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet, long nPoints,
CDblArray *Longs, CDblArray *Lats, float *Weight);
      void Initialize(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet, long nPoints,
CDblArray *Longs, CDblArray *Lats, float *Weight, CDWordArray* pPtMask = NULL);

      // Get attribute funcitons
      long GetCell(long pos) {return Index(m_pCol[pos], m_pRow[pos]);}
      long GetDistance() {return m_dFeet;}
      int GetMember(long pos) {return m_Member[pos];}
      float GetLines(long pos) {return m_Lines[pos];}
      long GetSize() {return m_nCols * m_nRows;}
      long GetNCols() {return m_nCols;}
      long GetNRows() {return m_nRows;}
      long GetPRow(long pos) {return m_pRow[pos];}
      long GetPCol(long pos) {return m_pCol[pos];}
      long Index(long x,long y) {return x + y*m_nCols;}
      long GetDistance(long Cell1, long Cell2);
      long GetRLDistance(long Cell1, long Cell2);

      // Set attribute funcitons
      void SetMember(long pos, long value) {m_Member[pos] = value;}
      void SetLines(long pos, float value) {m_Lines[pos] = value;}

private:
      long m_dFeet;
      double m_Height;
      double m_Width;
      long m_nRows;
      long m_nCols;
```

© PNR and Associates, All rights reserved

```
//      long *m_Grid;
        long *m_Member;
        float *m_Lines;
        long m_nPoints;
        long *m_pRow;
        long *m_pCol;
};

class CClustSet : public CObject
{
public:
        CClustSet() {}
        DECLARE_DYNCREATE(CClustSet)

        CClustSet(long nPoints, long feet);
        CClustSet(CClustSet *CSet);
        ~CClustSet();

        long GetSize() {return m_nClusters;}
        long GetCluster(long pos) {return m_Cluster[pos];}
        long GetOutStart() {return m_OutlStart;}
        BOOL IsOutlier(long);
        float GetClustLines(long Cluster);
        void SetCluster(int pos, int cluster) {m_Cluster[pos] = cluster;}
        void SetPointClust(long  point, long cluster) {m_pCluster[point] = cluster;}
        void SetOutlier(BOOL Outliers, long CurMembPos);
        void SetOutlierFlag(BOOL Outlier, long CurMembPos) {m_Outlier[CurMembPos] = Outlier;}

        long pCluster(long Point) {return m_pCluster[Point];} // Point Cluster
        void ClusConsolidate();
        void ResetClusts();
        BOOL Add(int  CurClust, long CellCount, float LineCount, CDWordArray *CellList);
        BOOL IsNoMerge(int Clust, long NM_Count);
        BOOL IsMergable(int Clust, long LM_Count);
        CDWordArray *GetClustList(long Cluster);

        void AddSplitsSub(int Cluster, CDWordArray &PtList, float &nLines, CDWordArray &CellList, float
*Weights, CClustGrid &Grid);
        void AddSplits(CObArray &NewClusts, CDWordArray &Oversized, CDWordArray &StartPos, float *Weights,
CClustGrid &Grid, int MinLines);

//Attributes
        long m_Feet;      // Distance this set created at
```

© PNR and Associates, All rights reserved

```
      // Cluster Information
      long m_nMaxClusters;
      long m_nClusters;

      long m_OutlStart;
      long m_OutlMemStart;

      CByteArray m_Outlier;
      CDblArray m_NumLines;
      CWordArray m_Cluster;
      CObArray m_ACellList;

      // Point Data
      long m_nPoints;
      int *m_pCluster;
};

class CClustComp
{
public:
      CClustComp(CData* Data = NULL, SPECS* Specs = NULL);
      ~CClustComp();

      BOOL Initialize(CData* Data, SPECS* Specs, CString &Errors);

// Attributes
      // Grid Specific
      CClustGrid m_ClGrid;

      // Point Data
      double m_MinLong;
      double m_MaxLong;
      double m_MinLat;
      double m_MaxLat;
      double m_Width;
      double m_Height;

      long m_MaxPoints;
      long m_nPoints;
    CDblArray *m_pLong;
    CDblArray *m_pLat;
    float *m_pWeight;

    // Cluster Data
```

© PNR and Associates, All rights reserved

```
    long m_LastCluster;       // number clusters (total count)
      CClustSet *m_CurClust;

      // Parameters
      int m_MinLines;
      long m_LM_Count;
      long m_NM_Count;
      long m_MaxClustDist;

      CStdioFile m_fTrace;

// Implementation
      double GetLong(long index) {return m_pLong->GetAt(index);}
      double GetLat(long index) {return m_pLat->GetAt(index);}

      void Clear();
    void SetGrid(long feet);
      void SetTrace(CString &FileName);
      void ComputeClusters(SPECS* Specs, BOOL Outliers, COutput *OutputFile); void ProcessCell(int row, int
col, char *Checked, int loop, CClustSet *NextClust);
      void CheckAdjacent(long iStart, long jStart, char *Checked, CDWordArray &CellList, int loop, BOOL
&Mergable, BOOL NoMerge, CWordArray &WithinClusts, CWordArray &RejectClusts);

      long Index(long x,long y) {return x + y*m_ClGrid.GetNCols();}
      BOOL WithinMaxDist(CDWordArray &CellList, long CurCell, CWordArray &WithinClusts, CWordArray
&RejectClusts);
      void ClusterCoord(int, CPointArray&);
      void Encompassed(CDWordArray *ClustList, char *Checked);

      void CloseTrace() {m_fTrace.Close();}
};

#endif

// GRDCLUST.CPP implementation of clustering algorithms

#include "stdafx.h"
#include "GrdClust.h"
#include "Data.h"
#include "Perimeter.h"

#include "string.h"
#include "math.h"
```

© PNR and Associates, All rights reserved

```
////////////////////////////////////////////////////////////////////////////////////////////
//
//  Distance functions
//
////////////////////////////////////////////////////////////////////////////////////////////

static double Radius = 3956.0;                          // approx earth radius in miles
static double Radians = acos(-1.0) / 180.0;     // pi / 180
static double Distance(double x1, double y1, double x2, double y2)
{
      double Distance = 0.0;

      if (x1 != x2 || y1 != y2) {
            x1 *= Radians;
            x2 *= Radians;
            y1 *= Radians;
            y2 *= Radians;

            Distance = pow(sin((y2-y1) / 2.0), 2.0) + cos(y1) * cos(y2) * pow(sin((x2-x1) / 2.0), 2.0);
            Distance = (2 * asin(min(1, pow(Distance, .5)))) * Radius;
      }

      return Distance;
}


static double LatFromDist(double Lat, double Distance)
{
      return (Distance/Radius + Lat*Radians) / Radians;
}

static double LongFromDist(double x1, double y1, double Distance)
{
      x1 *= Radians;
      y1 *= Radians;

      return (x1 + 2 * asin(pow( pow(sin(Distance/(2*Radius)), 2.) / (cos(y1)*cos(y1)), .5))) / Radians;
}

////////////////////////////////////////////////////////////////////////////////////////////
//
//  CClustGrid:  Grid class for cluster computation
//
////////////////////////////////////////////////////////////////////////////////////////////
```

© PNR and Associates, All rights reserved

```
CClustGrid::CClustGrid()
{
      m_dFeet = 0;
      m_Height = 0;
      m_Width = 0;
      m_nPoints = 0;

      m_Lines = NULL;
      m_Member = NULL;
      m_nRows = NULL;
      m_nCols = NULL;
      m_pRow = NULL;
      m_pCol = NULL;
}

CClustGrid::CClustGrid(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet, long
nPoints, CDblArray *pLong, CDblArray *pLat, float *pWeight)
{
      m_dFeet = 0;
      m_Height = 0;
      m_Width = 0;
      m_nPoints = 0;

      m_Lines = NULL;
      m_Member = NULL;
      m_nRows = NULL;
      m_nCols = NULL;
      m_pRow = NULL;
      m_pCol = NULL;

      Initialize(MinLong, MinLat, MaxLong, MaxLat, feet, nPoints, pLong, pLat, pWeight);
}

CClustGrid::~CClustGrid()
{
      delete[] m_Lines;
      delete[] m_Member;
      delete[] m_pRow;
      delete[] m_pCol;
}

long CClustGrid::GetDistance(long Cell1, long Cell2)
{
```

© PNR and Associates, All rights reserved

```
	long row1 = (long)floor(Cell1 / m_nCols);
	long col1 = Cell1 - row1*m_nCols;
	long row2 = (long)floor(Cell2 / m_nCols);
	long col2 = Cell2 - row2*m_nCols;

	float distance = (float)(pow(pow(row2-row1,2) + pow(col2-col1,2), .5) * m_dFeet);

	return (long)distance;
}

long CClustGrid::GetRLDistance(long Cell1, long Cell2)
{
	long row1 = (long)floor(Cell1 / m_nCols);
	long col1 = Cell1 - row1*m_nCols;
	long row2 = (long)floor(Cell2 / m_nCols);
	long col2 = Cell2 - row2*m_nCols;

	float distance = (float)(labs(row2-row1) + labs(col2-col1)) * m_dFeet;


	return (long)distance;
}

void CClustGrid::Initialize(double MinLong, double MinLat, double MaxLong, double MaxLat, long feet, long
nPoints, CDblArray *pLong, CDblArray *pLat, float *pWeight, CDWordArray* pPtMask /*NULL*/)
{
	if (!nPoints || !feet)
		return;

	// Determine width and height of grid
	double dist1 = Distance(MinLong, MinLat, MaxLong, MinLat);
	double dist2 = Distance(MinLong, MaxLat, MaxLong, MaxLat);
	m_Width = (dist1 > dist2) ? dist1 : dist2;
	m_Height = Distance(MinLong, MinLat, MinLong, MaxLat);

	// Deallocate
	delete[] m_Lines;
	delete[] m_Member;
	delete[] m_pRow;
	delete[] m_pCol;

	// Initialize member variables
	m_dFeet = feet;
	m_nPoints = nPoints;
```

© PNR and Associates, All rights reserved

```
m_nRows = (long)(m_Height / ((double)m_dFeet/5280.) + 1);
m_nCols = (long)(m_Width / ((double)m_dFeet/5280.) + 1);

// Allcoate member variables
m_Lines = new float[m_nRows * m_nCols];
m_Member = new long[m_nRows * m_nCols];
for (long Point = 0; Point < long(m_nRows * m_nCols); Point++) {
      m_Lines[Point] = 0;
      m_Member[Point] = 0;
}

m_pRow = new long[m_nPoints];
m_pCol = new long[m_nPoints];
for (Point = 0; Point < m_nPoints; Point++)
      m_pRow[Point] = m_pCol[Point] = 0;

// Assign point cell locations
BOOL bFound;
long CurPoint = 0, nMaskIndex = 0;
double miles = (double)m_dFeet/5280.;
double BegLat, BegLong, EndLat, EndLong;
for (Point = 0; Point < m_nPoints; Point++) {         // Loop works because data sorted by Lat then
Long

      bFound = FALSE;
      if (pPtMask) {
            if (pPtMask->GetAt(nMaskIndex) == (DWORD)Point)
                  ++nMaskIndex;
            else
                  bFound = TRUE;
      }

      if (!bFound) {
            BegLat = EndLat = MinLat;
            for (long i = 0; i < m_nRows; i++) {
                  BegLat = EndLat;
                  EndLat = LatFromDist(BegLat, miles);

                  if (!(pLat->GetAt(Point) < EndLat))
                        continue;

                  BegLong = EndLong = MinLong;
                  for (long j = 0; j < m_nCols; j++) {
                        BegLong = EndLong;
```

© PNR and Associates, All rights reserved

```
                                EndLong = LongFromDist(BegLong, (BegLat + EndLat) / 2.0, miles);
                                if (pLong->GetAt(Point) < EndLong) {
                                      m_pRow[Point] = i;
                                      m_pCol[Point] = j;
                                      bFound = TRUE;
                                      break;
                                }
                          }

                          if (bFound)
                                break;
                    }
              }
              if (pPtMask && nMaskIndex == pPtMask->GetSize())
                    break;
        }

        for (long i = 0; i < m_nPoints; i++) {
              m_Lines[Index(GetPCol(i), GetPRow(i))] += pWeight[i];
        }
}

////////////////////////////////////////////////////////////////////////////////////////////
//
//  CClustSet:  Storage for cluster information
//
////////////////////////////////////////////////////////////////////////////////////////////

IMPLEMENT_DYNCREATE(CClustSet, CObject)

CClustSet::CClustSet(CClustSet *CSet)
{
      m_Feet = CSet->m_Feet;

      m_nMaxClusters = CSet->m_nMaxClusters;

      m_nPoints = CSet->m_nPoints;
      m_pCluster = new int[m_nPoints];
      for (long i=0; i<m_nPoints; i++)
            m_pCluster[i] = CSet->m_pCluster[i];

      m_OutlStart = CSet->m_OutlStart;
      m_OutlMemStart = CSet->m_OutlMemStart;
      m_nClusters = CSet->m_nClusters;
```

© PNR and Associates, All rights reserved

```
	for (i=0; i<m_nClusters; i++) {
		m_NumLines.Add(CSet->m_NumLines[i]);
		m_Cluster.Add(CSet->m_Cluster[i]);
	}

	for (i=0; i<CSet->m_ACellList.GetSize(); i++)
		m_ACellList.Add(CSet->m_ACellList[i]);
}

CClustSet::CClustSet(long nPoints, long feet)
{
	m_Feet = feet;
	m_OutlStart = 0;
	m_OutlMemStart = 0;
	m_nClusters = 0;
	m_nMaxClusters = 10000;

	m_nPoints = nPoints;
	m_pCluster = new int[nPoints];
	for (long i=0; i<nPoints; i++)
		m_pCluster[i] = 0;
}

CClustSet::~CClustSet()
{
	delete[] m_pCluster;

	for (long i=0; i<m_ACellList.GetSize(); i++)
		delete m_ACellList[i];

	m_ACellList.RemoveAll();
}


void CClustSet::ClusConsolidate()
{
	for (long i=0; i<m_nClusters; i++) {
		long count=0;
		for (long n=0; n<(long)m_nPoints; n++) {
			if (m_pCluster[n] == m_Cluster[i]) {
				count++;
				m_pCluster[n] = 10000+i+1;
			}
		}
```

© PNR and Associates, All rights reserved

```
            m_Cluster[i] = i+1;

            m_Outlier.Add( (i < GetOutStart()-1) ? FALSE : TRUE);
      }

      for (long n=0; n<(long)m_nPoints; n++) {
            if (m_pCluster[n])
                  m_pCluster[n] -= 10000;
      }


}

void CClustSet::ResetClusts()
{
      if (m_OutlStart)
            m_nClusters = m_OutlStart - 1;
      else
            m_nClusters = 0;

      for (long i=m_nClusters; i<m_ACellList.GetSize(); i++)
            delete m_ACellList[i];

      long size = m_ACellList.GetSize();
      for (i=m_nClusters; i<size; i++) {
            m_NumLines.RemoveAt(m_nClusters);
            m_Cluster.RemoveAt(m_nClusters);
            m_ACellList.RemoveAt(m_nClusters);
      }
}

void CClustSet::AddSplitsSub(int Cluster, CDWordArray &PtList, float &nLines, CDWordArray &CellList, float
*Weights, CClustGrid &Grid)
{
      for (long i=0; i<PtList.GetSize(); i++) {
            m_pCluster[PtList[i]] = Cluster;
            nLines += Weights[i];
            long Cell = Grid.GetCell(i);

            for (long j=0; j<CellList.GetSize(); j++) {
                  if ((DWORD)Cell == CellList[j])
                        break;
            }
```

© PNR and Associates, All rights reserved

```
            if (j == CellList.GetSize())
                  CellList.Add(Cell);
      }

}

void CClustSet::AddSplits(CObArray &NewClusts, CDWordArray &Oversized, CDWordArray &StartPos, float
*Weights, CClustGrid &Grid, int MinLines)
{
      int Cluster, Count=0;

      for (long i=0; i<NewClusts.GetSize(); i++) {
            CDWordArray *PtList = (CDWordArray *)NewClusts[i];

            float nLines = 0;
            CDWordArray *CellList = new CDWordArray;

            if (StartPos[Count] == (DWORD)i) {        // Replace First, Compute Lines, Compute Cells
                  Cluster = Oversized[Count++]+1;
                  AddSplitsSub(Cluster, *PtList, nLines, *CellList, Weights, Grid);
                  delete m_ACellList[Cluster-1];
                  m_ACellList.SetAt(Cluster-1, CellList);
                  m_NumLines[Cluster-1] = nLines;
            } else {          // Append new...,  Set Outlier=FALSE, Compute Lines, Set Cluster ID, Compute
Cell List
                  int NewCluster = ++m_nClusters;
                  m_Cluster.Add(NewCluster);

                  AddSplitsSub(NewCluster, *PtList, nLines, *CellList, Weights, Grid);

                  if (nLines >= MinLines)
                        m_Outlier.Add(FALSE);
                  else
                        m_Outlier.Add(TRUE);

                  m_ACellList.Add(CellList);
                  m_NumLines.Add(nLines);

            }
      }

/*
      for (int n=0; n<Oversized.GetSize(); n++) {
```

© PNR and Associates, All rights reserved

```
            int Cluster = Oversized[n]+1;

            for (long i=0; i<NewClusts.GetSize(); i++) {
                  CDWordArray *PtList = (CDWordArray *)NewClusts[StartPos[n]+i];

                  long nLines = 0;
                  CDWordArray *CellList = new CDWordArray;

                  if (!i) {         // Replace First, Compute Lines, Compute Cells
                        AddSplitsSub(Cluster, *PtList, nLines, *CellList, Weights, Grid);
                        delete m_ACellList[Cluster-1];
                        m_ACellList.SetAt(Cluster-1, CellList);
                        m_NumLines[Cluster-1] = nLines;
                  } else {          // Append new...,  Set Outlier=FALSE, Compute Lines, Set Cluster ID,
Compute Cell List
                        int NewCluster = ++m_nClusters;
                        m_Cluster.Add(NewCluster);

                        AddSplitsSub(NewCluster, *PtList, nLines, *CellList, Weights, Grid);
                        m_Outlier.Add(FALSE);
                        m_ACellList.Add(CellList);
                        m_NumLines.Add(nLines);

                  }
            }
      }
*/
}

BOOL CClustSet::Add(int CurClust, long CellCount, float LineCount, CDWordArray *CellList)
{
      if (m_nClusters+1 >= m_nMaxClusters) {
            AfxMessageBox("Error, Max Clusters Exceeded");
            return FALSE;
      }

      for (long i=0; i<m_nClusters; i++) {                  // $CHANGE$ Hopefully remove complete loop
            if (CurClust == m_Cluster[i]) {
AfxMessageBox("Repeating Cluster Add");
                  m_NumLines[i] += LineCount;

                  int offset = (m_OutlStart) ? m_OutlStart-1 : 0;
                  CDWordArray *CurList = (CDWordArray *)m_ACellList[i-offset];
                  for (long j=0; j<CellList->GetSize(); j++) {
```

© PNR and Associates, All rights reserved

```
                        CurList->Add(CellList->GetAt(j));
                    }

                    delete CellList;

                    break;
                }
        }

        if (i == m_nClusters) {
                m_NumLines.Add(LineCount);
                m_Cluster.Add(CurClust);
                m_ACellList.Add(CellList);

                m_nClusters++;
        }

        return TRUE;
}

void CClustSet::SetOutlier(BOOL Outliers, long CurMembPos)
{
        if (Outliers) {
                m_OutlStart = m_nClusters+1;
                m_OutlMemStart = CurMembPos+1;
        } else {
                m_OutlStart = FALSE;
                m_OutlMemStart = 0;
        }
}

BOOL CClustSet::IsNoMerge(int Clust, long NM_Count)
{
return FALSE;                           // $CHANGE$ Cluster Size Reduction occurs in finalization
        BOOL NoMerge = FALSE;

        for (long i=0; i<m_nClusters; i++) {
                if (m_Cluster[i] == Clust) {
//                      if (m_NumLines[i] > NM_Count || (Clust < m_OutlStart))
                        if (m_NumLines[i] > NM_Count || (Clust < m_OutlMemStart))
                                NoMerge = TRUE;
                        break;
                }
        }
```

© PNR and Associates, All rights reserved

```
	return NoMerge;
}

BOOL CClustSet::IsMergable(int Clust, long LM_Count)
{
	BOOL IsMergable = TRUE;

	for (long i=0; i<m_nClusters; i++) {
		if (m_Cluster[i] == Clust) {
			if (m_NumLines[i] > LM_Count)
				IsMergable = FALSE;
			break;
		}
	}

	return IsMergable;
}

float CClustSet::GetClustLines(long Cluster)
{
	float nLines = 0;

	for (long i=0; i<m_nClusters; i++) {
		if (Cluster == m_Cluster[i]) {
			nLines = (float)m_NumLines[i];
			break;
		}
	}

	return nLines;
}

CDWordArray *CClustSet::GetClustList(long Cluster)
{
	for (long i=0; i<m_nClusters; i++)
		if (Cluster == m_Cluster[i])
			return ((CDWordArray *)m_ACellList[i]);

	return NULL;
}

BOOL CClustSet::IsOutlier(long index)
{
```

© PNR and Associates, All rights reserved

```
	return m_Outlier[index];
/*
	long BegOutIndex = GetOutStart() - 1;
	if (index < BegOutIndex)
		return FALSE;
	else
		return TRUE;
*/
}


////////////////////////////////////////////////////////////////////////////////////////////
//
//  CClustComp:  Calculates cluster solution using grid and storage classes
//
////////////////////////////////////////////////////////////////////////////////////////////

CClustComp::CClustComp(CData* Data, SPECS* Specs)
{
	m_pWeight = NULL;
	m_CurClust = NULL;

	Clear();

	CString Error;
	if (Data && Specs)
		Initialize(Data, Specs, Error);
}

CClustComp::~CClustComp()
{
	delete m_pWeight;
	delete m_CurClust;
}

void CClustComp::SetTrace(CString &FileName)
{
	CString TraceFile = FileName + ".trace.txt";
	m_fTrace.Open(TraceFile, CFile::modeWrite | CFile::modeCreate);
}

void CClustComp::Clear()
{
	m_nPoints = 0;
```

© PNR and Associates, All rights reserved

```
	m_pLong = NULL;
	m_pLat = NULL;
	m_MinLong = 0;
	m_MaxLong = 0;
	m_MinLat = 0;
	m_MaxLat = 0;

	m_Width = 0;
	m_Height = 0;
	m_LastCluster = 0;
	m_MinLines = 0;
	m_LM_Count = 0;
	m_NM_Count = 0;

	m_MaxClustDist = 36000; // Should be a user accessable parameter
//	m_MaxClustDist = 24000; // Should be a user accessable parameter

	delete m_pWeight;
	m_pWeight = NULL;

	delete m_CurClust;
	m_CurClust = NULL;
}

BOOL CClustComp::Initialize(CData* Data, SPECS* Specs, CString &Errors)
{
	if (m_nPoints)
		Clear();
	m_MaxClustDist -= (2 * Specs->nGridSize);	// Adjust maximum clust distance for grid size errors

	m_nPoints = Data->GetNCases();
	m_pLong = Data->GetLongitude();
	m_pLat = Data->GetLatitude();

	m_pWeight = new float[m_nPoints];
	for (long i = 0; i < m_nPoints; i++)
		m_pWeight[i] = (float)Data->GetWt(i);

	// Determine the minimum/maximum longitudes and latitudes for grid compilation
	m_MinLong = GetLong(0);
	m_MaxLong = GetLong(0);
	m_MinLat = GetLat(0);
	m_MaxLat = GetLat(0);
	for (i = 1; i < m_nPoints; i++)
```

© PNR and Associates, All rights reserved

```
        {
                if (m_MinLong > GetLong(i))
                        m_MinLong = GetLong(i);

                if (m_MaxLong < GetLong(i))
                        m_MaxLong = GetLong(i);

                if (m_MinLat > GetLat(i))
                        m_MinLat = GetLat(i);

                if (m_MaxLat < GetLat(i))
                        m_MaxLat = GetLat(i);
        }


        double dist1 = Distance(m_MinLong, m_MinLat, m_MaxLong, m_MinLat);
        double dist2 = Distance(m_MinLong, m_MaxLat, m_MaxLong, m_MaxLat);
        double Width = (dist1 > dist2) ? dist1 : dist2;
        double Height = Distance(m_MinLong, m_MinLat, m_MinLong, m_MaxLat);

        if (Width > 100 || Height > 100) {
                char buffer[256];
                sprintf(buffer, "Width=%.0f   Height=%.0f    Lat = %.6f, %.6f   Long = %.6f, %.6f",
                                                                Width, Height, m_MinLat, m_MaxLat, m_MinLong,
m_MaxLong);
                Errors = buffer;
                return FALSE;
        }

        // Initialize grid that will contain cluster solutions
        m_ClGrid.Initialize(m_MinLong, m_MinLat, m_MaxLong, m_MaxLat, Specs->nGridSize, m_nPoints, m_pLong,
m_pLat, m_pWeight);

        // Initialize grid that will contain cluster solutions
//      m_Member.Initialize(m_MinLong, m_MinLat, m_MaxLong, m_MaxLat, Specs->nGridSize, m_nPoints, m_pLong,
m_pLat);

        // Initialize and set grid containing number of weighted lines
//      m_Lines.Initialize(m_MinLong, m_MinLat, m_MaxLong, m_MaxLat, Specs->nGridSize, m_nPoints, m_pLong,
m_pLat);
//      for (i = 0; i < m_nPoints; i++)
//      {
//              long row = m_Lines.GetPRow(i);
//              long col = m_Lines.GetPCol(i);
```

© PNR and Associates, All rights reserved

```
//          m_Lines.SetValue(Index(col, row), m_ClGrid.GetLines(Index(col, row)) + m_pWeight[i]);
//    }

      return TRUE;
}

BOOL CClustComp::WithinMaxDist(CDWordArray &CellList, long CurCell, CWordArray &WithinClusts, CWordArray
&RejectClusts)
{
      long i, j, n;

      int NewClust = m_ClGrid.GetMember(CurCell);

      // Check if Current Cell is a member of the approved cluster list
      if (NewClust) {
            for (i=0; i<WithinClusts.GetSize(); i++)
                  if (WithinClusts[i] == NewClust)
                        return TRUE;
      }

      // If New Cell is not yet Clustered, Compare with all other Unclustered Cells in CellList and with
Current clusts
      if (!NewClust) {
            for (i=0; i<CellList.GetSize(); i++) {
                  if (m_ClGrid.GetRLDistance(CellList[i], CurCell) > m_MaxClustDist) {
                        return FALSE;
                  }
            }

            for (n=0; n<WithinClusts.GetSize(); n++) {
                  CDWordArray *CurClustList = m_CurClust->GetClustList(WithinClusts[n]);

                  for (i=0; i<CurClustList->GetSize(); i++) {
                        if (m_ClGrid.GetRLDistance(CurClustList->GetAt(i), CurCell) > m_MaxClustDist) {
                              return FALSE;
                        }
                  }
            }
      } else {
            // check if new cluster already has failed merge with current cluster
static BOOL Test=TRUE;
            for (i=0; i<RejectClusts.GetSize(); i++) {
                  if (RejectClusts[i] == NewClust)
                        return FALSE;
```

© PNR and Associates, All rights reserved

```
            }

            // Else For Each member of included clusters, compare to new cluster
            CDWordArray *pNewClustList = m_CurClust->GetClustList(NewClust);

            for (i=0; i<CellList.GetSize(); i++) {
                  for (j=0; j<pNewClustList->GetSize(); j++) {
                        if (m_ClGrid.GetRLDistance(CellList[i], pNewClustList->GetAt(j)) > m_MaxClustDist) {
                              if (NewClust && Test)
                                    RejectClusts.Add(NewClust);
                              return FALSE;
                        }
                  }
            }

            // If existing clusters, For Each member of existing cluster, compare to new cluster
            if (WithinClusts.GetSize()) {
                  for (n=0; n<WithinClusts.GetSize(); n++) {
                        CDWordArray *CurClustList = m_CurClust->GetClustList(WithinClusts[n]);

                        for (i=0; i<CurClustList->GetSize(); i++) {
                              for (j=0; j<pNewClustList->GetSize(); j++) {
                                    if (m_ClGrid.GetRLDistance(CurClustList->GetAt(i), pNewClustList-
>GetAt(j)) > m_MaxClustDist) {
                                          if (NewClust && Test)
                                                RejectClusts.Add(NewClust);
                                          return FALSE;
                                    }
                              }
                        }
                  }
            }
      }

      // If cluster passes the test, insert into Cluster list
      if (NewClust)
            WithinClusts.Add(NewClust);

      return TRUE;
}


void CClustComp::CheckAdjacent(long iStart, long jStart, char *Checked, CDWordArray &CellList, int loop,
BOOL &Mergable, BOOL NoMerge, CWordArray &WithinClusts, CWordArray &RejectClusts)
```

© PNR and Associates, All rights reserved