EX PARTE OR LATE FILED

DOCKET FILE COPY ORIGINAL

AT&T

295 North Maple Avenue
Basking Ridge, NJ 07920

January 13, 1998

RECEIVED

JAN 13 1998

FEDERAL COMMUNICATIONS COMMISSION
OFFICE OF THE SECRETARY

Ms. Magalie Roman Salas
Secretary
Federal Communications Commission
1919 M. St., NW, Room 222
Washington, D.C. 20554

RE: Ex Parte Presentation – Proxy Cost Models
CC Docket No. 96-45

Dear Ms. Salas:

Attached, please find a copy of the C++ code for PNR's Spatial Clustering Module. This code, which is only very slightly revised from that filed on September 30, 1997 with the Commission, is used by PNR to determine the customer locations that cluster together for use by the Hatfield Model, v5.0. In addition, we attach some further runs of the Hatfield Model using the Commission staff's set of "unified" input values. These runs differ from those submitted previously in that they assume that all residence line costs in excess of $31.00 per month, and all business line costs in excess of $51.00 per month will receive support. In addition runs are supplied using a maximum analog copper distance of 12,000 feet.

Two copies of this Notice are being submitted to the Secretary of the FCC in accordance with Section 1.1206(a)(1) of the Commission's rules.

Sincerely,

Richard N. Clarke

Attachments

cc: S. Todd
R. Loube
N. Wales

No. of Copies rec'd 0+1
List A B C D E

```
// Calling routine that runs SCM

void CClusterDlg::OnRun()
{
	if (access(m_dataFileName, 0)) {
		if (!m_Batch)
			AfxMessageBox("ERROR: Data file is missing " + m_dataFileName);
		else
			m_Error = 2;

		return;
	}

	if (!access(m_outFileName, 0) && (!m_Batch || !m_Overwrite)) {
		if (AfxMessageBox("Do you want to overwrite " + m_outFileName + "?", MB_YESNO) == IDNO)
			return;
	}

	// Show run status
	m_Status.ShowWindow(TRUE);
	CStatus Stat(&m_Status);
	Stat.SetText("Setting up run ...");

	// Disable app buttons
	EnableDlg(FALSE);

	// Open the data and output files
	try {
		if (!OpenFiles()) {
			EnableDlg(TRUE);
			return;
		}
	} catch(CException* e) {
		throw(e);
		return;
	}

	// Write specifications to log file
	m_OutputFile->SaveLog(m_Specs, m_dataFileName, m_outFileName);

#ifdef _DEBUG
	// Set trace
	m_Cluster->SetTrace(m_outFileName);
#endif
```

© PNR and Associates, All rights reserved

```
	// Open the mapping database
	CMapDlg mapDB;
	if (m_SaveMapDB)
		mapDB.OpenDB();
	BOOL bNewDB = TRUE;

	// Process all of the blocks on the list
	CString str;
	BOOL bDone = FALSE;
	while (!bDone)
	{
		// Initialize member variables based on each Block Group Processed
		Init();

		// Read in data
		Stat.SetText("Retrieving data ...");
		bDone = GetData();

		// Initialize the status bar to reflect current block group
		CString StatLn1 = "Wire Center ";

BOOL dbg_skip=FALSE;
#ifdef _DEBUG
long item = m_DataFile->GetNDone();
static long BreakAt=0;
if (item < BreakAt)
	dbg_skip = TRUE;
else
	dbg_skip = dbg_skip;
#endif

		StatLn1 += Stat.GetNumText(m_DataFile->GetNDone()) + " ";
		StatLn1 += m_DataFile->GetGroup(m_DataFile->GetNDone());
		StatLn1 += " \r\n";
		Stat.PeekMsg();
		if (!m_bSkip && !dbg_skip) {
			// Initialize cluster
			Stat.SetText(StatLn1, "  Initializing cluster grid ...");
			CString Errors;
			if (!m_Cluster->Initialize(m_DataFile, &m_Specs, Errors)) {
				char buff[20];
				m_OutputFile->SaveLog("\nWarning: Oversized WireCenter: " +
(CString)itoa(m_DataFile->GetNDone(), buff, 10) + " " + m_DataFile->GetGroup(m_DataFile->GetNDone()));
```

© PNR and Associates, All rights reserved

```
                m_OutputFile->SaveLog(Errors + "\n");
            } else {
                // Cluster split group
                Stat.SetText(StatLn1, "  Calculating clusters ...");
                m_Cluster->ComputeClusters(&m_Specs, FALSE, m_OutputFile);

                // Cluster outliers
                Stat.SetText(StatLn1, "  Calculating outlier clusters ...");
                m_Cluster->ComputeClusters(&m_Specs, TRUE, m_OutputFile);

                // Allocate member variables based on number of clusters
                Stat.SetText(StatLn1, "  Finalizing cluster solution ...");
                m_Final->Run(m_Cluster, &m_Specs);

                // Save
                Stat.SetText(StatLn1, "Saving output.");
                m_OutputFile->Save(m_DataFile, m_Final);
                if (m_SaveMapDB)
                {
                    Stat.SetText(StatLn1, "Saving map database.");
                    if (bNewDB)
                    {
                        mapDB.Create(m_DataFile, m_Final);
                        bNewDB = FALSE;
                    }
                    else
                        mapDB.Create(m_DataFile, m_Final, TRUE);
                }
            }
        }
    }

    // Clean up
    if (m_SaveMapDB)
    {
        mapDB.CloseDB(m_SaveMapDB);
        CString msg = "IMPORTANT:\r\n\r\nYou must COPY\r\n";
        msg += mapDB.GetDBName();
        msg += "\r\nto another file or the clustering information will be lost on the next run.";
        AfxMessageBox(msg, MB_OK | MB_ICONEXCLAMATION);
    }

    char buffer[80];
    sprintf(buffer, "\nCompleted %d Wire Centers\n", m_DataFile->GetNDone());
```

© PNR and Associates, All rights reserved

```
	m_OutputFile->SaveLog(buffer);

	Stat.SetText("Run complete.");
	CloseFiles();

	m_bRun = TRUE;
	EnableDlg(TRUE);

	return;
}


// Area.h : definition of the CArea class

#ifndef AREA__H
#define AREA__H

#include "array.h"

#define IsClockwise 1		// 1 if Clockwise, -1 if CounterClockwise
#define MaxSlope 1.79e+308
#define MinSlope -1.79e+308

class CArea : public CObject
{
// Construction
public:
	CArea();
	~CArea();

// Attributes
protected:
	CDblArray *m_X;
	CDblArray *m_Y;
	CDWordArray m_Points;
	CDWordArray m_Perimeter;

// Functions
public:
//	double Compute(long n, const double* x, const double* y, const CDWordArray* Outlier = NULL);
	double Compute(const CDWordArray &Member, CDblArray *x, CDblArray *y, CDblArray &Vertices, CDWordArray
&Perimeter, const CDWordArray* Outlier = NULL);
	void Init();
	void GetPerimiterPts(CDWordArray& Perim);
```

© PNR and Associates, All rights reserved

```
protected:
      double Compute();
      double ComputeArea();
      double ComputeDistance(double x1, double y1, double x2, double y2);
      double ComputeSlope(long Point1, long Point2, long Direction);
      long GetNextPoint(long CurPoint, long Direction, BOOL &LastXSloped);
      void GetPerimeter(CDblArray &Vertices, CDWordArray &Perimeter);
};

#endif

// Area.cpp : implementation of the CArea class
//

#include "stdafx.h"
#include "Area.h"
#include "math.h"

/////////////////////////////////////////////////////////////////////////////
// CArea construction/destruction

CArea::CArea()
{
      m_X = NULL;
      m_Y = NULL;
      m_Points.RemoveAll();
      m_Perimeter.RemoveAll();
}

CArea::~CArea()
{
}

/////////////////////////////////////////////////////////////////////////////
// CArea functions

void CArea::Init()
{
      m_X = NULL;
      m_Y = NULL;
      m_Points.RemoveAll();
      m_Perimeter.RemoveAll();
}
```

© PNR and Associates, All rights reserved

```
/*
double CArea::Compute(long n, const double* x, const double* y, const CDWordArray* Outlier /*NULL*//*)
{
	Init();

	m_X = x;
	m_Y = y;

	for (long i = 0; i < n; i++)
	{
		if (Outlier)
		{
			if (!Outlier->GetAt(i))
				m_Points.Add(i);
		}
		else
			m_Points.Add(i);
	}

	// Determine perimeter points of cluster
	GetPerimeter();

	return ComputeArea();
}
*/

double CArea::Compute(const CDWordArray &Member, CDblArray *x, CDblArray *y, CDblArray &Vertices,
CDWordArray &Perimeter, const CDWordArray* Outlier /*NULL*/)
{
	Init();

	m_X = x;
	m_Y = y;

	for (long i=0; i<Member.GetSize(); i++) {
		if (Outlier) {
			if (!Outlier->GetAt(Member[i]))
				m_Points.Add(Member[i]);
		} else {
			m_Points.Add(Member[i]);
		}
	}
```

© PNR and Associates, All rights reserved

```
	// Determine perimeter points of cluster
	GetPerimeter(Vertices, Perimeter);

	return ComputeArea();
}

////////////////////////////////////////////////////////////////////////////////
//
//	CArea:  Area calculation functions
//
////////////////////////////////////////////////////////////////////////////////

double CArea::ComputeArea()
{
	long nPerimPts = m_Perimeter.GetSize();
	double TotalArea = 0.0;

	// Determine origin
	double Origin = m_Y->GetAt(m_Perimeter[0]);
	for (long i = 1; i < nPerimPts; i++)
		if (m_Y->GetAt(m_Perimeter[i]) < Origin)
			Origin = m_Y->GetAt(m_Perimeter[i]);

	// Compute Area of cluster
	long pos1, pos2;
	double Area, yMin, yMax;
	for (i = 0; i < nPerimPts; i++)
	{
		pos1 = m_Perimeter[i];
		if (i + 1 == nPerimPts)
			pos2 = m_Perimeter[0];
		else
			pos2 = m_Perimeter[i + 1];

		yMin = min(m_Y->GetAt(pos1), m_Y->GetAt(pos2));
		yMax = max(m_Y->GetAt(pos1), m_Y->GetAt(pos2));

		Area = ComputeDistance(m_X->GetAt(pos1), Origin, m_X->GetAt(pos2), Origin) *
			ComputeDistance(m_X->GetAt(pos1), Origin, m_X->GetAt(pos1), yMin) +
		       (ComputeDistance(m_X->GetAt(pos1), yMin, m_X->GetAt(pos2), yMin) *
			ComputeDistance(m_X->GetAt(pos2), yMin, m_X->GetAt(pos2), yMax)) / 2 ;

		// If points are headed East, add distance.  If West, subtract distance.
		if (m_X->GetAt(pos1) < m_X->GetAt(pos2))
```

© PNR and Associates, All rights reserved

```
                TotalArea += Area;
            else
                TotalArea -= Area;
        }

        return TotalArea;
}

double CArea::ComputeDistance(double x1, double y1, double x2, double y2)
{
        // Compute linear euclidian distance
        double Distance = 0.0;
        double Radius = 3956.0; // Approximate radius of the earth in miles
        double pi = acos(-1.0);
        double radians = pi / 180.0;

        x1 *= radians;
        x2 *= radians;
        y1 *= radians;
        y2 *= radians;

        if (x1 != x2 || y1 != y2)
        {
            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, 0.5)))) * Radius;
        }

        return Distance;
}

//////////////////////////////////////////////////////////////////////////////
//
//    CArea:  Perimeter functions
//
//////////////////////////////////////////////////////////////////////////////

double CArea::ComputeSlope(long Point1, long Point2, long Direction)
{
        double slope = 0.0;

        if (m_Y->GetAt(Point1) == m_Y->GetAt(Point2))
            slope = 0.0;
        else if (m_X->GetAt(Point1) == m_X->GetAt(Point2))
            slope = (m_Y->GetAt(Point2) < m_Y->GetAt(Point1) || Direction) ? MinSlope : MaxSlope;
```

© PNR and Associates, All rights reserved

```
	else
		slope = (m_Y->GetAt(Point2) - m_Y->GetAt(Point1)) / (m_X->GetAt(Point2) - m_X->GetAt(Point1));

	return slope;
}

long CArea::GetNextPoint(long CurPoint, long Direction, BOOL &LastXSloped)
{
	long NextPoint = -1;
	double ExtremeSlope = 0.0;
	BOOL ExtremeSet = FALSE;
	double CurSlope;
	long sz = m_Points.GetSize();

	// Loop to find greatest slope from current point
	for (long i = 0; i < sz; i++)
	{
		long TestPoint = m_Points[i];
		if (TestPoint == CurPoint)
			continue;

		if (m_X->GetAt(TestPoint) == m_X->GetAt(CurPoint) && m_Y->GetAt(TestPoint) == m_Y-
>GetAt(CurPoint))
			continue;

		if (!Direction)
		{
			if (m_X->GetAt(TestPoint) < m_X->GetAt(CurPoint))
				continue;
		}
		else
		{
			if (m_X->GetAt(TestPoint) > m_X->GetAt(CurPoint))
				continue;
		}

		CurSlope = ComputeSlope(CurPoint, TestPoint, Direction);
		if (!ExtremeSet || CurSlope > ExtremeSlope)
		{
			if (!LastXSloped || (CurSlope != 0 && CurSlope != MaxSlope && CurSlope != MinSlope))
			{
				ExtremeSet = TRUE;
				ExtremeSlope = CurSlope;
				NextPoint = TestPoint;
```

© PNR and Associates, All rights reserved

```
                }
        }
}

LastXSloped = FALSE;

// If next point is found, check for horizontal and vertical
if (NextPoint > -1)
{
        CurSlope = ComputeSlope(CurPoint, NextPoint, Direction);

        if (!CurSlope)
        {
                // Vertical line
                LastXSloped = TRUE;
                for (long i = 0; i < sz; i++)
                {
                        if (m_Points[i] == CurPoint || m_Points[i] == NextPoint)
                                continue;

                        if (m_Y->GetAt(NextPoint) == m_Y->GetAt(m_Points[i]))
                        {
                                if (m_X->GetAt(NextPoint) > m_X->GetAt(CurPoint))
                                {
                                        if (m_X->GetAt(m_Points[i]) > m_X->GetAt(NextPoint))
                                                NextPoint = m_Points[i];
                                }
                                else
                                {
                                        if (m_X->GetAt(m_Points[i]) < m_X->GetAt(NextPoint))
                                                NextPoint = m_Points[i];
                                }
                        }
                }
        }
        else if (CurSlope == MinSlope || CurSlope == MaxSlope)
        {
                // Horizontal lines
                LastXSloped = TRUE;
                for (long i = 0; i < sz; i++)
                {
                        if (m_Points[i] == CurPoint || m_Points[i] == NextPoint)
                                continue;
```

© PNR and Associates, All rights reserved

```
                        if (m_X->GetAt(NextPoint) == m_X->GetAt(m_Points[i]))
                        {
                              if (m_X->GetAt(NextPoint) > m_X->GetAt(CurPoint))
                              {
                                    if (m_X->GetAt(m_Points[i]) > m_X->GetAt(NextPoint))
                                          NextPoint = m_Points[i];
                              }
                              else
                              {
                                    if (m_X->GetAt(m_Points[i]) < m_X->GetAt(NextPoint))
                                          NextPoint = m_Points[i];
                              }
                        }
                  }
            }
      }

      return NextPoint;
}

void CArea::GetPerimeter(CDblArray &Vertices, CDWordArray &Perimeter)
{
      BOOL LastXSloped = FALSE;
      long sz = m_Points.GetSize();

      long WestMost = m_Points[0];
      for (long i = 1; i < sz; i++) {
            if (m_X->GetAt(m_Points[i]) < m_X->GetAt(WestMost))
                  WestMost = m_Points[i];
            else if (m_X->GetAt(m_Points[i]) == m_X->GetAt(WestMost) && m_Y->GetAt(m_Points[i]) < m_Y-
>GetAt(WestMost))
                  WestMost = m_Points[i];
      }

      long Position=0;
      m_Perimeter.Add(WestMost);
      Perimeter[WestMost] = ++Position;

      long NextPoint;
      long CurPoint = WestMost;

      long Direction = 0;
      for (i = 0; i < sz * 2; i++) {
            NextPoint = GetNextPoint(CurPoint, Direction, LastXSloped);
```

© PNR and Associates, All rights reserved

```
            if (NextPoint < 0) {
                  ++Direction;      // just a duplicate point, not end of search (possible???)
                  continue;
            } else if (m_X->GetAt(CurPoint) == m_X->GetAt(NextPoint) && m_Y->GetAt(CurPoint) > m_Y-
>GetAt(NextPoint)) {
                  ++Direction;
            }

            if (NextPoint == WestMost)
                  break;

            CurPoint = NextPoint;
            m_Perimeter.Add(CurPoint);
            Perimeter[CurPoint] = ++Position;
      }
}


/////////////////////////////////////////////////////////////////////////////
//
//    CArea:  Attribute functions
//
/////////////////////////////////////////////////////////////////////////////

void CArea::GetPerimiterPts(CDWordArray& P)
{
      long i, sz;
      sz = m_Perimeter.GetSize();
      P.SetSize(sz);
      for (i = 0; i < sz; i++)
            P[i] = m_Perimeter[i];
}

// Data.h : header file
//

#ifndef DATA__H
#define DATA__H

#include "Global.h"
#include "output.h"
```

© PNR and Associates, All rights reserved

```
/////////////////////////////////////////////////////////////////////////////
// CData dialog

class CData : public CObject
{
// Construction
public:
      CData();
      ~CData();

// Attributes
protected:
      // Data variables
      CDblArray m_Long;
      CDblArray m_Lat;
      CDblArray m_nWt;              // Array of number of HH at same location
      CStringArray m_BG;                  // Block groups
      CStringArray m_Type;          // Location type (Residential/Business)
      CDblArray m_slb;
      CDblArray m_emp;
      CDblArray m_ctx;

      long m_nCases;          // Number of cases
      double m_nCasesWt;                  // Number of cases weighted
      long m_nCurr;           // Number of block groups read

      BOOL m_bIsOpen;                     // Flag to indicate open data file and Long/Lat allocations
      BOOL m_bEOF;                        // Flag to indicate end of file
      CStdioFile* m_File;                 // Data file pointer
      CStdioFile* m_TmpFile;        // Records read for split group
      CString m_Buffer;             // Read buffer
      CString m_LastRead;                 // Last record read

      CString m_ListFile;                 // List file name
      CStringArray m_List;          // Array of splits to process

// Functions
protected:
      // List routines
      void InitList(LPCSTR listName);
      BOOL ReadList(LPCSTR listName);

      // Data routines
      void Init();
```

© PNR and Associates, All rights reserved

```
	void InitOpen();
	long GetLongField();
	double GetDblField();
	CString GetStrField();

public:
	BOOL Open(LPCSTR, LPCSTR);
	BOOL Read(COutput *OutputFile);
	BOOL Close();

	// Attribute routines
	BOOL IsOpen();
	CDblArray* GetLongitude() {return &m_Long;}
	CDblArray* GetLatitude() {return &m_Lat;}
	double GetLongitude(long index) {return m_Long[index];}
	double GetLatitude(long index) {return m_Lat[index];}
	long GetNCases();
	double GetNCasesWt();
	double GetWt(long index);

	double GetSLB(long index) {return m_slb[index];}
	double GetEMP(long index) {return m_emp[index];}
	double GetCTX(long index) {return m_ctx[index];}

	long GetNDone();
	BOOL GetEOF();
	CString GetGroup(long nGrp);
	CString GetBlockGroup(long nCase);
	CString GetType(long nCase);

	CString GetTempFileName();
};

#endif

// Data.cpp : implementation file

#include "stdafx.h"
#include "Data.h"

CData::CData()
{
	m_File = new CStdioFile();
//	m_TmpFile = new CStdioFile();
```

© PNR and Associates, All rights reserved

```
	m_bIsOpen = FALSE;
	InitOpen();
}

CData::~CData()
{
	if (m_File)
		delete m_File;

//	if (m_TmpFile)
//	{
//		CFile::Remove(m_TmpFile->GetFilePath());
//		delete m_TmpFile;
//	}
}

////////////////////////////////////////////////////////////////////////////////
// CData:  Specific List

void CData::InitList(LPCSTR list)
{
	m_List.RemoveAll();
	m_ListFile = list;
}

BOOL CData::ReadList(LPCSTR list)
{
	InitList(list);

	// If specific split groups, open list file and read split groups.
	if (m_ListFile != "ALL" && m_ListFile != "NONE") {
		// Open and read
		CStdioFile* pf;
		try	{
			pf = new CStdioFile(list, CFile::modeRead);
		} catch (CFileException* e) {
			AfxMessageBox("ERROR: Could not open the split file list that was specified.");
			e->Delete();
			return FALSE;
		}

		// Scan split groups to be processed
		CString str;
```

© PNR and Associates, All rights reserved

```
            while (pf->ReadString(str))
                  m_List.Add(str.Left(BGNUMLEN));

            // Close and return
            pf->Close();
            delete pf;
      }

      return TRUE;
}

////////////////////////////////////////////////////////////////////////////
// CData:  Data File

void CData::Init()
{
      m_nCases = 0;
      m_nCasesWt = 0.;
      m_Long.RemoveAll();
      m_Lat.RemoveAll();
      m_nWt.RemoveAll();
      m_BG.RemoveAll();
      m_Type.RemoveAll();
      m_slb.RemoveAll();
      m_emp.RemoveAll();
      m_ctx.RemoveAll();
}

void CData::InitOpen()
{
      // Close any files left open
      Close();

      // Initialize and allocate members
      m_nCurr = 0;
      m_LastRead = "";
      m_bEOF = FALSE;

      Init();
}

BOOL CData::Open(LPCSTR dataName, LPCSTR listName)
{
      // Initialize one-time-only members
```

© PNR and Associates, All rights reserved

```
	InitOpen();

	// Read split list if specified
	if (!ReadList(listName))
		return FALSE;

	// Open data file
	if (!m_File->Open(dataName, CFile::modeRead | CFile::shareDenyWrite))
	{
		AfxMessageBox("ERROR: The attempt to open the data file has failed.");
		return FALSE;
	}

	m_bIsOpen = TRUE;
	return TRUE;
}

BOOL CData::Read(COutput *OutputFile)
{
	Init();

	// Declare
	CString Split = "";
	BOOL bBegin = FALSE;
	BOOL bEnd = FALSE;

	// Open temporary file
//	m_TmpFile->Open(GetTempFileName(), CFile::modeCreate | CFile::modeWrite);

double vLong, vLat, vWt, vSLB, vEMP, vCTX;
CString vBG, vType;


	// Set Split List if reading all Split
	if (m_ListFile == "NONE")
	{
		m_List.Add("None");

		// Read data
		while(!m_bEOF && !bEnd) {
			if (m_File->ReadString(m_Buffer)) {
				// Assign values to  member variables
				Split = (GetStrField()).Left(BGNUMLEN);
if (Split.GetLength() < 8) {
```

© PNR and Associates, All rights reserved

```
      CString Add = (GetStrField()).Left(BGNUMLEN);
      Split = Split + " " + Add;
}
                        vLong = GetDblField();
                        vLat = GetDblField();
                        vWt = GetDblField();
                        vBG = GetStrField();
                        vType = GetStrField();
                        vSLB = GetDblField();
                        vEMP = GetDblField();
                        vCTX = GetDblField();

                        if (vWt <= 0) {
                              char buff[256];
                              sprintf(buff, "WARNING:  Skipped %s, %.6f, %.6f, %f, %s", (LPCSTR)Split,
vLong, vLat, vWt, (LPCSTR)vType);
                              OutputFile->SaveLog(buff);
                        } else {
                              m_Long.Add(vLong);
                              m_Lat.Add(vLat);
                              m_nWt.Add(vWt);
                              m_BG.Add(vBG);
                              m_Type.Add(vType);
                              m_slb.Add(vSLB);
                              m_emp.Add(vEMP);
                              m_ctx.Add(vCTX);
                        }

                        // Add original buffer to array
//                      m_TmpFile->WriteString(m_Buffer);
//                      m_TmpFile->WriteString(" \n");

                        // Increment number of cases/weighted cases
                        m_nCasesWt += m_nWt[m_nCases++];
//                      if (m_nCases == MAX_CASES) {
//                            OutputFile->SaveLog("\nWireCenter " + Split + "Exceeded 65k\n");
//                            m_bEOF = TRUE;
//                            bEnd = TRUE;
//                      }
                  } else {
                        m_bEOF = TRUE;
                  }
            }
      } else {
```

© PNR and Associates, All rights reserved

```
            m_Buffer = m_LastRead;
            Split = (GetStrField()).Left(BGNUMLEN);

if (Split.GetLength() < 8) {
      CString Add = (GetStrField()).Left(BGNUMLEN);
      Split = Split + " " + Add;
}

            if (m_ListFile == "ALL")
                  if (m_nCurr)
                        m_List.Add(Split);
                  else
                        m_List.Add("SET FIRST");


            // If the last record read was part of the next split group, set first record
            if (Split == m_List[m_nCurr])
            {
                  bBegin = TRUE;

                  vLong = GetDblField();
                  vLat = GetDblField();
                  vWt = GetDblField();
                  vBG = GetStrField();
                  vType = GetStrField();
                  vSLB = GetDblField();
                  vEMP = GetDblField();
                  vCTX = GetDblField();

                  if (vWt <= 0) {
                        char buff[256];
                              sprintf(buff, "WARNING:  Skipped %s, %.6f, %.6f, %f, %s", (LPCSTR)Split,
vLong, vLat, vWt, (LPCSTR)vType);
                        OutputFile->SaveLog(buff);
                  } else {
                        m_Long.Add(vLong);
                        m_Lat.Add(vLat);
                        m_nWt.Add(vWt);
                        m_BG.Add(vBG);
                        m_Type.Add(vType);
                        m_slb.Add(vSLB);
                        m_emp.Add(vEMP);
                        m_ctx.Add(vCTX);
```

© PNR and Associates, All rights reserved

```
//                      m_TmpFile->WriteString(m_LastRead);
//                      m_TmpFile->WriteString(" \n");

                        m_nCasesWt += m_nWt[m_nCases++];
                  }
            }

            // If the last record read was greater than the next split group, skip read
            if (Split > m_List[m_nCurr])
                  bEnd = TRUE;

            while(!bEnd)
            {
                  // Read data
                  if (!m_File->ReadString(m_Buffer))
                        m_bEOF = TRUE;

                  if (m_bEOF)
                        break;

                  m_LastRead = m_Buffer;

                  // Get split and set first split for ALL
                  Split = (GetStrField()).Left(BGNUMLEN);
if (Split.GetLength() < 8) {
      CString Add = (GetStrField()).Left(BGNUMLEN);
      Split = Split + " " + Add;
}
                  if (m_List[m_nCurr] == "SET FIRST")
                        m_List[m_nCurr] = Split;

                  // If correct split group, set flags
                  if (Split == m_List[m_nCurr])
                        bBegin = TRUE;
                  else if (Split > m_List[m_nCurr])
                  {
                        bBegin = FALSE;
                        bEnd = TRUE;
                  }

                  // Fill in data arrays
                  if (bBegin)
                  {
                        vLong = GetDblField();
```

© PNR and Associates, All rights reserved

```
                        vLat = GetDblField();
                        vWt = GetDblField();
                        vBG = GetStrField();
                        vType = GetStrField();
                        vSLB = GetDblField();
                        vEMP = GetDblField();
                        vCTX = GetDblField();

                        if (vWt <= 0) {
                              char buff[256];
                              sprintf(buff, "WARNING:  Skipped %s, %.6f, %.6f, %f, %s", (LPCSTR)Split,
vLong, vLat, vWt, (LPCSTR)vType);
                              OutputFile->SaveLog(buff);
                        } else {
//                            m_Long[index] = GetDblField();
//                            m_Lat[index++] = GetDblField();
//                            m_nWt.Add(GetDblField());
//                            m_BG.Add(GetStrField());
//                            m_Type.Add(GetStrField());
//                            m_slb.Add(GetDblField());
//                            m_emp.Add(GetDblField());
//                            m_ctx.Add(GetDblField());

                              m_Long.Add(vLong);
                              m_Lat.Add(vLat);
                              m_nWt.Add(vWt);
                              m_BG.Add(vBG);
                              m_Type.Add(vType);
                              m_slb.Add(vSLB);
                              m_emp.Add(vEMP);
                              m_ctx.Add(vCTX);

//                            m_TmpFile->WriteString(m_LastRead);
//                            m_TmpFile->WriteString(" \n");

                              m_nCasesWt += m_nWt[m_nCases++];
//                            if (m_nCases == MAX_CASES)
//                            {
//                                  m_nCasesWt -= m_nWt[m_nCases - 1];
//                                  m_nCases--;
//                                  m_bEOF = TRUE;          // Do Not End File but just this pass of WC
//                                  bEnd = TRUE;
//                                  OutputFile->SaveLog("\nWARNING:  WireCenter " + Split + " Exceeded
65k\n");
```

© PNR and Associates, All rights reserved

```
//                                }
                            }
                        }
                    }
            }
        m_nWt.FreeExtra();
        m_BG.FreeExtra();
        m_Type.FreeExtra();
        m_slb.FreeExtra();
        m_emp.FreeExtra();
        m_ctx.FreeExtra();

        // Increment number of groups read
        m_nCurr++;

        // Set end of Split list if list is specified and has been exhausted
        if (!m_bEOF && m_ListFile != "ALL" && m_nCurr == m_List.GetSize())
                m_bEOF = TRUE;

//      m_TmpFile->Close();
        return m_bEOF;
}

BOOL CData::Close()
{
        // Close the data file and reset
        if (m_File && m_bIsOpen)
        {
                m_File->Close();
                m_bIsOpen = FALSE;
                return TRUE;
        }
        return FALSE;
}

CString CData::GetStrField()
{
        CString str;

        m_Buffer.TrimLeft();
        int pos = m_Buffer.FindOneOf(" \t,");
        if (pos > -1)
        {
                if (pos > 0)
```

© PNR and Associates, All rights reserved

```
                str = m_Buffer.Left(pos);
            m_Buffer = m_Buffer.Mid(pos + 1);
        }
        else
        {
            str = m_Buffer;
            m_Buffer = "";
        }

        return str;
}

long CData::GetLongField()
{
        if (m_Buffer == "")
            return -1;

        return atol(GetStrField());
}

double CData::GetDblField()
{
        if (m_Buffer == "")
            return -1.0;

        return atof(GetStrField());
}

/////////////////////////////////////////////////////////////////////////////
//
// CData:  Attribute funcitons
//
/////////////////////////////////////////////////////////////////////////////

BOOL CData::IsOpen()
{
        return m_bIsOpen;
}

long CData::GetNCases()
{
        return m_nCases;
}
```

© PNR and Associates, All rights reserved

```
double CData::GetNCasesWt()
{
	return m_nCasesWt;
}

double CData::GetWt(long index)
{
	return m_nWt[index];
}

long CData::GetNDone()
{
	return m_nCurr;
}

BOOL CData::GetEOF()
{
	return m_bEOF;
}

CString CData::GetGroup(long nGrp)
{
	return m_List[nGrp - 1];
}

CString CData::GetTempFileName()
{
	CString fName = m_File->GetFilePath();
	fName += ".TEMP";
	return fName;
}

CString CData::GetBlockGroup(long nCase)
{
	return m_BG[nCase];
}

CString CData::GetType(long nCase)
{
	return m_Type[nCase];
}


// Distance.h : header file
```

© PNR and Associates, All rights reserved