SEG-Y FORMAT READER FOR PERSONAL COMPUTER

By

NASAR MAHMOOD

COMPUTER CENTRE
Qauid-i-Azam University
Islamabad
OCTOBER 2003

Dedicated To My Beloved
" younger brother" Whose
smiling face is source of my
happiness.

COMPUTER CENTRE
QUAID-I-AZAM UNIVERISITY
ISLAMABAD

FINAL APPROVAL OF PROJECT

This is to certify that we have read the project submitted by Mr. NASAR MAHMOOD and it is our judgment that this project is of sufficient standard to warrant to its acceptance by the Quaid-I-Azam University for the postgraduate diploma in computer sciences.

COMMITTEE

SIR NAZIMMUDDIN ----------
(Supervisor)

DR. GHULAM MAHAMMAD ----------
(Director computer center)

EXTERNAL EXAMINAR ----------

DECLARATION

I declare that this software neither as a whole nor as the part has been copied from any other sources. It is further declared that I have completed my final project of the postgraduate diploma in computer sciences successfully as the result of my own struggle and the research. No portion of this whole work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning. If any part of the project and the write is proved to be copied out or there is any duplication of the code then I will be responsible for the consequences.

Name of student: Nasar Mahmood

CONTENTS

Acknowledgement

All praises for Almighty Allah, the most merciful, and the creator of the universe who blessed me the knowledge and enabled me to complete this dissertation.

I express my profound gratitude to my respectable teacher and supervisor Nazimuddin Sahib for his encouragement, great cooperation, invaluable guidance, healthy and constructive criticism, and expert advices during the project.

I owe a lot of favors to Dr. Ghulam Mahammad, director of the computer center, who was always helpful. I would like to express my gratitude to all my teachers in the center who taught and made me potentially capable of all this what I am today.

I can't express my feeling for my father, my mother who are the real source of my encouragement, especially my Father, whose efforts and emotional attachment made me capable of all this what I am today.

I am extremely obliged to all my class fellows and friends for their cooperation during my stay in university especially Ashfaq, Waheed Gujjer for their wholehearted cooperation and sharing with me the unforgettable and everlasting memories which we had in the university.

NASIR MAHMOOD

Abstract

The aim is to develop software, which could read the Seg-y format and translate it into a format that is readable by a personal computer.

The Seg-y format is used on mainframe computers and work stations, these computers use EBCDIC and IBM 32-bit floating point formats and operating systems such as Unix or Linux, while a personal computer available to the students neither supports the two above mentioned format nor are Unix or Linux operating systems common on a personal computer.

The software breaks the barrier and allows the user to work with the data on a personal computer.

Seg-y reader has been developed in VC++6.0 the software open the Seg-y format file and allows the user to view the EBCDIC header, binary header, the trace header and the trace data and the plot for the individual traces, further it allows the user to save the file ad a word document, an excel spreadsheet, or a text file. It has help files that guide the user in operating the software and other files that describe the Seg-y format in detail.

Chapter # 1

Introduction To The Project

Introduction to project

Seismic waves are the messenger of the subsurface exploration targets. In seismic methods sound waves are generated with the help of dynamite. These waves travel in the subsurface and reflect from different boundaries, reflected and refracted energy causes the ground vibration; very sensitive instruments record this vibration. These measurements are actually the amplitude of the ground vibration, this vibration is sampled at 2ms intervals and stored on the magnetic tapes or on seismic reels in the digital form and then processed on mainframe computers and the workstation for the further interpretation work .SEG the society of the exploration geophysics has made the standards for particular formats that are used for the the storage of the data, these formats are work station and mainframe supported. This software is Pc-based reader for a particular format called Seg –y format.

DESCRIPTION OF REEL

The seismic reel is divided into the reel identification header and the trace data block

Reel Identification Header

The reel identification header consists of 3600 bytes and is divided into two parts:

(1) The early image EBCDIC block (3200 bytes-40 cards equivalent) followed by an IBG.

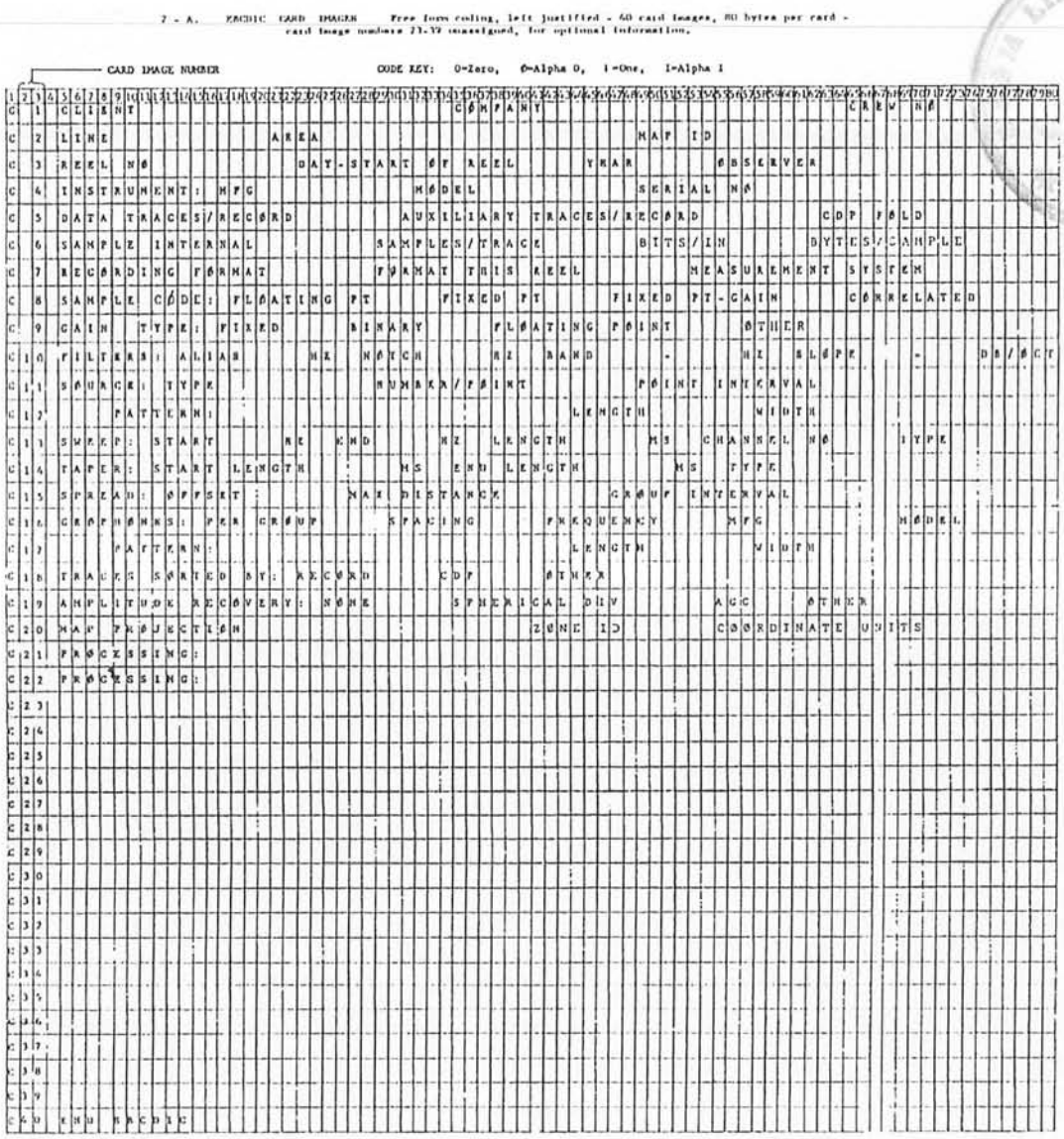


FIG. 2A. Reel identification header. Part 1, the EBCDIC card image block.

3-10. BINARY CARD. - Right justified

Byte Numbers		Description
3201-3204		Job identification number.
3205-3208	*	Line number (only one line per reel).
3209-3212	*	Reel number.
3213-3214	*	Number of data traces per record (includes dummy and zero traces inserted to fill out the record or common depth point).
3215-3216	*	Number of auxillilary traces per record (includes sweep, timing, gain, sync and all other non-data traces).
3217-3218	*	Sample interval in μs. (for this reel of data). } Designated in microseconds to accommodate
3219-3220		Sample interval in μs. (for original field recording). } sample intervals less than one millisecond.
3221-3222	*	Number of samples per data trace (for this reel of data).
3223-3224		Number of samples per data trace (for original field recording).
3225-3226	*	Data sample format code: 1 = floating point (4 bytes) 3 = fixed point (2 bytes) 2 = fixed point (4 bytes) 4 = fixed point w/gain code (4 bytes) Auxiliary traces use the same number of bytes per sample.
3227-3228	*	CDP fold (expected number of data traces per CDP ensemble).
3229-3230		Trace sorting code: 1 = as recorded (no sorting) 3 = single fold continuous profile 2 = CDP ensemble 4 = horizontally stacked
3231-3232		Vertical sum code: 1 = no sum, 2 = two sum, - - -, N = N sum (N = 32,767)
3233-3234		Sweep frequency at start.
3235-3236		Sweep frequency at end.
3237-3238		Sweep length (ms.)
3239-3240		Sweep type code: 1 = linear 3 = exponential 2 = parabolic 4 = other
3241-3242		Trace number of sweep channel.
3243-3244		Sweep trace taper length in ms. at start if tapered (the taper starts at zero time and is effective for this length).
3245-3246		Sweep trace taper length in ms. at end (the ending taper starts at sweep length minus the taper length at end).
3247-3248		Taper type: 1 = linear 3 = other 2 = cos^2
3249-3250		Correlated data traces: 1 = no 2 = yes
3251-3252		Binary gain recovered: 1 = yes 2 = no
3253-3254		Amplitude recovery method: 1 = none 3 = AGC 2 = spherical divergence 4 = other
3255-3256	*	Measurement system: 1 = meters 2 = feet
3257-3258		Impulse Signal Polarity: 1 = Increase in pressure or upward geophone case movement gives negative number on tape. 2 = Increase in pressure or upward geophone case movement gives positive number on tape.
3259-3260		Vibratory Polarity Code: Seismic Signal Lags Pilot Signal By: 1 = 337.5° to 22.5° 2 = 22.5° to 67.5° 3 = 67.5° to 112.5° 4 = 112.5° to 157.5° 5 = 157.5° to 202.5° 6 = 202.5° to 247.5° 7 = 247.5° to 292.5° 8 = 292.5° to 337.5°
3261-3600		Unassigned - for optional information.

* Strongly recommended that this information always be recorded.

FIG. 2B. Reel identification header. Part 2, the binary coded block.

(2) The binary coded block (400 bytes) followed by an IBG.

The EBCDIC part of the reel header describes the data from a line of shot points in a fixed specified format consisting of 40 card images with each image contain 80 bytes. All unused card image character is EBCDIC blank. Card image numbers 23 through 39 are unassigned for optional use. Each card image should contain the character "c" in the first card column. Each 80 bytes would yield one line of format print to produce the form shown in figure 2-A

(3) The binary coded section of the real header consists of 400 bytes of information common to the seismic data on the related reel as shown in figure 2-B. there are 60 bytes assigned; 340 are unassigned for optional use.

There are certain bytes of information that may not apply to a particular recording or processing procedure. It is strongly recommended that bytes designated with an asterisk (*) in figures 2-B and 3-E always contain the required information. The data in the reel identification header could be printed and edited prior to the actual input of seismic data for processing.

A complete header listing of both the EBCDIC and binary parts would accompany an exchange tape and serve as a table of contents and summary of specification for that reel of seismic data. No more than one in of seismic data is permitted on any reel. Additional reels would be used for long lines, and each reel must start with a reel identification header.

Description of the trace data block

Each trace data block consists of a fixed 240-bytes trace identification header and the seismic trace data. Each trace data block is separated from the next by an IBG. The trace header is written in the binary code and detail in the figure 3-E.

The trace data sample can be written in one of four data sample formats described in figures 3-A 3-B, 3-C, and 3-D

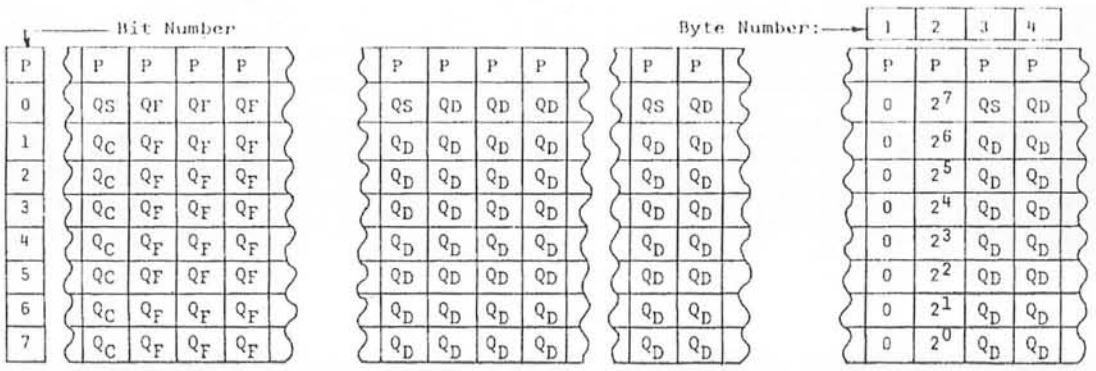
. The trace data format for each reel is identified in bytes 3225 –3226 of reel identification header. Only one data sample format is permitted with in each reel.

Figure 3-a details a 32-bit floating point formatted in which each data value of a seismic channel is recorded in four successive bytes, in IBM compatible floating point notation.

The four bytes from a 32-bit word consisting of the sign bit Qs, a seven-bit characteristic Qc, and a 24-bit fraction Qf. Qs indicates signal polarity and is a one for a negative value. Qc signifies a power of 16 expresses in excess 64 binary notation

indicates signal polarity and is a one for a negative value. Qc signifies a power of 16 expresses in excess 64 binary notation allowing both negative and positive powers of 16 to be represented by a

TRACE DATA SAMPLE FORMATS



NOTE: Least significant bit is always in bit position 7 of byte 4 (or byte 2 for 3-C).

Q_S = Sign bit

Q_C = Characteristic

Q_F = Fraction

Q_D = Data bits

FIG. 3A-D. Trace data block. Four data sample options.

true number. Qr is a six hexadecimal digit (24 binary bit) number with a radix point to the left of the most significant digit. The data value represented by a floating-point number is Qs * $16^{(Qc-64)}$ *Qf.

Figure 3-B details a 32-bit, fixed-point format and each data value of a seismic channel is recorded in four successive bytes. This format consists of a sign bit Qs (one represents negative) and 31 data bits Qd with a radix point at the right of the least significant digit.

Figure 3-c represented a 16-bit, fixed-point format, and each data value of a seismic channel is recorded in two successive bytes. This format is similar to figure 3-b except there are 15 data bits Qd.

Figure 3-D represents a 32-bit, fixed-point format with gain values. The first byte of this format is all zeros. The second byte provides eight available gain bits 2^0 through 2^7 the last two bytes are identical to figure 3-C

In all four data formats, the channel or trace data should represent the absolute input voltage at the recording instrument. The 32-bit, floating point field format defined as the Seg C comprehends the input voltage level. The fixed point formats 3-B and 3-C require a trace weighting factor (trace identification header, bytes 169-170), defined as 2^{-n} volts for the least

significant bit, to comprehend the absolute, the byte of amplitude recovery can be described in the appropriate reel identification header sections, and the algorithm described in the unassigned portions.

TRACE IDENTIFICATION HEADER

Byte Numbers	Description
1 - 4	* Trace sequence number within line--numbers continue to increase if additional reels are required on same line.
5 - 8	Trace sequence number within reel--each reel starts with trace number one.
9 -12	* Original field record number.
13-16	* Trace number within the original field record.
17-20	Energy source point number--used when more than one record occurs at the same effective surface location.
21-24	CDP ensemble number
25-28	Trace number within the CDP ensemble--each ensemble starts with trace number one.
29-30	* Trace identification code: 1 = seismic data, 4 = time break, 7 = timing 2 = dead, 5 = uphole, 8 = water break 3 = dummy, 6 = sweep, 9----, N = optional use (N = 32,767)
31-32	Number of vertically summed traces yielding this trace. (1 is one trace, 2 is two summed traces, etc.)
33-34	Number of horizontally stacked traces yielding this trace. (1 is one trace, 2 is two stacked traces, etc.)
35-36	Data use: 1 = production. 2 = test.
37-40	Distance from source point to receiver group (negative if opposite to direction in which line is shot).
41-44	Receiver group elevation; all elevations above sea level are positive and below sea level are negative.
45-48	Surface elevation at source.
49-52	Source depth below surface (a positive number).
53-56	Datum elevation at receiver group.
57-60	Datum elevation at source.
61-64	Water depth at source.
65-68	Water depth at group.
69-70	Scaler to be applied to all elevations and depths specified in bytes 41-68 to give the real value. Scaler = 1, ±10, ±100, ±1000, or ±10,000. If positive, scaler is used as a multiplier; if negative, scaler is used as a divisor.
71-72	Scaler to be applied to all coordinates specified in bytes 73-88 to give the real value. Scaler = 1, ±10, ±100, ±1000, or ±10,000. If positive, scaler is used as a multiplier; if negative, scaler is used as divisor.
73-76	Source coordinate - X.
77-80	Source coordinate - Y.
81-84	Group coordinate - X.
85-88	Group coordinate - Y.
	If the coordinate units are in seconds of arc, the X values represent longitude and the Y values latitude. A positive value designates the number of seconds east of Greenwich Meridian or north of the equator and a negative value designates the number of seconds south or west.
89-90	Coordinate units: 1 = length (meters or feet). 2 = seconds of arc.
91-92	Weathering velocity.
93-94	Subweathering velocity.
95-96	Uphole time at source.
97-98	Uphole time at group.

FIG. 3E. Trace identification header written in binary code.

Byte Numbers	Description
99-100	Source static correction.
101-102	Group static correction.
103-104	Total static applied. (Zero if no static has been applied.)
105-106	Lag time A. Time in ms. between end of 240-byte trace identification header and time break. Positive if time break occurs after end of header, negative if time break occurs before end of header. Time break is defined as the initiation pulse which may be recorded on an auxiliary trace or as otherwise specified by the recording system.
107-108	Lag Time B. Time in ms. between time break and the initiation time of the energy source. May be positive or negative.
109-110	Delay recording time. Time in ms. between initiation time of energy source and time when recording of data samples begins. (For deep water work if data recording does not start at zero time.)
111-112	Mute time--start.
113-114	Mute time--end.
115-116	* Number of samples in this trace.
117-118	* Sample interval in μs for this trace.
119-120	Gain type of field instruments: 1 = fixed. 2 = binary. 3 = floating point. 4 --- N = optional use.
121-122	Instrument gain constant.
123-124	Instrument early or initial gain (db).
125-126	Correlated: 1 = no. 2 = yes.
127-128	Sweep frequency at start.
129-130	Sweep frequency at end.
131-132	Sweep length in ms.
133-134	Sweep type: 1 = linear. 2 = parabolic. 3 = exponential. 4 = other.
135-136	Sweep trace taper length at start in ms.
137-138	Sweep trace taper length at end in ms.
139-140	Taper type: 1 = linear. 2 = $\cos^2$. 3 = other.
141-142	Alias filter frequency, if used.
143-144	Alias filter slope
145-146	Notch filter frequency, if used.
147-148	Notch filter slope.
149-150	Low cut frequency, if used.
151-152	High cut frequency, if used.
153-154	Low cut slope
155-156	High cut slope
157-158	Year data recorded.
159-160	Day of year.
161-162	Hour of day (24 hour clock)
163-164	Minute of hour.
165-166	Second of minute.
167-168	Time basis code: 1 = local. 2 = GMT. 3 = other.
169-170	Trace weighting factor--defined as 2^{-N} volts for the least significant bit. (N = 0, 1, ... 32, 767.)
171-172	Geophone group number of roll switch position one.
173-174	Geophone group number of trace number one within original field record.
175-176	Geophone group number of last trace within original field record.
177-178	Gap size (total number of groups dropped).
179-180	Overtravel associated with taper at beginning or end of line: 1 = down (or behind). 2 = up (or ahead).
181-240	Unassigned--for optional information.

* Strongly recommended that this information always be recorded.

FIG. 3E. Trace identification header written in binary code (cont.)

Chapter # 2

Seg-Y Format Reader

Seg-y format Reader

The aim is to develop software, which could read the Seg-y format and translate it into a format that is readable by a personal computer. The Seg-y format is used on main frame computers and work stations, these computers use EBCDIC and IBM 32-bit floating point formats and operating systems such as UNIX or LINUX, while a personal computer available to the students neither supports the two above mentioned formats nor UNIX or LINUX operating systems common on a personal computer.

The software breaks the barrier and allows the user to work with the data on a personal computer.

Seg-y reader has been developed in VC++6.0 the software open the Seg-y format file and allows the user to view the EBCDIC header, BINARY header, the trace header and the trace data and the plot for the individual traces, further it allows the user to save the file as a word document, an excel spreadsheet, or a text file. It has help files that guide the user in operating the software and other files that describe the Seg-y format in detail.

Seg-y reader demo run

The opening page of the software is shown below (fig 1.1): -

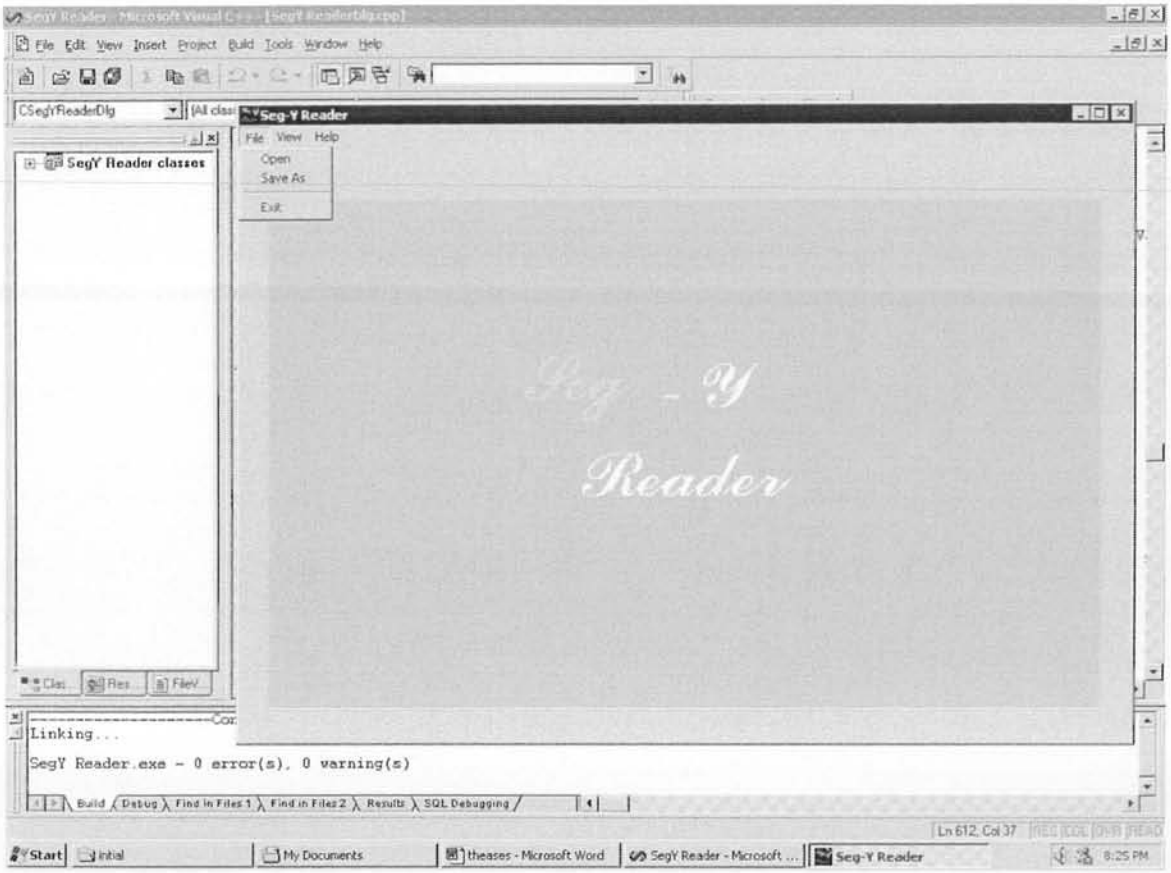
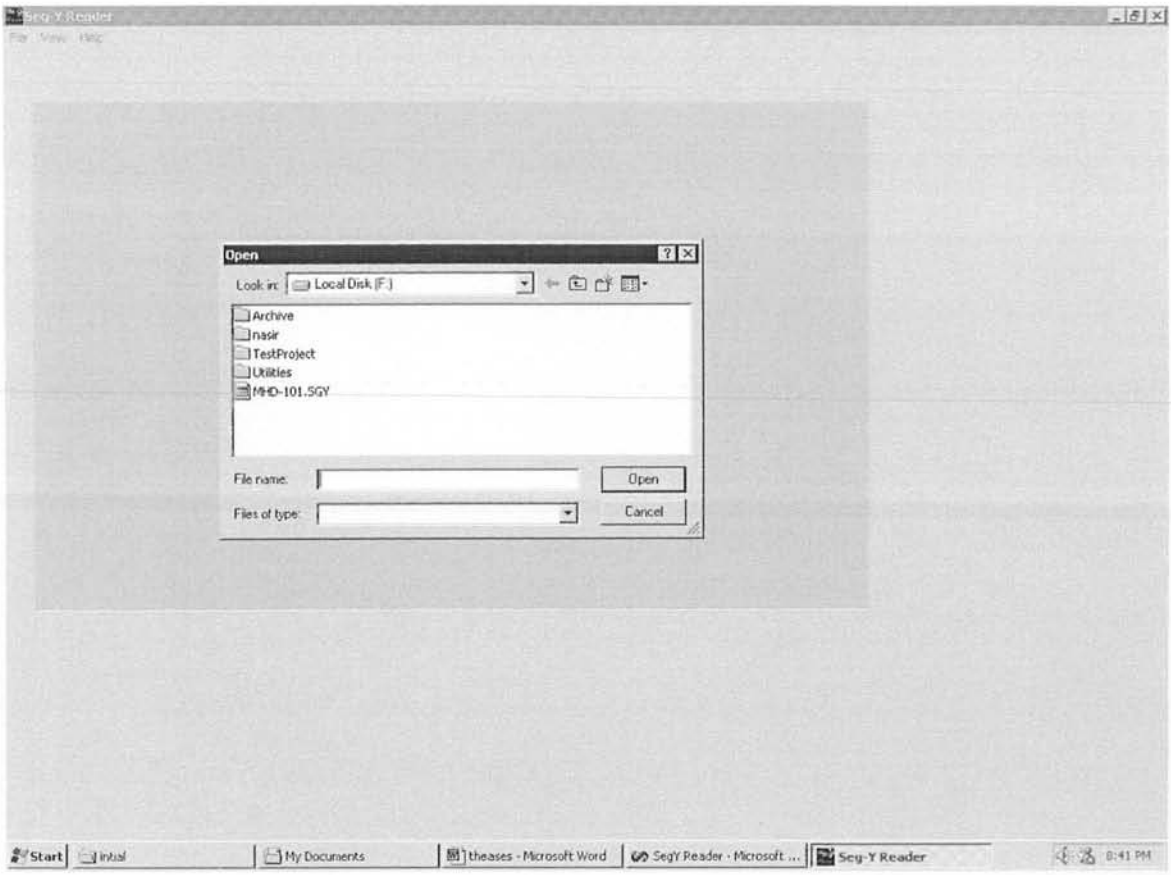
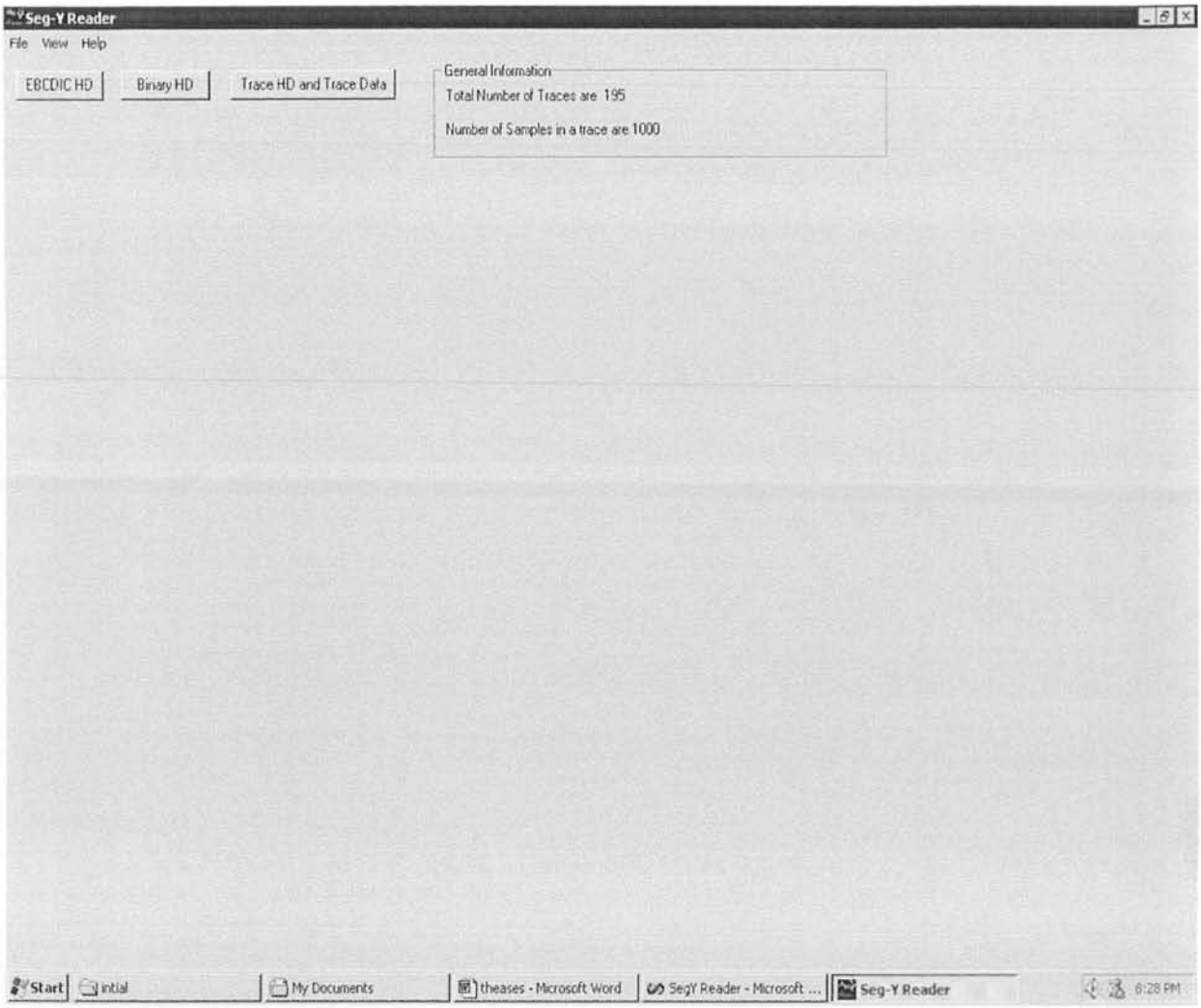


Fig. 1.1 Seg-y reader's opening page

From the file menu click open the file open dialog box appears, browse to the file to be opened and click open, the new view of the reader appears click on either of the three buttons to view the EBCDIC header, the BINARY header or the TRACE header and the trace data (fig. 1.1)



Open dialog box Fig(1.2)



fig(1.3)

The two headers along with the trace data and the trace plot are shown below.

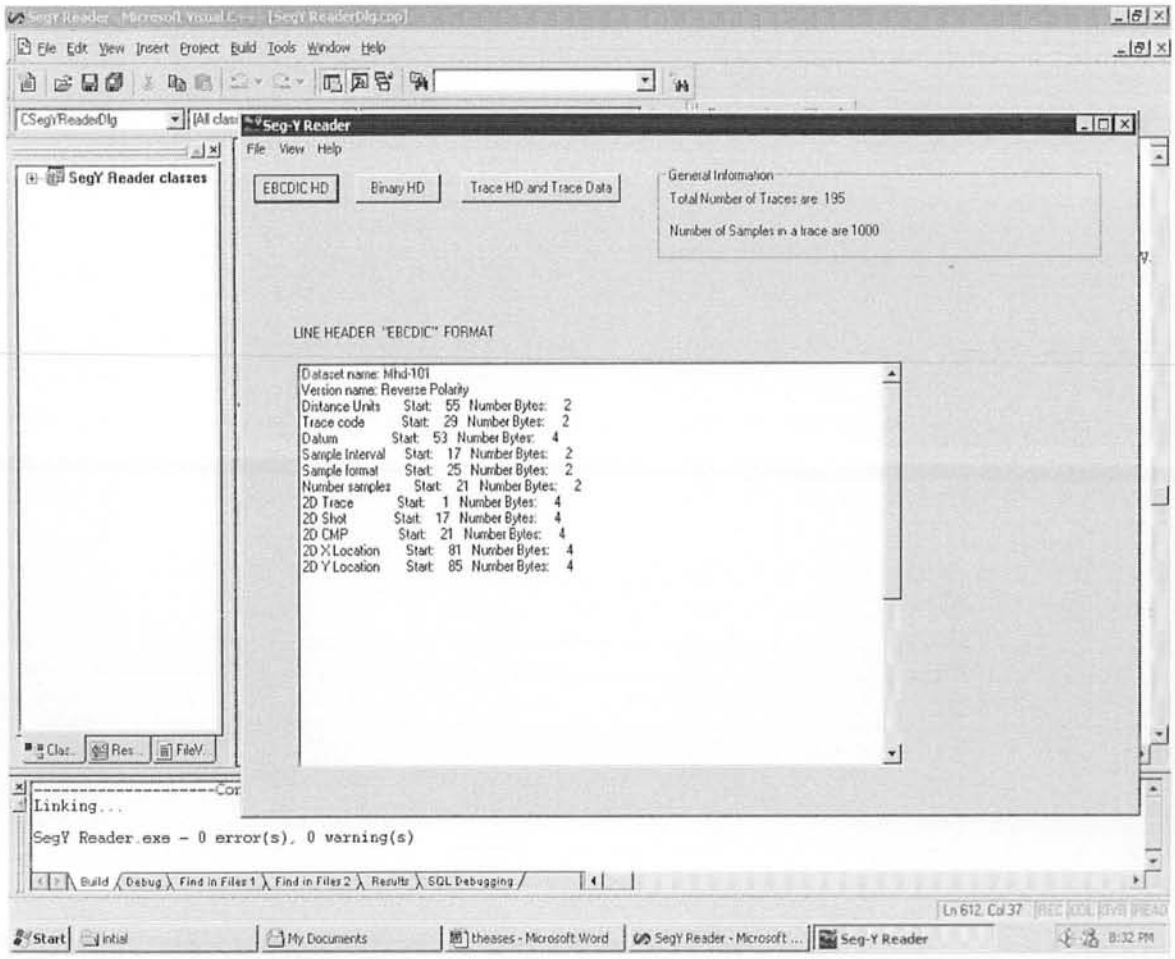


Fig. 1.4 The EBCDIC header.

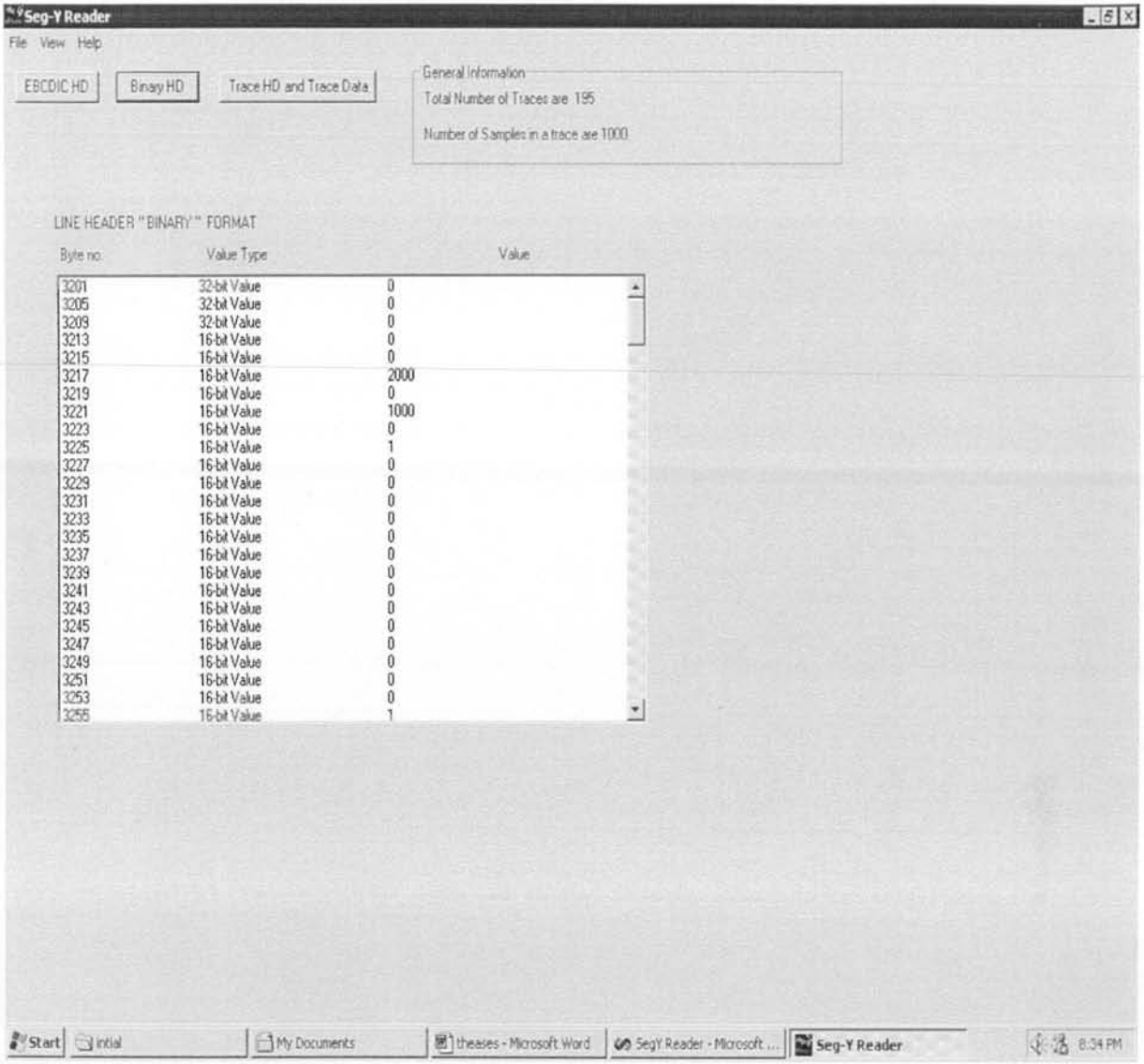


Fig. 1.5 The BINARY header.

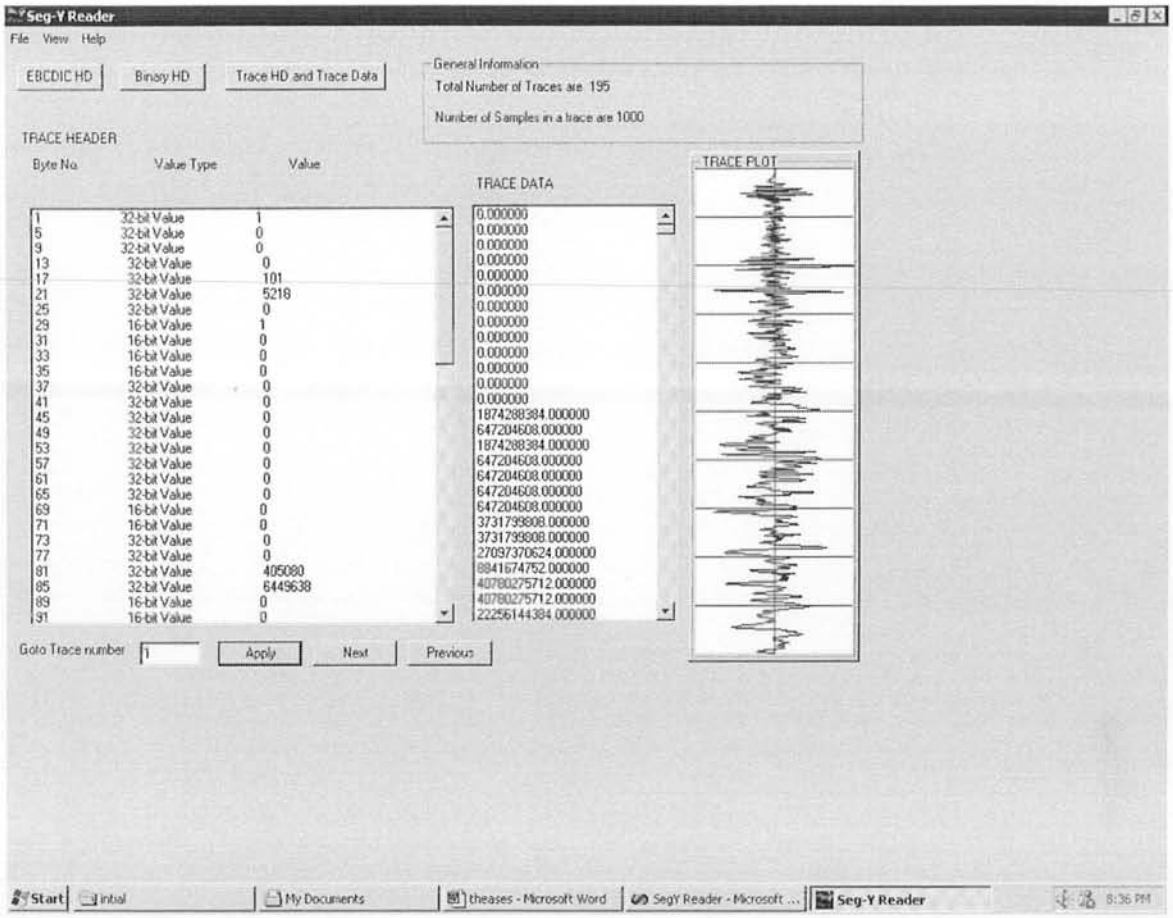
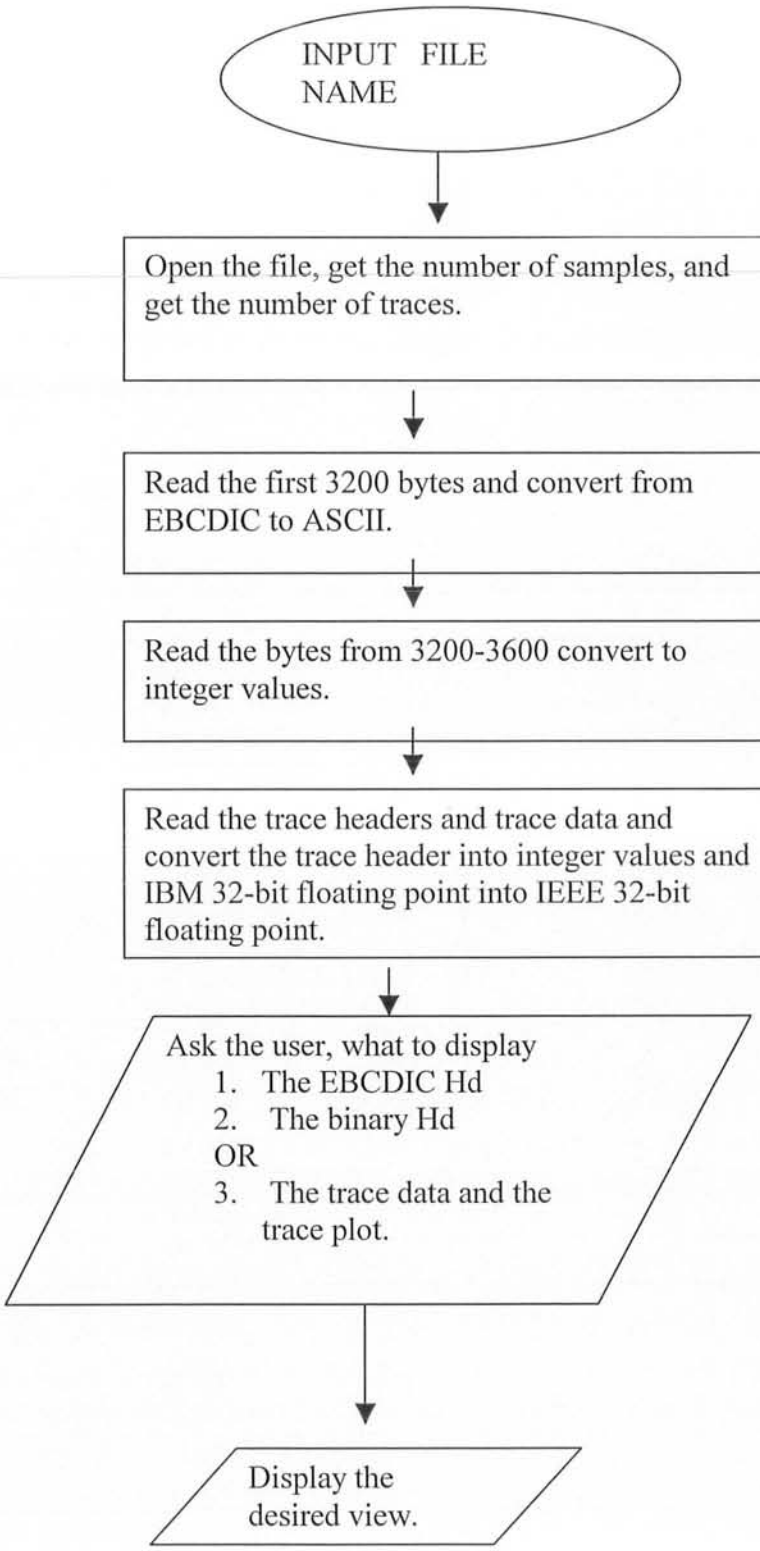


Fig. (1.6) Trace data and Trace header with the plot.

Chapter # 3

Flowchart
&
Source Code

Seg-y reader flow chart



```
// SegY ReaderDlg.cpp : implementation file
//

#include "stdafx.h"
#include "SegY Reader.h"
#include "SegY ReaderDlg.h"
#include "math.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSegYReaderDlg dialog

CSegYReaderDlg::CSegYReaderDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSegYReaderDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSegYReaderDlg)
	m_TotalTraces = _T("");
	m_TraceNo = 1;
	m_NoOfSamples = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CSegYReaderDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSegYReaderDlg)
	DDX_Control(pDX, IDC_LISTBOX_TRACE, m_TraceData);
	DDX_Control(pDX, IDC_LISTBOX_TRACEHD, m_TraceHd);
	DDX_Control(pDX, IDC_LISTBOX_BINARY, m_binary);
	DDX_Control(pDX, IDC_LISTBOX_EBCDIC, m_ebcdic);
	DDX_Text(pDX, IDC_TOTAL_TRACES, m_TotalTraces);
	DDX_Text(pDX, IDC_EDIT_TRACE_NO, m_TraceNo);
	DDX_Text(pDX, IDC_SAMPLE_NO, m_NoOfSamples);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSegYReaderDlg, CDialog)
	//{{AFX_MSG_MAP(CSegYReaderDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
```

```
	ON_WM_QUERYDRAGICON()
	ON_COMMAND(ID_OPEN, OnOpen)
	ON_BN_CLICKED(IDC_APPLY, OnApply)
	ON_COMMAND(ID_EXIT, OnExit)
	ON_COMMAND(ID_ABOUT, OnAbout)
	ON_COMMAND(ID_TRACE_VIEW, OnTraceView)
	ON_COMMAND(ID_SAVE_AS, OnSaveAs)
	ON_BN_CLICKED(IDC_BNEXT, OnBnext)
	ON_BN_CLICKED(IDC_BPREVIOUS, OnBprevious)
	ON_BN_CLICKED(IDC_BEBCDIC, OnBebcdic)
	ON_BN_CLICKED(IDC_BBINARY, OnBbinary)
	ON_BN_CLICKED(IDC_BTRACE, OnBtrace)
	ON_COMMAND(ID_EBCDIC_HLP, OnEbcdicHlp)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSegYReaderDlg message handlers

BOOL CSegYReaderDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
```

```
	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CSegYReaderDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSegYReaderDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
```

```
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSegYReaderDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CSegYReaderDlg::OnOpen()
{
	CFileDialog fdlg(true);//file open dialog box
		if(fdlg.DoModal ()==IDOK)
		{
			filename= fdlg.GetPathName ();
			//file is opened with name "SegFile"
			SegFile.Open (filename,CFile::modeRead,NULL);
			m_ebcdic.ResetContent ();
			m_binary.ResetContent ();
			m_TraceData.ResetContent ();
			m_TraceHd.ResetContent ();
			////////////////////////////////////////
			/////for reading the EBCDIC portion///////////////
			////////////////////////////////////////
			unsigned char pEbcBuf[3200];//declaring buffer for EBCDIC
portion
			SegFile.Read (pEbcBuf,3199);//reading from the file
			int i=0,c=0;//formatting the strings
			for (i=0;i<3200;i++)
				{
					value=pEbcBuf[i];
					c=c+1;

	if((c==80)||(c==160)||(c==240)||(c==320)||(c==400)||(c==480)||(c==560)||(c==640
)||(c==720)||(c==800)||(c==880)||(c==960)||(c==1040)||(c==1120)||(c==1200)||(c==1280)||
(c==1360)||(c==1440)||(c==1520)||(c==1600)||(c==1680)||(c==1760)||(c==1840)||(c==192
0)||(c==2000)||(c==2080)||(c==2160)||(c==2240)||(c==2320)||(c==2400)||(c==2480)||(c==
2560)||(c==2640)||(c==2720)||(c==2800)||(c==2880)||(c==2960)||(c==3040)||(c==3120)||(c
==3200))
						{
						AddStrToEbcListBox(str);
						str="";
						}
					else
						{
```

```
                                        //callling the function to convert EBCDIC to
ASCII
                                        ascii=("%lc",EbcdicToAscii(value));
                                        str+=ascii;
                                        }

                        }
        ////////////////////////////////////////////
                        ////////for reading the binary portion///////
                ////////////////////////////////////////////
                        SegFile.Seek (3200,CFile::begin);
                        CurPos=SegFile.GetPosition ();
                        for(i=0;i<3;i++)
                                {
                                FourByteBinaryReader(CurPos);
                                CurPos=SegFile.GetPosition ();
                                }
                        for(i=0;i<194;i++)
                                {
                                TwoByteBinaryReader(CurPos);
                                CurPos=SegFile.GetPosition ();
                                }
////////////////////////////////////////////////////////
                        ////////getting trace positions
                        //and storing them in the Array
                        SegFile.Seek (3220,CFile::begin);//getting the
                        unsigned char buff[3];//number fo samples
                        SegFile.Read (buff,3);//per trace
                        SampNo=buff[0]*256+buff[1];
                        if(SampNo==0)
                                {
                                SegFile.Seek (3222,CFile::begin);
                                SegFile.Read (buff,3);
                                SampNo=buff[0]*256+buff[1];
                                }
                        ////////////////////////////////////
                        TracePosArray.RemoveAll ();///preparing the array
                        TracePosArray.SetSize(0,1);
                        DWORD FileLength=SegFile.GetLength ();
                        SegFile.Seek (3600,CFile::begin);
                        i=0;
                        unsigned long int TracePos;
                        TracePos=3600;
                        TracePosArray.Add (TracePos);
                        DWORD test=FileLength-((SampNo*4)+240);
                        while(TracePos<test)
```

```
                                {
                                TracePos=((SampNo*4)+240)+TracePos;
                                TracePosArray.Add (TracePos);
                                }
        ////////////////////////////////////////////////////
                        TracePosArray.FreeExtra();
                        TotTraces=TracePosArray.GetSize ();
                        GetDlgItem(IDC_GEN_INF)->ShowWindow(true);
                        m_TotalTraces.Format ("Total Number of Traces are
%d",TotTraces);
                        m_NoOfSamples.Format ("Number of Samples in a trace are
%Ld",SampNo);
                        UpdateData(false);



//////////////Closing the file//////////////
                SegFile.Close ();
///turning the controls vsible
                GetDlgItem(IDC_BEBCDIC)->ShowWindow(true);
                GetDlgItem(IDC_BBINARY)->ShowWindow(true);
                GetDlgItem(IDC_BTRACE)->ShowWindow(true);
                GetDlgItem(IDC_BMP_SEGY)->ShowWindow(false);
                        }
}


//////////////EBCDIC to ASCII converter////////////////////
unsigned char CSegYReaderDlg::EbcdicToAscii(unsigned char val)
{
switch (val)
        {
        case 193:
                val=65;//A
                break;
        case 194:
                val=66;//B
                break;
        case 195:
                val=67;//C
                break;
        case 196:
                val=68;//D
                break;
        case 197:
                val=69;//E
```

```
            break;
case 198:
            val=70;//F
            break;
case 199:
            val=71;//G
            break;
case 200:
            val=72;//H
            break;
case 201:
            val=73 ;//I
            break;
case 209:
            val=74;//J
            break;
case 210 :
            val=75 ;//K
            break;
case 211:
            val=76 ;//L
            break;
case 212:
            val=77 ;//M
            break;
case 213:
            val=78 ;//N
            break;
case 214:
            val=79 ;//O
            break;
case 215:
            val=80 ;//P
            break;
case 216:
            val=81 ;//Q
            break;
case 217:
            val=82 ;//R
            break;
case 226:
            val=83 ;//S
            break;
case 227:
            val=84 ;//T
            break;
```

```
case 228:
        val=85 ;//U
        break;
case 229:
        val=86 ;//V
        break;
case 230:
        val=87 ;//W
        break;
case 231:
        val=88 ;//X
        break;
case 232:
        val=89 ;//Y
        break;
case 233:
        val=90 ;//Z
        break;
case 129:
        val=97 ;//a
        break;
case 130:
        val=98 ;//b
        break;
case 131:
        val=99 ;//c
        break;
case 132:
        val= 100;//d
        break;
case 133:
        val=101 ;//e
        break;
case 134:
        val=102 ;//f
        break;
case 135:
        val=103 ;//g
        break;
case 136:
        val=104 ;//h
        break;
case 137:
        val=105 ;//i
        break;
case 145:
```

```
            val=106 ;//j
            break;
case 146:
            val=107 ;//k
            break;
case 147:
            val=108 ;//l
            break;
case 148:
            val=109 ;//m
            break;
case 149:
            val=110 ;//n
            break;
case 150:
            val=111 ;//o
            break;
case 151:
            val=112 ;//p
            break;
case 152:
            val=113 ;//q
            break;
case 153:
            val=114 ;//r
            break;
case 162:
            val=115 ;//s
            break;
case 163:
            val=116 ;//t
            break;
case 164:
            val=117 ;//u
            break;
case 165:
            val=118 ;//v
            break;
case 166:
            val=119 ;//w
            break;
case 167:
            val=120 ;//x
            break;
case 168:
            val=121 ;//y
```

```
            break;
case 169:
            val=122 ;//z
            break;
case 240:
            val=48 ;//0
            break;
case 241:
            val=49 ;//1
            break;
case 242:
            val=50 ;//2
            break;
case 243:
            val=51 ;//3
            break;
case 244:
            val=52 ;//4
            break;
case 245:
            val=53 ;//5
            break;
case 246:
            val=54 ;//6
            break;
case 247:
            val=55 ;//7
            break;
case 248:
            val=56 ;//8
            break;
case 249:
            val=57 ;//9
            break;
case 109:
            val=45;//"-"
            break;
case 122:
            val=58;//":"
            break;
case 97:
            val=47;//"/"
            break;
case 75:
            val=46;//"."
            break;
```

```
        default :
                val=32;//" "
        }
return val;
}

void CSegYReaderDlg::AddStrToEbcListBox(CString)
{
m_ebcdic.AddString (str);
}

unsigned long int CSegYReaderDlg::FourByteBinaryReader(long int CurPos)
{
//declaring the buffer
unsigned char buf[4];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,4);
long int Value= buf[0]*(8388608)+buf[1]*(65536)+buf[2]*(256)+buf[3];
str.Format ("%Ld                    32-bit Value
%Ld",(CurPos+1),Value);
AddStrToBinHd(str);
return CurPos;
}

void CSegYReaderDlg::AddStrToBinHd(CString str)
{
m_binary.AddString (str);
}

unsigned long int CSegYReaderDlg::TwoByteBinaryReader(unsigned long CurPos)
{
//declaring the buffer
unsigned char buf[3];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,2);
long int Value= buf[0]*(256)+buf[1];
str.Format ("%Ld                    16-bit Value
%Ld",(CurPos+1),Value);
AddStrToBinHd(str);
return CurPos;
}


void CSegYReaderDlg::OnApply()
{
SegFile.Open (filename,CFile::modeRead,NULL);
```

```
m_TraceHd.ResetContent ();
m_TraceData.ResetContent ();
UpdateData(true);
TraceNo=m_TraceNo-1;
TraceLoct=TracePosArray.GetAt (TraceNo);
SegFile.Seek (TraceLoct,CFile::begin);
CurPos=SegFile.GetPosition ();
int i;
for(i=0;i<7;i++)
        {
        THdFourByteBinaryReader(CurPos);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<4;i++)
        {
        THdTwoByteBinaryReader(CurPos);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<8;i++)
        {
        THdFourByteBinaryReader(CurPos);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<2;i++)
        {
        THdTwoByteBinaryReader(CurPos);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<4;i++)
        {
        THdFourByteBinaryReader(CurPos);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<46;i++)
        {
        THdTwoByteBinaryReader(CurPos);
        CurPos=SegFile.GetPosition ();
        }
SegFile.Seek ((TraceLoct+240),CFile::begin);
CurPos=SegFile.GetPosition ();
for(i=0;i<SampNo;i++)
        {
        FlPntRd(CurPos);
        CurPos=SegFile.GetPosition ();
        }
SegFile.Close ();
```

```
OnTraceView();
}

unsigned long int CSegYReaderDlg::THdFourByteBinaryReader(unsigned long CurPos)
{
//declaring the buffer
unsigned char buf[4];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,4);
long int Value= buf[0]*(8388608)+buf[1]*(65536)+buf[2]*(256)+buf[3];
str.Format ("%Ld                32-bit Value                %Ld",(CurPos-
TraceLoct+1),Value);
AddStrToBinTraceHd(str);
return CurPos;
}

void CSegYReaderDlg::AddStrToBinTraceHd(CString str)
{
m_TraceHd.AddString (str);
}


unsigned long int CSegYReaderDlg::THdTwoByteBinaryReader(unsigned long)
{
//declaring the buffer
unsigned char buf[3];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,2);
long int Value= buf[0]*(256)+buf[1];
str.Format ("%Ld                16-bit Value                %Ld",(CurPos-
TraceLoct+1),Value);
AddStrToBinTraceHd(str);
return CurPos;
}

unsigned long int CSegYReaderDlg::FlPntRd(unsigned long CurPos)
{

unsigned char flbuf[4]; ///declaring the buffer which will have unconverted number
SegFile.Read (flbuf,4); //Reading from the file
unsigned char A=flbuf[0],B,C,D;
unsigned char sign = flbuf[0] & 0x80; //Anding to get the value of sign bit
unsigned char Exp =  flbuf[0] & 0x7F; //Anding to get the value of Exponent

if(sign == 1)//getting the sign
        sign=-1;//If sign bit is 1 then the number is negative
```

```
else
        sign=1; //else the number is positive

double charac;
charac=pow(16,Exp - 64);//using the formula

float fraction=flbuf[3]*65536+flbuf[2]*256+flbuf[1]; //Calculating the fraction. This line
can be made
//more efficient by using bit shift operators
float FloatNo = (int)sign*(charac)*fraction; //The converted number

CurPos=SegFile.GetPosition ();
str.Format ("%f",FloatNo);
AddStrToTrace(str);
return CurPos;
}

void CSegYReaderDlg::AddStrToTrace(CString str)
{
m_TraceData.AddString (str);
}


void CSegYReaderDlg::OnExit()
{
OnOK();
}

void CSegYReaderDlg::OnAbout()
{
        CAboutDlg AboutDlg;
        AboutDlg.DoModal ();
}



void CSegYReaderDlg::OnTraceView()
{
//enabling the plot frame
GetDlgItem(IDC_TRACE_PLOT)->EnableWindow(true);
////getting the device context
CClientDC dc(this);
        {
        CPen pen;
        pen.CreatePen (PS_SOLID,1,RGB(255,0,0));//creating the solid pen
        CRgn lrgn;
```

```
        lrgn.CreateRectRgn (608,100,748,510);//creating the region
        CBrush brush;
        brush.CreateSolidBrush (RGB(255,255,255));//creating the brush
        dc.FillRgn (&lrgn,&brush);//filling the region
        //drawing the central axis
        dc.SelectObject (pen);
        dc.MoveTo (678,100);
        dc.LineTo (678,510);
        UpdateData(true);
        UINT TrDtPs=TracePosArray.GetAt (m_TraceNo-1);
        TrDtPs=TrDtPs+240;
        SegFile.Open (filename,CFile::modeRead,NULL);
        SegFile.Seek (TrDtPs,CFile::begin);
        Trdt.SetSize(1,1);
        max=0;
        int i;
        for(i=0;i<SampNo;i++)
                TrDtPlot();
        PointPlotter(max);
        SegFile.Close ();
        }


}

void CSegYReaderDlg::OnSaveAs()
{


///for saving the file with a new name
CFileDialog SavAsdlg(false);//file open dialog box
        if(SavAsdlg.DoModal ()==IDOK)
                {
                ////checking the validity of the file to be saved
                int i=0;
                i=SegFile.Open(filename,CFile::modeRead,NULL);
                if(i==0)
                        {
                        MessageBox("OPEN a file before saving it.",
                        "ERROR!",MB_OK);
                        }
                //getting the path of the file to be created
                SFileName= SavAsdlg.GetPathName ();
                SaveFile.Open
(SFileName,CFile::modeCreate|CFile::modeWrite,NULL);//opening the
                //newly created file
```

```
////////////////////////////////////
        //reading and saving data in the newly created file
        SaveFile.WriteString ("*********The EBCDIC Header*********\n");
        unsigned char byte[1];
        unsigned char pEbcBuf[3200];//declaring buffer for EBCDIC portion
        SegFile.Read (pEbcBuf,3199);//reading from the file
        for (i=0;i<3200;i++)
                {
                        value=pEbcBuf[i];
                        byte[0]=("%lc",EbcdicToAscii(value));
                        SaveFile.Write (byte,1);
                }
        ////reading the binary header
        SaveFile.WriteString ("\n**********The Binary Header*********\n");
        SegFile.Seek (3200,CFile::begin);
        CurPos=SegFile.GetPosition ();
        for(i=0;i<3;i++)
                {
                FourByteSaver(CurPos);
                CurPos=SegFile.GetPosition ();
                }
        for(i=0;i<24;i++)
                {
                TwoByteSaver(CurPos);
                CurPos=SegFile.GetPosition ();
                }

        SaveFile.WriteString ("**NOTE:-Rest of the bytes upto # 3600 are
unassigned for future development**\n");
        SaveFile.WriteString ("************ THE TRACES ************\n");
//////////reading the traces and adding them to the files////////////////
        unsigned long int T,TraceLocation;
        for(T=0;T<TotTraces;T++)
                {
                TraceLocation=TracePosArray.GetAt (T);
                TraceHdSaver(TraceLocation);
                SaveFile.WriteString ("**********End Of Trace
Header**********\n");
                SaveFile.WriteString ("**********Trace Data*********\n");
                CurPos=SegFile.GetPosition ();
                SegFile.Seek ((CurPos+60),CFile::begin);
                for(i=0;i<SampNo;i++)
                FltPntSav(CurPos);
                SaveFile.WriteString ("**********End of Trace
Data*********\n");
                }
```

```
                SaveFile.Close ();
                SegFile.Close ();
                }

}

unsigned long int CSegYReaderDlg::FourByteSaver(unsigned long CurPos)
{
//declaring the buffer
unsigned char buf[4];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,4);
long int Value= buf[0]*(8388608)+buf[1]*(65536)+buf[2]*(256)+buf[3];
str.Format("%Ld   32-bit Value   %Ld\n",(CurPos+1),Value);
SaveFile.WriteString (str);
return CurPos;
}

unsigned long int CSegYReaderDlg::TwoByteSaver(unsigned long CurPos)
{
//declaring the buffer
unsigned char buf[3];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,2);
long int Value= buf[0]*(256)+buf[1];
str.Format ("%Ld   16-bit Value   %Ld\n",(CurPos+1),Value);
SaveFile.WriteString (str);
return CurPos;
}

unsigned long int CSegYReaderDlg::TraceHdSaver(unsigned long TraceLocation)
{
SegFile.Seek (TraceLocation,CFile::begin);
CurPos=SegFile.GetPosition ();
int i;
for(i=0;i<7;i++)
        {
        FourByteTHdSaver(CurPos,TraceLocation);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<4;i++)
        {
        TwoByteTHdSaver(CurPos,TraceLocation);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<8;i++)
```

```
        {
        FourByteTHdSaver(CurPos,TraceLocation);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<2;i++)
        {
        TwoByteTHdSaver(CurPos,TraceLocation);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<4;i++)
        {
        FourByteTHdSaver(CurPos,TraceLocation);
        CurPos=SegFile.GetPosition ();
        }
for(i=0;i<46;i++)
        {
        TwoByteTHdSaver(CurPos,TraceLocation);
        CurPos=SegFile.GetPosition ();
        }
return CurPos;
}

unsigned long int CSegYReaderDlg::FourByteTHdSaver(unsigned long CurPos,unsigned
long TraceLocation)
{
//declaring the buffer
unsigned char buf[4];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,4);
long int Value= buf[0]*(8388608)+buf[1]*(65536)+buf[2]*(256)+buf[3];
str.Format("%Ld   32-bit Value   %Ld\n",(CurPos-TraceLocation+1),Value);
SaveFile.WriteString (str);
return CurPos;
}

unsigned long int CSegYReaderDlg::TwoByteTHdSaver(unsigned long CurPos,
unsigned long TraceLocation)
{
//declaring the buffer
unsigned char buf[2];
SegFile.Seek (CurPos,CFile::begin);
SegFile.Read (buf,2);
long int Value= buf[0]*(256)+buf[1];
str.Format("%Ld   16-bit Value   %Ld\n",(CurPos-TraceLocation+1),Value);
SaveFile.WriteString (str);
return CurPos;
```

```
}

unsigned long int CSegYReaderDlg::FltPntSav(unsigned long CurPos)
{
///declaring the buffer
unsigned char Sbuf[4];
int sign;
SegFile.Read (Sbuf,4);
unsigned char A=Sbuf[0],B,C,D;
B=A<<1;//removing sign bit
double Char=B>>1;
C=Sbuf[0];
D=C>>7;
if(D==1)//getting the sign
        sign=-1;
else
        sign=1;
double charac;
charac=pow(16,(Char)-64);
float fraction=(Sbuf[1]*65536+Sbuf[2]*256+Sbuf[3])*(0.0000001);
float FloatNo=sign*(charac)*fraction;
CurPos=SegFile.GetPosition ();
CString FloatingPt;
FloatingPt.Format ("%Lf\n",FloatNo);
SaveFile.WriteString (FloatingPt);
return CurPos;
}

void CSegYReaderDlg::TrDtPlot()
{
///declaring the buffer
unsigned char Pbuf[4];
int sign;
SegFile.Read (Pbuf,4);
unsigned char A=Pbuf[0],B,C,D;
B=A<<1;//removing sign bit
double Char=B>>1;
C=Pbuf[0];
D=C>>7;
if(D==1)//getting the sign
        sign=-1;
else
        sign=1;
double charac;
charac=pow(16,(Char)-64);
float fraction=(Pbuf[1]*65536+Pbuf[2]*256+Pbuf[3])*(0.0000001);
```

```
float FloatNo=sign*(charac)*fraction;
int AbsFloatNo=abs(FloatNo);
if(AbsFloatNo>max)
max=AbsFloatNo;
Trdt.Add (FloatNo);
}

void CSegYReaderDlg::PointPlotter(double max)
{
CClientDC dc(this);
        {
        CPen Ppen;
        Ppen.CreatePen (PS_SOLID,1,RGB(0,120,10));//creating the solid pen
        dc.SelectObject (Ppen);
        float VerScale,HoriScale,SmpNum=SampNo;
        VerScale=410/SmpNum;
        float GridLine=100+VerScale;
        int i=0;
        for(i=0;i<SampNo;i++)
                {
                dc.MoveTo (608,GridLine);
                dc.LineTo (748,GridLine);
                GridLine=GridLine+(VerScale*100);
                if(GridLine>510)
                        break;
                }
        CPen Tpen;
        Tpen.CreatePen (PS_SOLID,1,RGB(0,0,255));//creating the solid pen
        dc.SelectObject (Tpen);
        HoriScale=70/max;
        GridLine=100+VerScale;
        dc.MoveTo (680,100);
        for(i=0;i<SampNo;i++)
                {
                int SVal=Trdt.GetAt (i);
                int HVal=678+(SVal*HoriScale);
                dc.LineTo (HVal,GridLine);
                dc.MoveTo (HVal,GridLine);
                GridLine=GridLine+VerScale;
                }
        }
}

void CSegYReaderDlg::OnBnext()
{
UpdateData(true);
```

```
if(m_TraceNo==TotTraces)
        {
        MessageBox("This is the last Trace","ERROR!",MB_OK);
        }
else
        {
        m_TraceNo=m_TraceNo+1;
        UpdateData(false);
        OnApply();
        }
}

void CSegYReaderDlg::OnBprevious()
{
        UpdateData(true);
        if(m_TraceNo==1)
                {
                MessageBox("This is the First Trace","ERROR!",MB_OK);
                }
        else
                {
                m_TraceNo=m_TraceNo-1;
                UpdateData(false);
                OnApply();
                }

}

void CSegYReaderDlg::OnBebcdic()
{
GetDlgItem(IDC_APPLY)->ShowWindow(false);
GetDlgItem(IDC_EDIT_TRACE_NO)->ShowWindow(false);
GetDlgItem(IDC_STATIC_TRACE_NO)->ShowWindow(false);
GetDlgItem(IDC_BNEXT)->ShowWindow(false);
GetDlgItem(IDC_BPREVIOUS)->ShowWindow(false);

GetDlgItem(IDT_BINARY)->ShowWindow(false);
GetDlgItem(IDC_VAL_TYPE)->ShowWindow(false);
GetDlgItem(IDT_VALUE)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_BINARY)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_TRACEHD)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_TRACE)->ShowWindow(false);
GetDlgItem(IDC_VALTYPE)->ShowWindow(false);
GetDlgItem(IDC_VAL)->ShowWindow(false);
GetDlgItem(IDC_BYTENO)->ShowWindow(false);
GetDlgItem(IDC_TRACEHD)->ShowWindow(false);
```

```
GetDlgItem(IDC_TRACEDATA)->ShowWindow(false);
GetDlgItem(IDT_BYTE_NO)->ShowWindow(false);
GetDlgItem(IDT_EBCDIC)->ShowWindow(true);
GetDlgItem(IDC_TRACE_PLOT)->ShowWindow(false);
GetDlgItem(IDT_EBCDIC)->ShowWindow(true);
GetDlgItem(IDT_EBCDIC)->ShowWindow(true);
GetDlgItem(IDC_LISTBOX_EBCDIC)->ShowWindow(true);
GetDlgItem(IDT_EBCDIC)->ShowWindow(true);
}

void CSegYReaderDlg::OnBbinary()
{
GetDlgItem(IDC_APPLY)->ShowWindow(false);
GetDlgItem(IDC_EDIT_TRACE_NO)->ShowWindow(false);
GetDlgItem(IDC_STATIC_TRACE_NO)->ShowWindow(false);
GetDlgItem(IDC_BNEXT)->ShowWindow(false);
GetDlgItem(IDC_BPREVIOUS)->ShowWindow(false);

GetDlgItem(IDT_BINARY)->ShowWindow(true);
GetDlgItem(IDC_VAL_TYPE)->ShowWindow(true);
GetDlgItem(IDT_VALUE)->ShowWindow(true);
GetDlgItem(IDC_LISTBOX_BINARY)->ShowWindow(true);
GetDlgItem(IDC_LISTBOX_TRACEHD)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_TRACE)->ShowWindow(false);
GetDlgItem(IDC_VALTYPE)->ShowWindow(false);
GetDlgItem(IDC_VAL)->ShowWindow(false);
GetDlgItem(IDC_BYTENO)->ShowWindow(false);
GetDlgItem(IDC_TRACEHD)->ShowWindow(false);
GetDlgItem(IDC_TRACEDATA)->ShowWindow(false);
GetDlgItem(IDT_BYTE_NO)->ShowWindow(true);
GetDlgItem(IDT_EBCDIC)->ShowWindow(false);
GetDlgItem(IDC_TRACE_PLOT)->ShowWindow(false);
GetDlgItem(IDT_EBCDIC)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_EBCDIC)->ShowWindow(false);
}

void CSegYReaderDlg::OnBtrace()
{
GetDlgItem(IDC_APPLY)->ShowWindow(true);
GetDlgItem(IDC_EDIT_TRACE_NO)->ShowWindow(true);
GetDlgItem(IDC_STATIC_TRACE_NO)->ShowWindow(true);
GetDlgItem(IDC_BNEXT)->ShowWindow(true);
GetDlgItem(IDC_BPREVIOUS)->ShowWindow(true);

GetDlgItem(IDT_BINARY)->ShowWindow(false);
GetDlgItem(IDC_VAL_TYPE)->ShowWindow(false);
```

```
GetDlgItem(IDT_VALUE)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_BINARY)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_TRACEHD)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_TRACE)->ShowWindow(true);
GetDlgItem(IDC_VALTYPE)->ShowWindow(true);
GetDlgItem(IDC_VAL)->ShowWindow(true);
GetDlgItem(IDC_BYTENO)->ShowWindow(true);
GetDlgItem(IDC_TRACEHD)->ShowWindow(true);
GetDlgItem(IDC_TRACEDATA)->ShowWindow(true);
GetDlgItem(IDT_BYTE_NO)->ShowWindow(false);
GetDlgItem(IDT_EBCDIC)->ShowWindow(false);
GetDlgItem(IDC_TRACE_PLOT)->ShowWindow(true);
GetDlgItem(IDT_EBCDIC)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_EBCDIC)->ShowWindow(false);
GetDlgItem(IDC_LISTBOX_TRACEHD)->ShowWindow(true);
}

void CSegYReaderDlg::OnEbcdicHlp()
{
WinExec("C:/SegY Reader/EBCDIC HEADER.htm",SW_SHOW);
}
```

REFERENCES

- Chapman Davis (1980), "Visual C++ in 21 days", G.C Jain of Techmedia, New Delhi,
- Barry K.M, Cavers, D.A, Kneale, C.W, " Recommended standard for digital tape formats". Seg technical standard committee,
- Dobrin M.B & Savit C.H., (1976), "Introduction to Geophysical Prospecting", (4^{th} Edition), Mc, Graw-Hill Book Compan,