**Polar HRM Data Analysis System**

Table of Contents

[Introduction: 2](#_Toc532278133)

[Version Control 2](#_Toc532278134)

[Commits on Visual Studio 2](#_Toc532278135)

[Commit on GitHub 3](#_Toc532278136)

[Architecture 4](#_Toc532278137)

[Front End Design: 4](#_Toc532278138)

[**Main Form:** 5](#_Toc532278139)

[**Data Summary** 5](#_Toc532278140)

[Backend Code 9](#_Toc532278141)

[HRDataClass 9](#_Toc532278142)

[MasterClass 9](#_Toc532278143)

[Backend of DataSummary 13](#_Toc532278144)

[Backend Code of Chart 33](#_Toc532278145)

[Backend of Main Form: 52](#_Toc532278146)

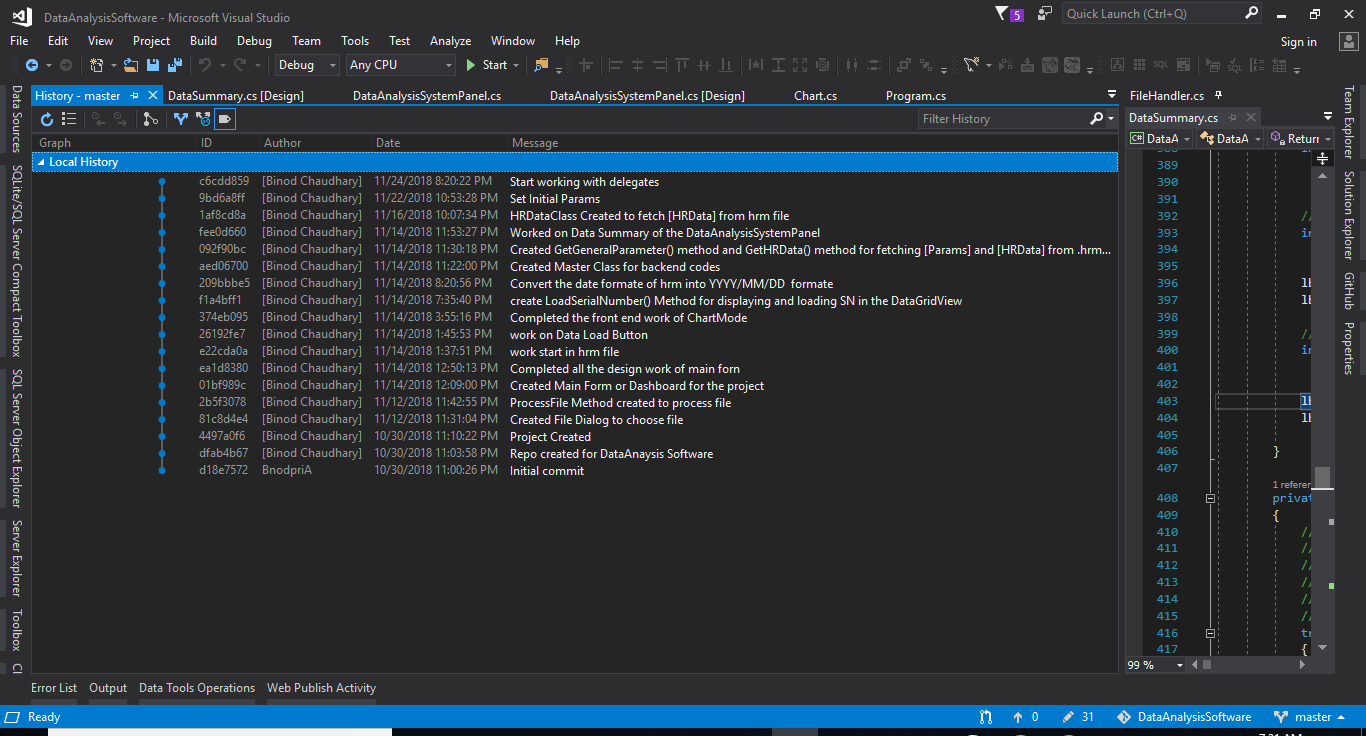
# Introduction:

Polar HRM Data Analysis System is Data Analysis Software which read data from .hrm file of polar devices. Mainly this project focus to read two types of data from .hrm file They are [Params] Parameter or Header file which hold information of devices and different parameters and [HRData] which hold dynamic data of Speed, Cadence, Heart Beat, Pressure, etc. which can be shown in Datagridview and ZedGraph. Likewise, for tracking the project changes I have used git as version controlling system and GitHub to host it.

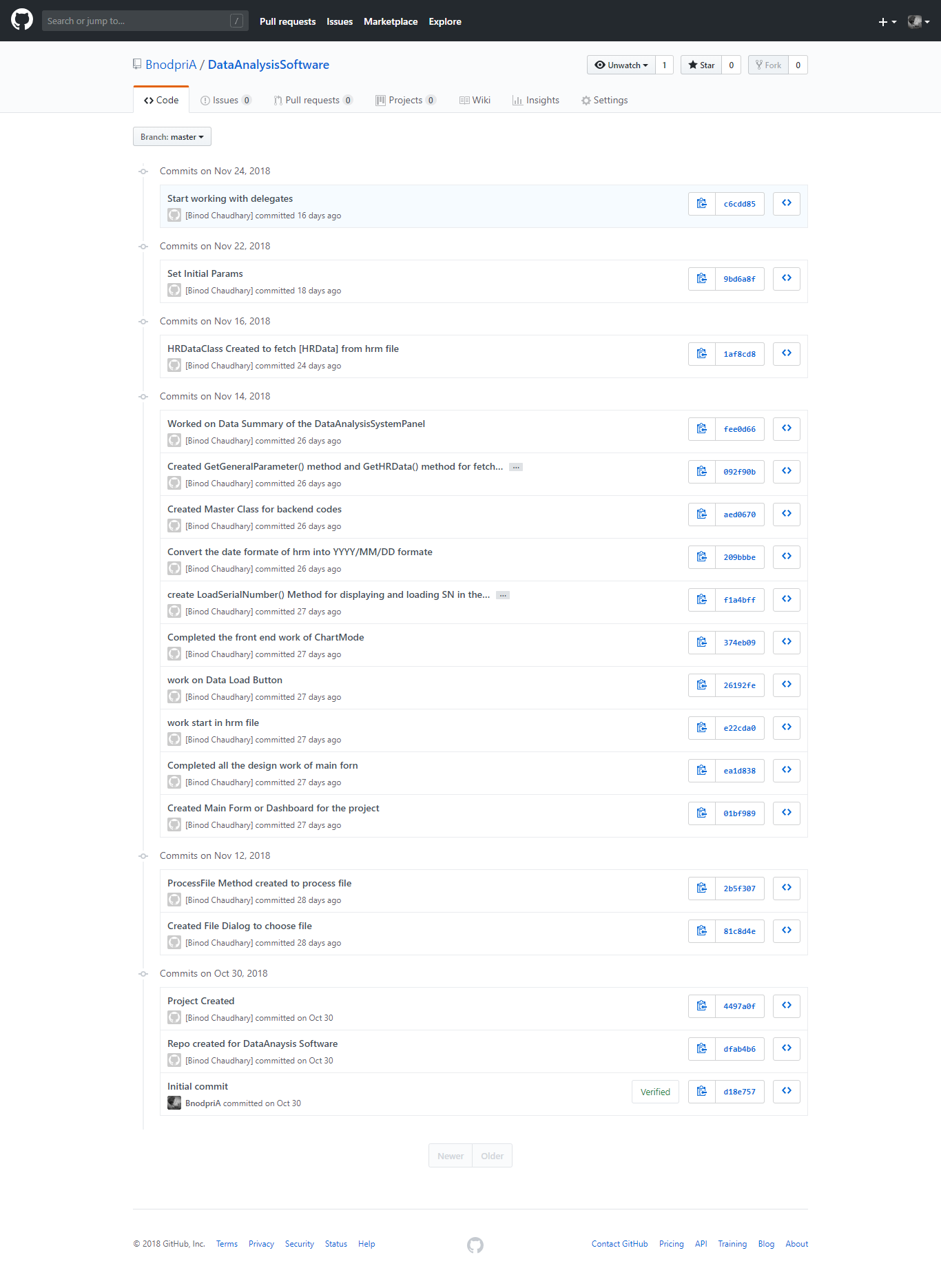
# Version Control

To ensure that the code has been synchronized in the GitHub I have commits changes and checked both commit from local and GitHub.

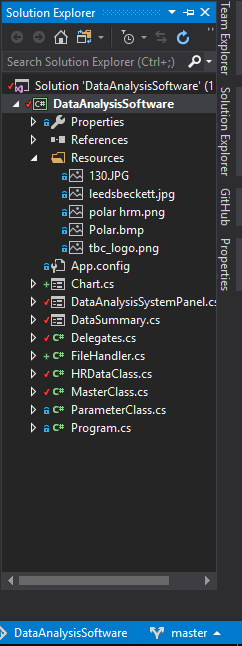
## Commits on Visual Studio



## Commit on GitHub



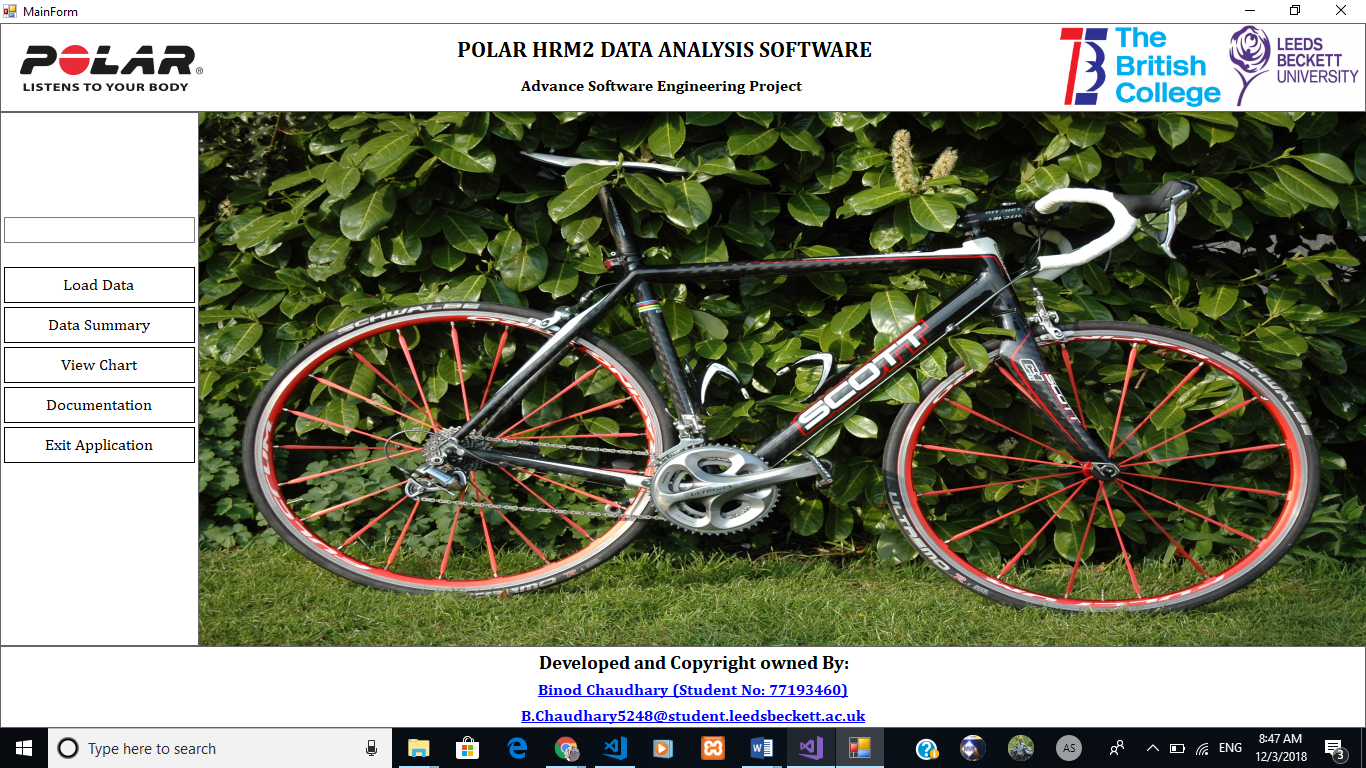
# Architecture



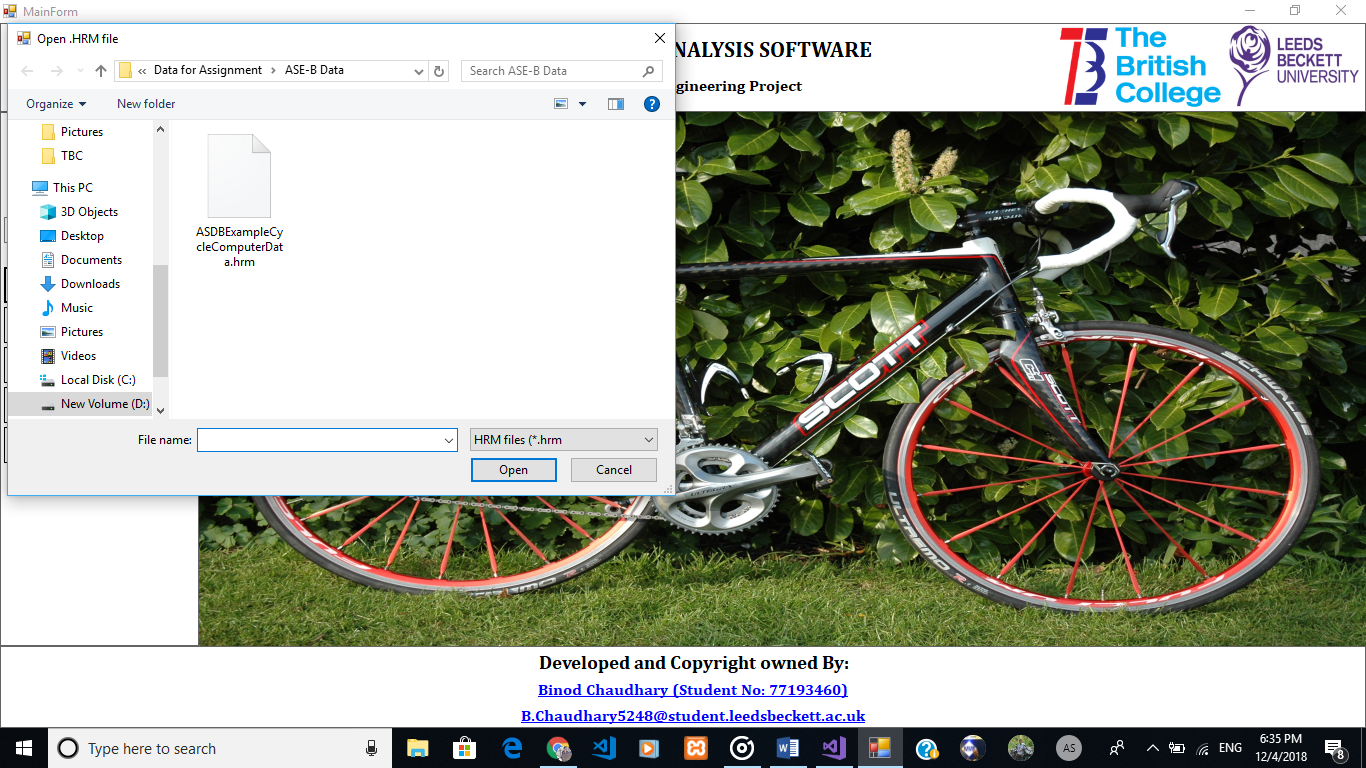
# Front End Design:

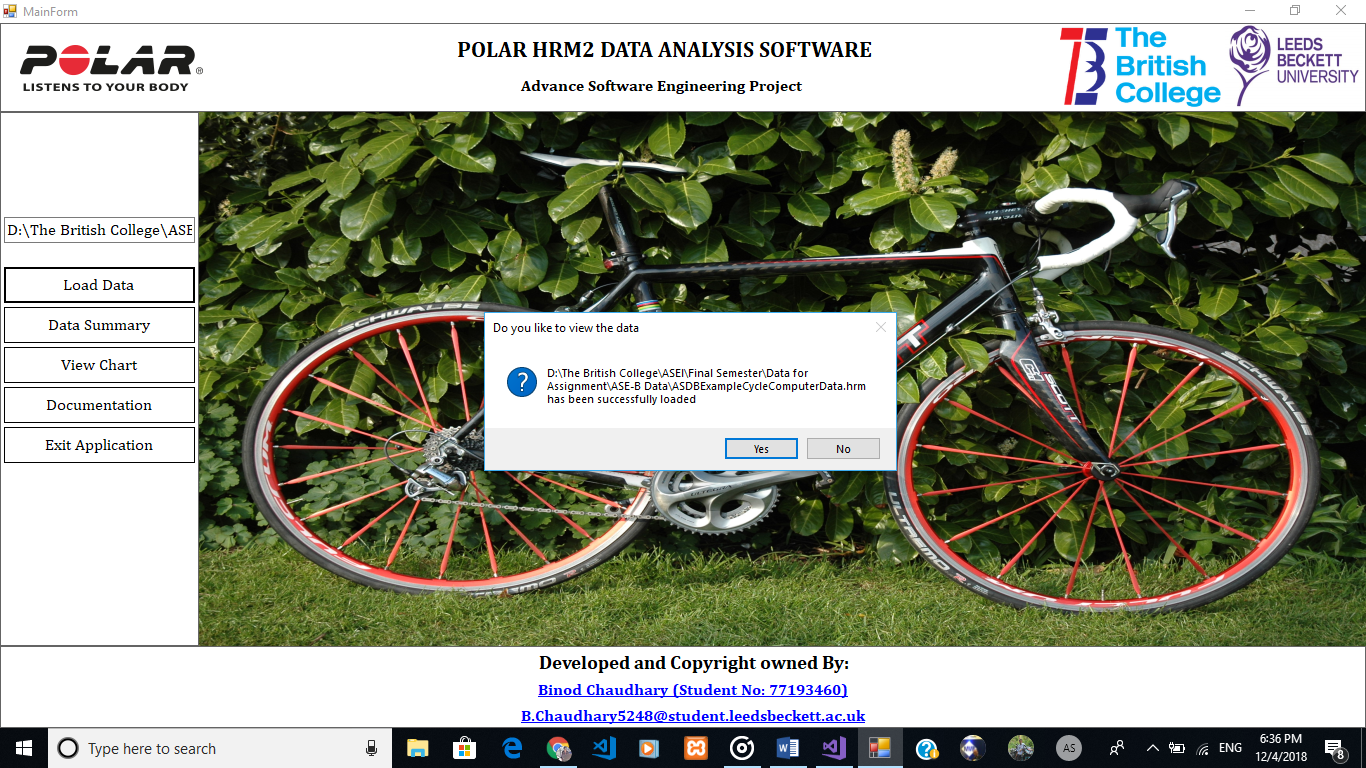
For Front End Design I used Window Form Application of DOT NET Framework (C#) of Microsoft technology. Here I have 3 major form. Main Form and other two form named as DataSummary and Chart DataSummary display data in tabular form in datagridview while chart shows the graph of data in ZedGraph.

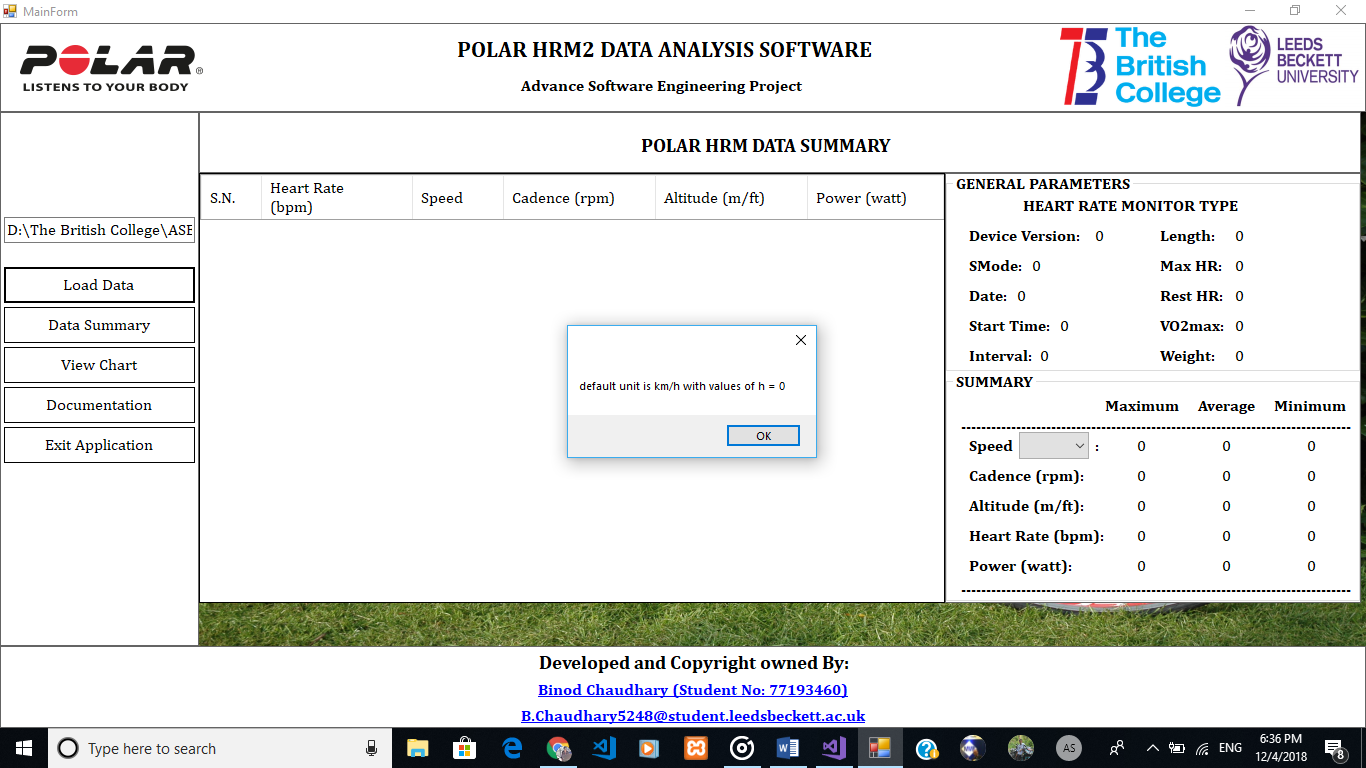
### **Main Form:**

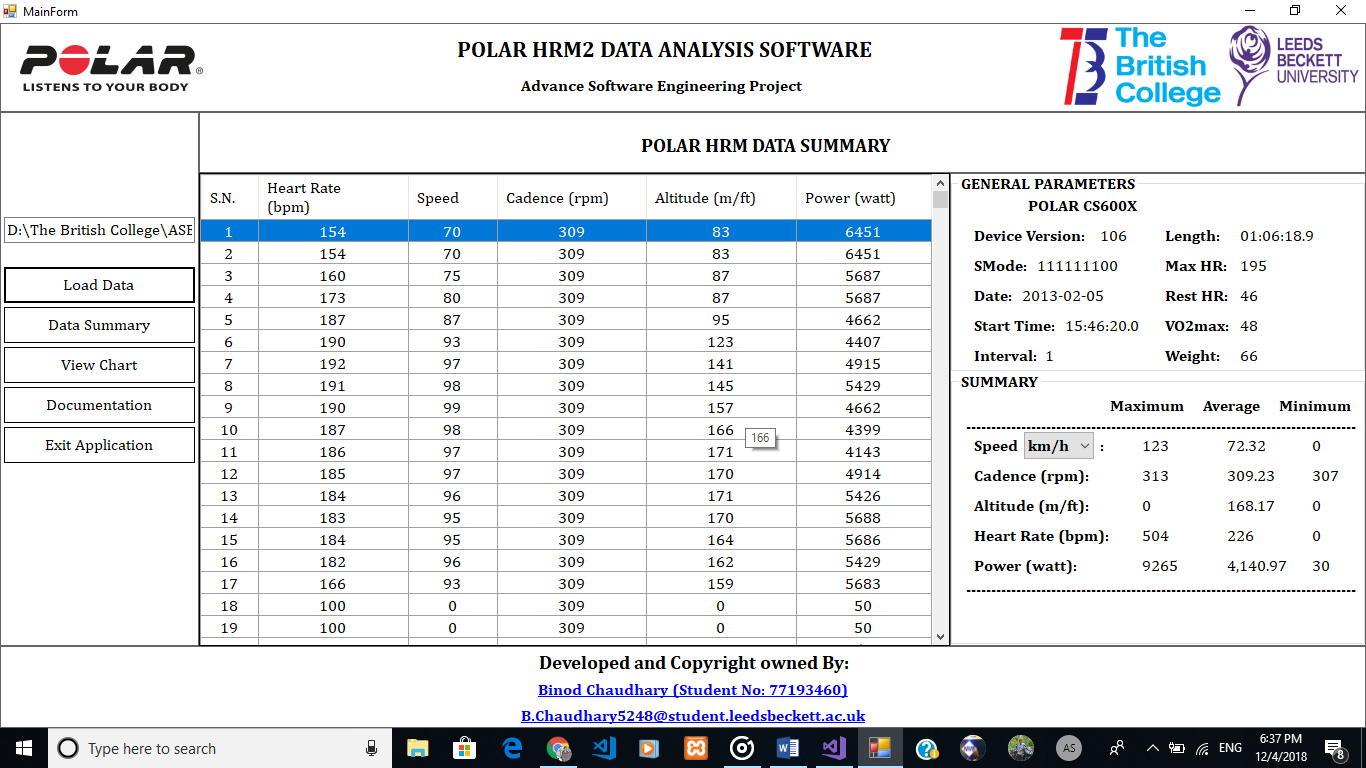


### **Data Summary**











# Backend Code

## HRDataClass

This class handle [HRData] and their values in corresponding variables

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAnalysisSoftware

{/// <summary>

/// Class to fetch [HRData] as per HRM versions

/// </summary>

public class HRDataClass

{

//Data for HRM version 1.06

public int[] HeartRate { get; set; }

public double[] Speed { get; set; }

public int[] Cadence { get; set; }

public int[] Altitude { get; set; }

public double Power { get; set; }

public int PowerBalanceAndPedallingIndex { get; set; }

//Addition data add in HRM version 1.07

public int AirPressure { get; set; }

}

}

## MasterClass

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using static DataAnalysisSoftware.Delegates;

namespace DataAnalysisSoftware

{

/// <summary>

/// Class for controlling backed of the DataAnalysisSoftware

/// </summary>

public class MasterClass

{

/// <summary>

/// GetPolarDeviceName from its monitor no for e.g. if monitor=1 then device name = "Polar Sport Tester / Vantage"

/// </summary>

/// <param name="monitor"></param>

/// <returns></returns>

///

public string fileName;

private ReadContentsCallBack readContentsCallBack;

private ConfigurationLoadedCallBack configLoadCallBack;

public static string GetPolarDeviceName(int monitor)

{

string deviceName = null;

switch (monitor)

{

case 1:

{

deviceName = "Polar Sport Tester/ Vantage";

break;

}

case 2:

{

deviceName = "Polar Vantage NV (VNV)";

break;

}

case 3:

{

deviceName = "Polar Accurex Plus";

break;

}

case 4:

{

deviceName = "Polar XTrainer Plus";

break;

}

case 6:

{

deviceName = "Polar S520";

break;

}

case 7:

{

deviceName = "Polar Coach";

break;

}

case 8:

{

deviceName = "Polar S210";

break;

}

case 9:

{

deviceName = "Polar S410";

break;

}

case 10:

{

deviceName = "Polar S510";

break;

}

case 11:

{

deviceName = "Polar S610 / S610i";

break;

}

case 12:

{

deviceName = "Polar S710 / S710i / S720i";

break;

}

case 13:

{

deviceName = "Polar S810 / S810i";

break;

}

case 15:

{

deviceName = "Polar E600";

break;

}

case 20:

{

deviceName = "Polar AXN500";

break;

}

case 21:

{

deviceName = "Polar AXN700";

break;

}

case 22:

{

deviceName = "Polar S625X / S725X";

break;

}

case 23:

{

deviceName = "Polar S725";

break;

}

case 33:

{

deviceName = "Polar CS400";

break;

}

case 34:

{

deviceName = "Polar CS600X";

break;

}

case 35:

{

deviceName = "Polar CS600";

break;

}

case 36:

{

deviceName = "Polar RS400";

break;

}

case 37:

{

deviceName = "Polar RS800";

break;

}

case 38:

{

deviceName = "Polar RS800X";

break;

}

default:

break;

}

return deviceName;

}

public void FileHandler(string fileName, ReadContentsCallBack readContents)

{

this.fileName = fileName;

this.readContentsCallBack = readContents;

}

public void FileHandler(string fileName, ConfigurationLoadedCallBack configLoadBack)

{

this.fileName = fileName;

this.configLoadCallBack = configLoadBack;

}

/// <summary>

/// Get [Params] from hrm file

/// </summary>

public static void GetGeneralParameters()

{

var config = new Dictionary<string, string>();

bool fileExists = File.Exists(""); ///don't forget to assign file path

if(!fileExists)

{

//file doesn't exist, no point is continuing

}

bool startAppending=false;

string line;

StreamReader paramsFiles = new StreamReader("");

try

{

while ((line = paramsFiles.ReadLine()) != null)

{

if(!startAppending)

{

startAppending = line.Equals("[Params]") ? true : false;

continue;

}

}

}

catch (Exception)

{

throw;

}

}

/// <summary>

/// Get [HRData] from hrm file

/// </summary>

public static void GetHRMData()

{

}

}

}

## Backend of DataSummary

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Text.RegularExpressions;

using System.Collections;

namespace DataAnalysisSoftware

{

public partial class DataSummary : Form

{

public DataSummary()

{

InitializeComponent();

}

public string fileName;

private string singleLine;

private string defaultUnit = "0";

public string deviceName;

public string version;

public string monitor;

public string sMode;

public string date;

public string startTime;

public string length;

public string interval;

public string upper1;

public string upper2;

public string upper3;

public string lower1;

public string lower2;

public string lower3;

public string timer1;

public string timer2;

public string timer3;

public string activeLimit;

public string maxHR;

public string restHR;

public string startDelay;

public string vo2Max;

public string weight;

double totalSpeed;

double totalCadence;

double totalAltitude;

int totalHeartRate;

double totalPower;

double averageSpeed;

double averageCadence;

double averageAltitude;

int averageHeartRate;

double averagePower;

double maxSpeed;

IDictionary<string, string> param = new Dictionary<string, string>();

ArrayList hra = new ArrayList();

ParameterClass pc = new ParameterClass();

MasterClass mc = new MasterClass();

HRDataClass hrdc = new HRDataClass();

private void DataSummary\_Load(object sender, EventArgs e)

{

// cmbSpeedUnit.SelectedIndex = -1;

ShowOrHidePowerBySMode();

DefaultSpeedUnitBySMode();

lblMonitorType.Text = GetPolarDeviceName(Convert.ToInt32(monitor));

lblDeviceVersion.Text = version;

lblSMode.Text = sMode;

lblDate.Text = date;

lblStartTime.Text = startTime;

lblInterval.Text = interval;

lblLength.Text = length;

lblMaxHR.Text = maxHR;

lblRestHR.Text = restHR;

lblVO2max.Text = vo2Max;

lblWeight.Text = weight;

ReadHRData();

LoadSerialNumber();

ReturnMaxMinHRMValues();

CalculateAverageHeartRate();

CalculateAverageSpeed();

CalculateAverageCadence();

CalculateAverageAltitude();

CalculateAveragePower();

}

/// <summary>

/// Load Serial Number

/// </summary>

private void LoadSerialNumber()

{

int i = 1;

foreach (DataGridViewRow row in dgvHRMData.Rows)

{

row.Cells["SNo"].Value = i;

i++;

i.ToString();

}

}

public static string[] SplitParm(string line)

{

return line.Split('=');

}

public static string[] SplitData(string line)

{

return Regex.Split(line, @"\W+");

}

/// <summary>

/// Display Date in Understandable formate i.e. 20130205 to 2013/02/05 (YYYY/MM/DD) Format

/// </summary>

private void ManageDateFormate()

{

}

public void LoadData()

{

}

/// <summary>

/// Return Device Name or Monitor type by reading monitor number from hrm file

/// </summary>

/// <param name="monitor"></param>

/// <returns></returns>

private static string GetPolarDeviceName(int monitor)

{

string deviceName = "HEART RATE MONITOR TYPE";

switch (monitor)

{

case 1:

{

deviceName = "POLAR SPORT TESTER/ VANTAGE";

break;

}

case 2:

{

deviceName = "POLAR VANTAGE NV (VNV)";

break;

}

case 3:

{

deviceName = "POLAR ACCUREX PlUS";

break;

}

case 4:

{

deviceName = "POLAR XTRAINER PLUS";

break;

}

case 6:

{

deviceName = "POLAR S520";

break;

}

case 7:

{

deviceName = "POLAR COACH";

break;

}

case 8:

{

deviceName = "POLAR S210";

break;

}

case 9:

{

deviceName = "POLAR S410";

break;

}

case 10:

{

deviceName = "POLAR S510";

break;

}

case 11:

{

deviceName = "POLAR S610 / S610I";

break;

}

case 12:

{

deviceName = "POLAR S710 / S710I / S720I";

break;

}

case 13:

{

deviceName = "POLAR S810 / S810I";

break;

}

case 15:

{

deviceName = "POLAR E600";

break;

}

case 20:

{

deviceName = "POLAR AXN500";

break;

}

case 21:

{

deviceName = "POLAR AXN700";

break;

}

case 22:

{

deviceName = "POLAR S625X / S725X";

break;

}

case 23:

{

deviceName = "POLAR S725";

break;

}

case 33:

{

deviceName = "POLAR CS400";

break;

}

case 34:

{

deviceName = "POLAR CS600X";

break;

}

case 35:

{

deviceName = "POLAR CS600";

break;

}

case 36:

{

deviceName = "POLAR RS400";

break;

}

case 37:

{

deviceName = "POLAR RS800";

break;

}

case 38:

{

deviceName = "POLAR RS800X";

break;

}

default:

break;

}

return deviceName;

}

private void CalculateAverageSpeed()

{

try

{

totalSpeed = 0;

for (int i = 0; i < dgvHRMData.Rows.Count; i++)

{

totalSpeed += Convert.ToDouble(dgvHRMData.Rows[i].Cells["Speed"].Value.ToString());

}

averageSpeed = totalSpeed / (Convert.ToDouble(dgvHRMData.Rows.Count));

lblAvgSpeed.Text = averageSpeed.ToString("N2");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void CalculateAverageCadence()

{

try

{

totalCadence = 0;

for (int i = 0; i < dgvHRMData.Rows.Count; i++)

{

totalCadence += Convert.ToDouble(dgvHRMData.Rows[i].Cells["Cadence"].Value.ToString());

}

averageCadence = totalCadence / (Convert.ToDouble(dgvHRMData.Rows.Count));

lblAvgCadence.Text = averageCadence.ToString("N2");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void CalculateAverageAltitude()

{

try

{

totalAltitude = 0;

for (int i = 0; i < dgvHRMData.Rows.Count; i++)

{

totalAltitude += Convert.ToDouble(dgvHRMData.Rows[i].Cells["Altitude"].Value.ToString());

}

averageAltitude = totalAltitude / (Convert.ToDouble(dgvHRMData.Rows.Count));

lblAvgAltitude.Text = averageAltitude.ToString("N2");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void CalculateAverageHeartRate()

{

try

{

totalHeartRate = 0;

for (int i = 0; i < dgvHRMData.Rows.Count; i++)

{

totalHeartRate += Convert.ToInt32(dgvHRMData.Rows[i].Cells["HeartRate"].Value.ToString());

}

averageHeartRate = totalHeartRate / (Convert.ToInt32(dgvHRMData.Rows.Count));

lblAvgHeartRate.Text = averageHeartRate.ToString();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void CalculateAveragePower()

{

try

{

totalPower = 0;

for (int i = 0; i < dgvHRMData.Rows.Count; i++)

{

totalPower += Convert.ToDouble(dgvHRMData.Rows[i].Cells["Power"].Value.ToString());

}

averagePower = totalPower / (Convert.ToDouble(dgvHRMData.Rows.Count));

lblAvgPower.Text = averagePower.ToString("N2");

}

catch (Exception)

{

throw;

}

}

/// <summary>

/// Convert Speed From KM per hours to Mile per Hour if h=0 for SMode

/// </summary>

/// <param name="kmph"></param>

/// <returns></returns>

private double ConvertSpeedKMPHToMPH( double kmph)

{

return 0.614 \* kmph;

}

/// <summary>

///

/// </summary>

private double ConvertSpeedMPHToKMPH(double mph)

{

return mph \* 1.60934;

}

private void ReturnMaxMinHRMValues()

{

///Finding max and min value from Speed Column of dgvHRMData DataGridView

double[] columnSpeed = (from DataGridViewRow row in dgvHRMData.Rows

where row.Cells["Speed"].FormattedValue.ToString() != string.Empty

select Convert.ToDouble(row.Cells["Speed"].FormattedValue)).ToArray(); //Selected all the value from Speed Column of dgvHRMData

lblMaxSpeed.Text = columnSpeed.Max().ToString();//retrun the max value in Speed Column to lblMaxSpeed

lblMinSpeed.Text = columnSpeed.Min().ToString();

maxSpeed = Convert.ToDouble(columnSpeed.Max().ToString());

///Finding max and min value from Cadence Column of dgvHRMData DataGridView

int[] columnCadence = (from DataGridViewRow row in dgvHRMData.Rows

where row.Cells["Cadence"].FormattedValue.ToString() != string.Empty

select Convert.ToInt32(row.Cells["Cadence"].FormattedValue)).ToArray();

lblMaxCadence.Text = columnCadence.Max().ToString();

lblMinCadence.Text = columnCadence.Min().ToString();

///Finding max and min value from Altitude Column of dgvHRMData DataGridView

int[] columnAltitude = (from DataGridViewRow row in dgvHRMData.Rows

where row.Cells["Altitude"].FormattedValue.ToString() != string.Empty

select Convert.ToInt32(row.Cells["Altitude"].FormattedValue)).ToArray();

///Finding max and min value from HeartRate Column of dgvHRMData DataGridView

int[] columnHeartRate = (from DataGridViewRow row in dgvHRMData.Rows

where row.Cells["HeartRate"].FormattedValue.ToString() != string.Empty

select Convert.ToInt32(row.Cells["HeartRate"].FormattedValue)).ToArray();

lblMaxHeartRate.Text = columnHeartRate.Max().ToString();

lblMinHeartRate.Text = columnHeartRate.Min().ToString();

///Finding max and min value from Power Column of dgvHRMData DataGridView

int[] columnPower = (from DataGridViewRow row in dgvHRMData.Rows

where row.Cells["Power"].FormattedValue.ToString() != string.Empty

select Convert.ToInt32(row.Cells["Power"].FormattedValue)).ToArray();

lblMaxPower.Text = columnPower.Max().ToString();

lblMinPower.Text = columnPower.Min().ToString();

}

private void ReadHRData()

{

//string fileContent = File.ReadAllText(data);

//int indexOfHRData = fileContent.IndexOf("[HRData]");

//string formatedHRData = fileContent.Substring(indexOfHRData + 10, fileContent.Length - (indexOfHRData + 10));

//StreamReader hrmDataReader = new StreamReader(formatedHRData);

//int row = 0;

//while(

try

{

string fileContent = File.ReadAllText(fileName);

string toFind = "[HRData]";

int index = fileContent.IndexOf(toFind);

string formattedFileContent = fileContent.Substring(index + 10, fileContent.Length - (index + 10));

StringReader stringReader = new StringReader(formattedFileContent);

int row = 0;

while ((singleLine = stringReader.ReadLine()) != null)

{

string[] columnData = singleLine.Split('\t');

dgvHRMData.Rows.Add();

for (int i = 1; i < columnData.Length; i++)

{

dgvHRMData[i, row].Value = columnData[i];

}

row++;

}

}

catch (NullReferenceException ex)

{

MessageBox.Show(ex.Message);

}

catch(ArgumentNullException ex)

{

MessageBox.Show(ex.Message);

}

}

public void LoadSummartData()

{

List<int> heartRate = new List<int>();

List<double> speed = new List<double>();

List<int> cadence = new List<int>();

List<int> altitude = new List<int>();

List<int> power = new List<int>();

foreach (DataGridViewRow row in dgvHRMData.Rows)

{

heartRate.Add(Convert.ToInt32(row.Cells["HeartRate"].Value.ToString()));

speed.Add(Convert.ToDouble(row.Cells["Speed"].Value.ToString()));

cadence.Add(Convert.ToInt32(row.Cells["Cadence"].Value.ToString()));

altitude.Add(Convert.ToInt32(row.Cells["Altitude"].Value.ToString()));

power.Add(Convert.ToInt32(row.Cells["Power"].Value.ToString()));

}

hrdc.HeartRate = heartRate.ToArray();

}

private void cmbSpeedUnit\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmbSpeedUnit.Text== "km/h")

{

lblMaxSpeed.Text = maxSpeed.ToString();

}

else

{

maxSpeed = ConvertSpeedKMPHToMPH(maxSpeed);

lblMaxSpeed.Text = maxSpeed.ToString();

}

}

private void ShowOrHidePowerBySMode()

{

try

{

if (fileName != "")

{

//DataSummary ds = new DataSummary();

StreamReader sModeReader = new StreamReader(fileName, Encoding.Default);

string textOfSMode = sModeReader.ReadToEnd();

int indexOfSMode = textOfSMode.IndexOf("SMode=");

string speedStatus = textOfSMode.Substring(indexOfSMode + 6, 1);

string cadenceStatus = textOfSMode.Substring(indexOfSMode + 7, 1);

string altitudeStatus = textOfSMode.Substring(indexOfSMode + 8, 1);

string powerStatus = textOfSMode.Substring(indexOfSMode + 9, 1);

if (speedStatus.ToString()=="0")

{

dgvHRMData.Columns[2].Visible = false;

}

else if (speedStatus.ToString()=="1")

{

dgvHRMData.Columns[2].Visible = true;

}

else

{

MessageBox.Show("Invalid SMode 'a' value for Speed, a must be 0=off or 1=on");

dgvHRMData.Columns[2].Visible = false;

}

//Checking SMode value for (b) or cadence

if (cadenceStatus.ToString() == "0")

{

dgvHRMData.Columns[3].Visible = false;//Column[3] is for Cadence

}

else if (cadenceStatus.ToString() == "1")

{

dgvHRMData.Columns[3].Visible = true;

}

else

{

MessageBox.Show("Invalid SMode 'b' value for Cadence, b must be 0=off or 1=on");

dgvHRMData.Columns[3].Visible = false;

}

//Checking SMode value for (c) or Altitude

if (altitudeStatus.ToString() == "0")

{

dgvHRMData.Columns[4].Visible = false;

}

else if (altitudeStatus.ToString() == "1")

{

dgvHRMData.Columns[4].Visible = true;

}

else

{

MessageBox.Show("Invalid SMode 'c' value for Altitude, c must be 0=off or 1=on");

dgvHRMData.Columns[4].Visible = false;

}

///Show or Hide Power based on SMode if d=1 show else d=0 hide Power Column

if (powerStatus.ToString() == "0")

{

dgvHRMData.Columns[5].Visible = false;

}

else if(powerStatus.ToString()=="1")

{

dgvHRMData.Columns[5].Visible = true;

}

else

{

MessageBox.Show("Invalid SMode 'd' value for Power, d must be 0=off or 1=on");

dgvHRMData.Columns[5].Visible = false;

}

}

}

catch (NullReferenceException ex)

{

MessageBox.Show("No hrm data was found");

}

}

private void DefaultSpeedUnitBySMode()

{

try

{

if (fileName != "")

{

StreamReader sModeReader = new StreamReader(fileName, Encoding.Default);

string textOfSMode = sModeReader.ReadToEnd();

int indexOfSMode = textOfSMode.IndexOf("SMode=");

defaultUnit = textOfSMode.Substring(indexOfSMode + 13, 1);

if (defaultUnit.ToString() == "0")

{

MessageBox.Show("default unit is km/h with values of h = " + defaultUnit);

cmbSpeedUnit.Text = "km/h";

}

else if (defaultUnit.ToString() == "1")

{

MessageBox.Show("default unit is mph with values of h = " + defaultUnit);

cmbSpeedUnit.Text= "mph";

}

else

{

MessageBox.Show("values of h = " + defaultUnit + "So Unit of Speed is Invalid");

cmbSpeedUnit.SelectedIndex = -1;

}

}

}

catch (NullReferenceException)

{

MessageBox.Show("No hrm data was found");

}

}

}

}

## Backend Code of Chart

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Windows.Forms;

using System.Threading;

using System.Text.RegularExpressions;

using System.Timers;

using ZedGraph;

namespace DataAnalysisSoftware

{

public partial class Chart : Form

{

public string fileName;

delegate void HRDataMethod(string hrData); //This Delegate is a type safe function pointer for HRDataMethod

delegate void AxisChangeZedGraphCallBack(ZedGraphControl zg);

public Thread garthererThread;

LineItem lineOfHeartRate, lineOfSpeed, lineOfCadence, lineOfAltitude, lineOfPower; //Declaring Curve Line Item

PointPairList pointPairListOfHeartRate;

PointPairList pointPairListOfSpeed;

PointPairList pointPairListOfCadence;

PointPairList pointPairListOfAltitude;

PointPairList pointPairListOfPower;

string[] heartRates; //Arrary to store Heart Rate

string[] speeds;

string[] cadences;

string[] altitudes;

string[] powers;

private void Chart\_Load(object sender, EventArgs e)

{

GraphPane graphPane = zrcHRMGraph.GraphPane;

graphPane.Title.Text = "Polar HRM Data Analysis";

graphPane.XAxis.Title.Text = "X Axis (Values)";

graphPane.YAxis.Title.Text = "Y Axis " + "\n\n" + "(Heart Rate, Speed, Cadence, Altitude, Power)";

pointPairListOfHeartRate = new PointPairList();//Declaring PointPairList for Heart Rate

lineOfHeartRate = graphPane.AddCurve("Heart Rate", pointPairListOfHeartRate, Color.DarkRed, SymbolType.TriangleDown); //Assign lineItem and symbol for PointPairList for Heart Rate

pointPairListOfSpeed = new PointPairList();//Declaring PointPairList for Speed

lineOfSpeed = graphPane.AddCurve("Speed", pointPairListOfSpeed, Color.Orange, SymbolType.Plus); //Assign lineItem and symbol for PointPairList for Speed

pointPairListOfCadence = new PointPairList();//Declaring PointPairList for Cadence

lineOfCadence = graphPane.AddCurve("Cadence", pointPairListOfCadence, Color.Green, SymbolType.Circle);

pointPairListOfAltitude = new PointPairList();

lineOfAltitude = graphPane.AddCurve("Altitude", pointPairListOfAltitude, Color.DarkGray, SymbolType.Triangle);

pointPairListOfPower = new PointPairList();

lineOfPower = graphPane.AddCurve("Power", pointPairListOfPower, Color.Maroon, SymbolType.XCross);

//call HeartRateThread

Thread heartRateThread = new Thread(new ThreadStart(HeartRateThread));

heartRateThread.Start();

//call SpeedThread

Thread speedThread = new Thread(new ThreadStart(SpeedThread));

speedThread.Start();

//call CadenceThread

Thread cadenceThread = new Thread(new ThreadStart(CadenceThread));

cadenceThread.Start();

//call AltitudeThread

Thread altitudeThread = new Thread(new ThreadStart(AltitudeThread));

altitudeThread.Start();

//call PowerThread

Thread powerThread = new Thread(new ThreadStart(PowerThread));

powerThread.Start();

//call CreateGraph Thread

Thread createGraphThread = new Thread(new ThreadStart(CreateGraph));

createGraphThread.Start();

}

public Chart()

{

InitializeComponent();

}

private void AxisChangeZedGraph(ZedGraphControl zg)

{

if (zg.InvokeRequired)

{

AxisChangeZedGraphCallBack ad = new AxisChangeZedGraphCallBack(AxisChangeZedGraph);

zg.Invoke(ad, new object[] { zg });

}

else

{

zrcHRMGraph.AxisChange();

zg.Invalidate();

zg.Refresh();

}

}

private void TimerCallBack(object source, ElapsedEventArgs e)

{

garthererThread = new Thread(new ThreadStart(GartherData));

}

/// <summary>

/// Find the work of it

/// </summary>

private void GartherData()

{

List<float> gadtheredInfo = new List<float>();

zrcHRMGraph.Invoke(new MethodInvoker(delegate { zrcHRMGraph.Invalidate(); }));

}

private void CreateGraph()

{

try

{

string hrmData = File.ReadAllText(fileName);

string[] hrmDataArr = hrmData.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);

int indexOfHRMDataArr = Array.IndexOf(hrmDataArr, "[HRData]");

heartRates = new string[hrmDataArr.Length - indexOfHRMDataArr];

speeds = new string[hrmDataArr.Length - indexOfHRMDataArr];

cadences = new string[hrmDataArr.Length - indexOfHRMDataArr];

altitudes = new string[hrmDataArr.Length - indexOfHRMDataArr];

powers = new string[hrmDataArr.Length - indexOfHRMDataArr];

int j = 0;

// Make up some, data arrays based on the Sine function

double x, y1, y2, y3, y4, y5;

PointPairList pointPairListOfHeartRate = new PointPairList();

PointPairList pointPairListOfSpeed = new PointPairList();

PointPairList pointPairListOfCadence = new PointPairList();

PointPairList pointPairListOfAltitude = new PointPairList();

PointPairList pointPairListOfPower = new PointPairList();

for (int i = indexOfHRMDataArr + 1; i < hrmDataArr.Length - 1; i++)

{

string hrData = hrmDataArr[i];

string[] hrDataArr = Regex.Split(hrData, @"\W+");

heartRates[j] = hrDataArr[0];

speeds[j] = hrDataArr[1];

cadences[j] = hrDataArr[2];

altitudes[j] = hrDataArr[3];

powers[j] = hrDataArr[4];

int heartRate = Convert.ToInt32(heartRates[j]);

int speed = Convert.ToInt32(speeds[j]);

int cadence = Convert.ToInt32(cadences[j]);

int altitude = Convert.ToInt32(altitudes[j]);

int power = Convert.ToInt32(powers[j]);

x = i;

y1 = heartRate;

y2 = speed;

y3 = cadence;

y4 = altitude;

y5 = power;

//Adding lineOfHeartRate in zrcHRMGraph

LineItem lineOfHeartRate = zrcHRMGraph.GraphPane.CurveList["Heart Rate"] as LineItem;

IPointListEdit heartList = lineOfHeartRate.Points as IPointListEdit;

heartList.Add(x, y1);

//Adding lineOfSpeed in zrcHRMGraph

LineItem lineOfSpeed = zrcHRMGraph.GraphPane.CurveList["Speed"] as LineItem;

IPointListEdit speedList = lineOfSpeed.Points as IPointListEdit;

speedList.Add(x, y2);

//Adding lineOfCadence in zrcHRMGraph

LineItem lineOfCadence = zrcHRMGraph.GraphPane.CurveList["Cadence"] as LineItem;

IPointListEdit cadenceList = lineOfCadence.Points as IPointListEdit;

cadenceList.Add(x, y3);

//Adding lineOfAltitude in zrcHRMGraph

LineItem lineOfAltitude = zrcHRMGraph.GraphPane.CurveList["Altitude"] as LineItem;

IPointListEdit altitudeList = lineOfAltitude.Points as IPointListEdit;

altitudeList.Add(x, y4);

//Adding lineOfPower in zrcGraph

LineItem lineOfPower = zrcHRMGraph.GraphPane.CurveList["Power"] as LineItem;

IPointListEdit powerList = lineOfPower.Points as IPointListEdit;

powerList.Add(x, y5);

AxisChangeZedGraph(zrcHRMGraph);

Thread.Sleep(700);

}

}

catch (Exception ex)

{

// MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Funtion to display Heart Rate in Summary

///

/// </summary>

/// <param name="heartRate"></param>

private void HRDataHeartRate(string heartRate)

{

try

{

if (lblHeartRate.InvokeRequired)

{

HRDataMethod hrDataMethod = new HRDataMethod(HRDataHeartRate);

this.Invoke(hrDataMethod, new object[] { heartRate });

}

else

{

lblHeartRate.Text = heartRate;

}

}

catch (InvalidAsynchronousStateException)

{

DataAnalysisSystemPanel asp = new DataAnalysisSystemPanel();

asp.Close();

//throw;

}

}

/// <summary>

/// Thead for fetch Heart Rate

/// </summary>

public void HeartRateThread()

{

try

{

string hrmData = File.ReadAllText(fileName);

string[] hrmDataArr = hrmData.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);

int indexOfHRMDataArr = Array.IndexOf(hrmDataArr, "[HRData]");

heartRates = new string[hrmDataArr.Length - indexOfHRMDataArr];

int j = 0;

for (int i = indexOfHRMDataArr + 1; i < hrmDataArr.Length - 1; i++)

{

string hrData = hrmDataArr[i];

string[] hrDataArr = Regex.Split(hrData, @"\W+");

heartRates[j] = hrDataArr[0];

this.HRDataHeartRate(heartRates[j].ToString());

Thread.Sleep(700);

j++;

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Fucntion to Display Value of Speed in the Summary

/// </summary>

/// <param name="speed"></param>

private void HRDataSpeed(string speed)

{

try

{

if (lblSpeed.InvokeRequired)

{

HRDataMethod hrDataMethod = new HRDataMethod(HRDataSpeed);

this.Invoke(hrDataMethod, new object[] { speed });

}

else

{

lblSpeed.Text = speed;

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Thread to fetch Speed

/// </summary>

private void SpeedThread()

{

try

{

string hrmData = File.ReadAllText(fileName);

string[] hrmDataArr = hrmData.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);

int indexOfHRMDataArr = Array.IndexOf(hrmDataArr, "[HRData]");

speeds = new string[hrmDataArr.Length - indexOfHRMDataArr];

int j = 0;

for (int i = indexOfHRMDataArr + 1; i < hrmDataArr.Length - 1; i++)

{

string hrData = hrmDataArr[i];

string[] hrDataArr = Regex.Split(hrData, @"\W+");

speeds[j] = hrDataArr[1];

this.HRDataSpeed(speeds[j].ToString());

Thread.Sleep(700);

j++;

}

}

catch ( Exception ex)

{

// MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Function to show values of candence in the summary

/// </summary>

/// <param name="cadence"></param>

private void HRDataCadence(string cadence)

{

try

{

if (lblCadence.InvokeRequired)

{

HRDataMethod hrDataMethod = new HRDataMethod(HRDataCadence);

this.Invoke(hrDataMethod, new object[] { cadence });

}

else

{

lblCadence.Text = cadence;

}

}

catch (Exception ex)

{

}

}

/// <summary>

/// Thread to Fetch the Value of Cadence

/// </summary>

private void CadenceThread()

{

try

{

string hrmData = File.ReadAllText(fileName);

string[] hrmDataArr = hrmData.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);

int indexOfHRMDataArr = Array.IndexOf(hrmDataArr, "[HRData]");

cadences = new string[hrmDataArr.Length - indexOfHRMDataArr];

int j = 0;

for (int i = indexOfHRMDataArr + 1; i < hrmDataArr.Length - 1; i++)

{

string hrData = hrmDataArr[i];

string[] hrDataArr = Regex.Split(hrData, @"\W+");

cadences[j] = hrDataArr[2];

this.HRDataCadence(cadences[j].ToString());

Thread.Sleep(700);

j++;

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

}

private void chkSpeed\_CheckedChanged(object sender, EventArgs e)

{

if (chkSpeed.Checked == true)

{

lineOfSpeed.IsVisible = true;

chkSpeed.Invalidate();

}

else

{

lineOfSpeed.IsVisible = false;

chkSpeed.Invalidate();

}

}

private void chkCadence\_CheckedChanged(object sender, EventArgs e)

{

if (chkCadence.Checked == true)

{

lineOfCadence.IsVisible = true;

chkCadence.Invalidate();

}

else

{

lineOfCadence.IsVisible = false;

chkCadence.Invalidate();

}

}

private void chkAltitude\_CheckedChanged(object sender, EventArgs e)

{

if (chkAltitude.Checked == true)

{

lineOfAltitude.IsVisible = true;

chkAltitude.Invalidate();

}

else

{

lineOfAltitude.IsVisible = false;

chkAltitude.Invalidate();

}

}

private void chkHeartRate\_CheckedChanged(object sender, EventArgs e)

{

if (chkHeartRate.Checked == true)

{

lineOfHeartRate.IsVisible = true;

chkHeartRate.Invalidate();

}

else

{

lineOfHeartRate.IsVisible = false;

chkHeartRate.Invalidate();

}

}

private void chkPower\_CheckedChanged(object sender, EventArgs e)

{

if (chkPower.Checked == true)

{

lineOfPower.IsVisible = true;

chkPower.Invalidate();

}

else

{

lineOfPower.IsVisible = false;

chkPower.Invalidate();

}

}

/// <summary>

/// Function to show altitude in data summary

/// </summary>

/// <param name="altitude"></param>

private void HRDataAltitude(string altitude)

{

try

{

if (lblAltitude.InvokeRequired)

{

HRDataMethod hrDataMethod = new HRDataMethod(HRDataAltitude);

this.Invoke(hrDataMethod, new object[] { altitude });

}

else

{

lblAltitude.Text = altitude;

}

}

catch (Exception ex)

{

// MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Thread to fetch Altitude

/// </summary>

private void AltitudeThread()

{

try

{

string hrmData = File.ReadAllText(fileName);

string[] hrmDataArr = hrmData.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);

int indexOfHRMDataArr = Array.IndexOf(hrmDataArr, "[HRData]");

altitudes = new string[hrmDataArr.Length - indexOfHRMDataArr];

int j = 0;

for (int i = indexOfHRMDataArr + 1; i < hrmDataArr.Length - 1; i++)

{

string hrData = hrmDataArr[i];

string[] hrDataArr = Regex.Split(hrData, @"\W+");

altitudes[j] = hrDataArr[3];

this.HRDataAltitude(altitudes[j].ToString());

Thread.Sleep(700);

j++;

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Function to show power in the summary

/// </summary>

/// <param name="power"></param>

private void HRDataPower(string power)

{

try

{

if (lblPower.InvokeRequired)

{

HRDataMethod hrDataMethod = new HRDataMethod(HRDataPower);

this.Invoke(hrDataMethod, new object[] { power });

}

else

{

lblPower.Text = power;

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Thread for Power

/// </summary>

private void PowerThread()

{

try

{

string hrmData = File.ReadAllText(fileName);

string[] hrmDataArr = hrmData.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);

int indexOfHRMDataArr = Array.IndexOf(hrmDataArr, "[HRData]");

powers = new string[hrmDataArr.Length - indexOfHRMDataArr];

int j = 0;

for (int i = indexOfHRMDataArr + 1; i < hrmDataArr.Length - 1; i++)

{

string hrData = hrmDataArr[i];

string[] hrDataArr = Regex.Split(hrData, @"\W+");

powers[j] = hrDataArr[4];

this.HRDataPower(powers[j].ToString());

Thread.Sleep(700);

j++;

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

}

}

}

## Backend of Main Form:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace DataAnalysisSoftware

{

public partial class DataAnalysisSystemPanel : Form

{

public DataAnalysisSystemPanel()

{

InitializeComponent();

}

/// <summary>

/// Setting Initial Params values

/// </summary>

string version = "1.0";

string monitor = "0";

string sMode = "00000000";

string date = "20181123";

string startTime = "00:00:00.0";

string length = "00:00:00.0";

string interval = "0";

string upper1 = "160";

string lower1 = "80";

string upper2 = "160";

string lower2 = "80";

string upper3 = "160";

string lower3 = "80";

string timer1 = "00:00";

string timer2 = "00:00";

string timer3 = "00:00";

string activeLimit = "0";

string maxHR = "192";

string restHR = "52";

string startDelay = "300";

string vo2Max = "50";

string weight = "75";

/// <summary>

/// Setting Initial [HRData] value

/// </summary>

private string fileName;

// string singleLine;

// IDictionary<string, string> Params = new Dictionary<string, string>();

ParameterClass pc = new ParameterClass();

private void DataAnalysisSystemPanel\_Load(object sender, EventArgs e)

{

//Setting date formate to YYYY/MM/DD

date = date.Insert(4, "/");///Adding foreslash after 4 digit in date

date = date.Insert(7, "/");///Adding foreslash after 7 digit in date

pc.Version = version;

pc.Monitor = monitor;

pc.SMode = sMode;

pc.Date = date;

pc.StartTime = startTime;

pc.Length = length;

pc.Interval = interval;

pc.Upper1 = upper1;

pc.Lower1 = lower1;

pc.Upper2 = upper2;

pc.Lower2 = lower2;

pc.Upper3 = upper3;

pc.Lower3 = lower3;

pc.Timer1 = timer1;

pc.Timer2 = timer2;

pc.Timer3 = timer3;

pc.ActiveLimit = activeLimit;

pc.MaxHR = maxHR;

pc.RestHR = restHR;

pc.StartDelay = startDelay;

pc.VO2max = vo2Max;

pc.Weight = weight;

}

/// <summary>

/// /Open Dialog to search hrm file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnLoadData\_Click(object sender, EventArgs e)

{

OpenFileDialog opd = new OpenFileDialog();

opd.Title = "Open .HRM file";

opd.Filter = "HRM files (\*.hrm|\*.hrm|All files (\*.\*)|\*.\*";

if (opd.ShowDialog() == DialogResult.OK)

{

txtFileName.Text = opd.FileName;

fileName = opd.FileName;

DialogResult dr = MessageBox.Show(fileName + " has been successfully loaded", "Do you like to view the data", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (dr == DialogResult.Yes)

{

FetchAndLoadParamsData();

LoadDataSummaryForm();

}

else

{

return;

}

}

else

{

MessageBox.Show("No File Selected");

}

FetchAndLoadParamsData();

// FetchAndLoadHRMData();

}

/// <summary>

/// Add Chart Mode in the Content Panel

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnChart\_Click(object sender, EventArgs e)

{

try

{

if (txtFileName.Text=="")

{

MessageBox.Show("You need to choose Polar HRM File to view data");

btnLoadData.Focus();

}

else

{

pnlContent.Controls.Clear(); //Clear Existing controls

Chart chart = new Chart();

chart.fileName = fileName;

chart.TopLevel = false;

pnlContent.Controls.Add(chart);

chart.FormBorderStyle = FormBorderStyle.None;

chart.Dock = DockStyle.Fill;

chart.Show();

}

}

catch ( Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void btnExit\_Click(object sender, EventArgs e)

{

this.Close();

}

/// <summary>

/// Add Data Summary in Content Panel

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnDataSummary\_Click(object sender, EventArgs e)

{

try

{

if (txtFileName.Text=="")

{

MessageBox.Show("You need to choose Polar HRM File to view data");

btnLoadData.Focus();

}

else

{

LoadDataSummaryForm();

//ShowOrHidePowerBySMode();

}

}

catch ( Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void LoadDataSummaryForm()

{

pnlContent.Controls.Clear();

DataSummary summary = new DataSummary();

summary.TopLevel = false;

pnlContent.Controls.Add(summary);

summary.FormBorderStyle = FormBorderStyle.None;

summary.fileName = fileName;

summary.Dock = DockStyle.Fill;

summary.version = pc.Version;

summary.monitor = pc.Monitor;

summary.sMode = pc.SMode;

summary.date = pc.Date;

summary.startTime = pc.StartTime;

summary.length = pc.Length;

summary.interval = pc.Interval;

summary.upper1 = pc.Upper1;

summary.upper2 = pc.Upper2;

summary.upper3 = pc.Upper3;

summary.lower1 = pc.Lower1;

summary.lower2 = pc.Lower2;

summary.lower3 = pc.Lower3;

summary.timer1 = pc.Timer1;

summary.timer2 = pc.Timer2;

summary.timer3 = pc.Timer3;

summary.activeLimit = pc.ActiveLimit;

summary.maxHR = pc.MaxHR;

summary.restHR = pc.RestHR;

summary.startDelay = pc.StartDelay;

summary.vo2Max = pc.VO2max;

summary.weight = pc.Weight;

summary.Show();

}

/// <summary>

/// Method to load Parameter (Params) of hrm file to InitialParams variables

/// </summary>

/\* public void LoadParametersToInitialParams()

{

version = Params["Version"];

monitor = Params["Monitor"];

sMode = Params["SMode"];

date = Params["Date"];

startTime = Params["StartTime"];

length = Params["Length"];

interval = Params["Interval"];

upper1 = Params["Upper1"];

lower1 = Params["Lower1"];

upper2 = Params["Upper2"];

lower2 = Params["Lower2"];

upper3 = Params["Upper3"];

lower3 = Params["Lower3"];

timer1 = Params["Timer1"];

timer2 = Params["Timer2"];

timer3 = Params["Timer3"];

activeLimit = Params["ActiveLimit"];

maxHR = Params["MaxHR"];

restHR = Params["RestHR"];

startDelay = Params["StartDelay"];

vo2Max = Params["VO2max"];

weight = Params["Weight"];

}

\*/

private void FetchAndLoadParamsData()

{

try

{

if (txtFileName.Text!="")

{

StreamReader reader = new StreamReader(txtFileName.Text, Encoding.Default);

string textOfHRMFile = reader.ReadToEnd();

//find 'Version=' in hrm file and load its data to version variable

int indexOfVersion = textOfHRMFile.IndexOf("Version=");//load the 'Version=' from hrm file

version = textOfHRMFile.Substring(indexOfVersion + 8, 3); //Retrive string of last 3 letter after 8 letter of this index i.e. indexOfVersion

pc.Version = version;//load the fetch value of Version in the Version Function

//find 'Monitor=' and Load its data to monitor

int indexOfMonitor = textOfHRMFile.IndexOf("Monitor=");

monitor = textOfHRMFile.Substring(indexOfMonitor + 8, 2);

pc.Monitor = monitor;

//find 'SMode=' and load its date to sMode

int indexOfSMode = textOfHRMFile.IndexOf("SMode=");

sMode = textOfHRMFile.Substring(indexOfSMode + 6, 9);

pc.SMode = sMode;

//finding 'Date=' and load its data to date

int indexOfDate = textOfHRMFile.IndexOf("Date=");

date = textOfHRMFile.Substring(indexOfDate + 5, 8);

date = date.Insert(4, "-");//Added Dash after 4th letter i.e.YYYYMMDD to YYYY-MMDD

date = date.Insert(7, "-");//Added Dash after 7th letter i.e YYYY-MMDD to YYYY-MM-DD

pc.Date = date;

//finding 'StartTime=' and load its data to startTime

int indexOfStartTime = textOfHRMFile.IndexOf("StartTime=");

startTime = textOfHRMFile.Substring(indexOfStartTime + 10, 10);

pc.StartTime = startTime;

//finding 'Length=' and load its data to length

int indexOfLength = textOfHRMFile.IndexOf("Length=");

length = textOfHRMFile.Substring(indexOfLength + 7, 10);

pc.Length = length;

//finding 'Interval=' and load its data to interval

int indexOfInterval = textOfHRMFile.IndexOf("Interval=");

interval = textOfHRMFile.Substring(indexOfInterval + 9, 1);

pc.Interval = interval;

//finding 'Upper1=' and load its data to upper1 and finally to Upper1

int indexOfUpper1 = textOfHRMFile.IndexOf("Upper1=");

upper1 = textOfHRMFile.Substring(indexOfUpper1 + 7, 1);

pc.Upper1 = upper1;

//finding 'Upper2=' and load its data to upper2 and finally to Upper2

int indexOfUpper2 = textOfHRMFile.IndexOf("Upper2=");

upper2 = textOfHRMFile.Substring(indexOfUpper2 + 7, 1);

pc.Upper2 = upper2;

//finding 'Upper3=' and load its data to upper3 and finally to Upper3

int indexOfUpper3 = textOfHRMFile.IndexOf("Upper3=");

upper3 = textOfHRMFile.Substring(indexOfUpper3 + 7, 1);

pc.Upper3 = upper3;

//finding 'Lower1=' and load its data to lower1 and finally to Lower1

int indexOfLower1 = textOfHRMFile.IndexOf("Lower1=");

lower1 = textOfHRMFile.Substring(indexOfLower1 + 7, 1);

pc.Lower1 = lower1;

//finding 'Lower2=' and load its data to lower2 and finally to Lower2

int indexOfLower2 = textOfHRMFile.IndexOf("Lower2=");

lower2 = textOfHRMFile.Substring(indexOfLower2 + 7, 1);

pc.Lower2 = lower2;

//finding 'Lower3=' and load its data to lower3 and finally to Lower3

int indexOfLower3 = textOfHRMFile.IndexOf("Lower3=");

lower3 = textOfHRMFile.Substring(indexOfLower3 + 7, 1);

pc.Lower3 = lower3;

//finding 'Timer1=' and load its data to timer1 and finally to Timer1

int indexOfTimer1 = textOfHRMFile.IndexOf("Timer1=");

timer1 = textOfHRMFile.Substring(indexOfTimer1 + 7, 10);

pc.Timer1 = timer1;

//finding 'Timer2=' and load its data to timer2 and finally to Timer2

int indexOfTimer2 = textOfHRMFile.IndexOf("Timer2=");

timer2 = textOfHRMFile.Substring(indexOfTimer2 + 7, 10);

pc.Timer2 = timer2;

//finding 'Timer3=' and load its data to timer3 and finally to Timer3

int indexOfTimer3 = textOfHRMFile.IndexOf("Timer3=");

timer3 = textOfHRMFile.Substring(indexOfTimer1 + 7, 10);

pc.Timer3 = timer3;

//finding 'ActiveLimit=' and load its data to activeLimit and then pass it to ActiveLimit

int indexOfActiveLimit = textOfHRMFile.IndexOf("ActiveLimit=");

activeLimit = textOfHRMFile.Substring(indexOfActiveLimit + 12, 1);

pc.ActiveLimit = activeLimit;

//finding 'MaxHR' and load its data to maxHR and finally to MaxHR

int indexOfMaxHR = textOfHRMFile.IndexOf("MaxHR=");

maxHR = textOfHRMFile.Substring(indexOfMaxHR + 6, 3);

pc.MaxHR = maxHR;

//finding 'RestHR' and load its data to restHR and finally to RestHR

int indexOfRestHR = textOfHRMFile.IndexOf("RestHR=");

restHR = textOfHRMFile.Substring(indexOfRestHR + 7, 2);

pc.RestHR = restHR;

//finding 'StartDelay=' and load its data to startDelay and finally to StartDelay

int indexOfStartDelay = textOfHRMFile.IndexOf("StartDelay=");

startDelay = textOfHRMFile.Substring(indexOfStartDelay + 11, 3);

pc.StartDelay = startDelay;

//finding 'VO2max=' and load its data to vo2Max and finally to VO2max

int indexOfVO2max = textOfHRMFile.IndexOf("VO2max=");

vo2Max = textOfHRMFile.Substring(indexOfVO2max + 7, 2);

pc.VO2max = vo2Max;

//finding 'Weight=' and load its data to weight and finally to Weight

int indexOfWeight = textOfHRMFile.IndexOf("Weight=");

weight = textOfHRMFile.Substring(indexOfWeight + 7, 3);

pc.Weight = weight;

}

}

catch (NullReferenceException ex)

{

MessageBox.Show("No hrm data was found");

}

}

}

}