# GRAPHICAL USER INTERFACE

A Graphical User Interface (GUI) is designed using Microsoft Visual Studio, programmed with C# programming language. This interface is used to send calibrated fuel probe chart data to first microcontroller, log output height, time-period data from second microcontroller and save it as a txt file.

## Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

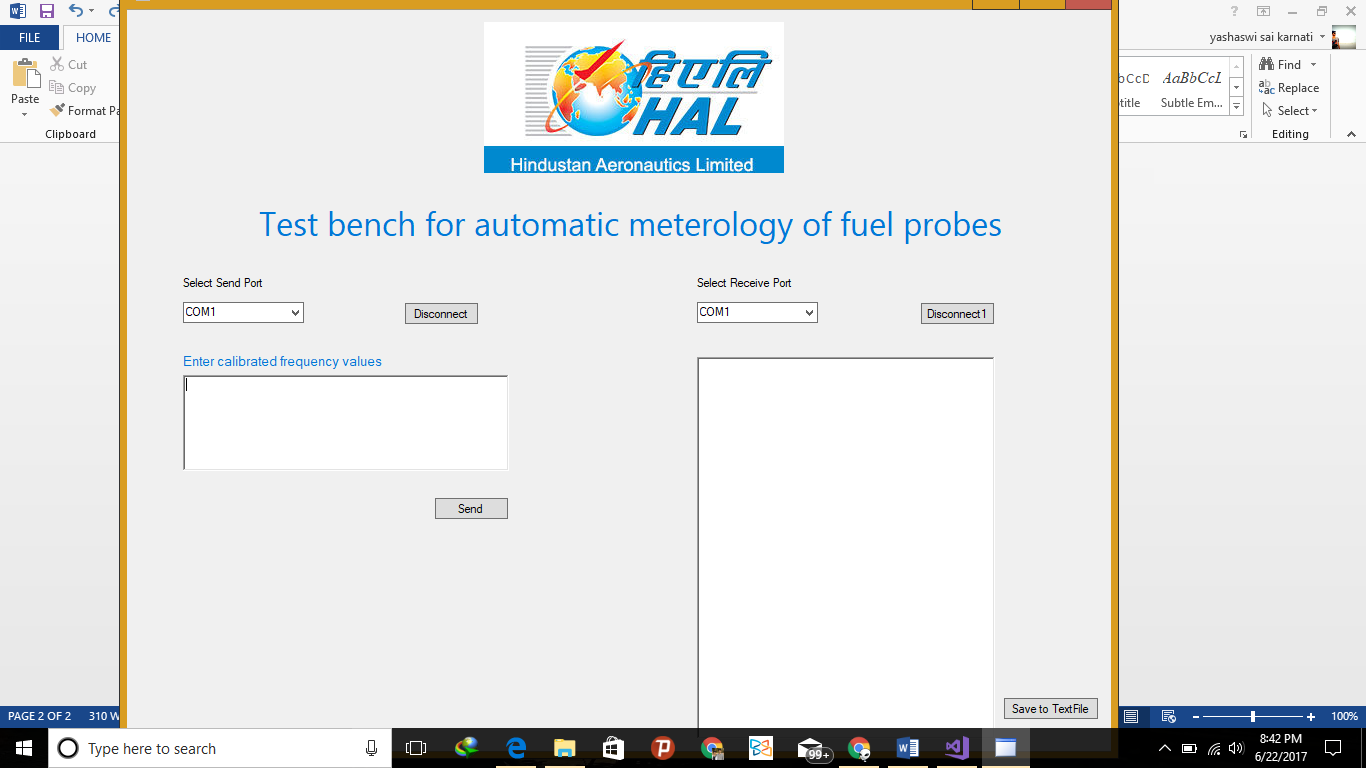
## Visual C#

C# (pronounced as see sharp) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is a general-purpose, object-oriented programming language.[12] Its development team is led by Anders Hejlsberg. The most recent version is C# 7.0 which was released in 2017 along with Visual Studio 2017.

## 

## Hands on GUI



On the left section of interface select ‘COM’ port to which first microcontroller is connected. Enter calibrated timeperiod values of fuelprobe seperated by comma and click send. On the right section select ‘COM’ port to which second microcontroller is connected. Expermental values of time period to the particular setpoint heights of different probes will be logged in the space provided. After completion of probe metrology click on the button “Save to TextFile”. The experimental calibration chart will be stored as.txt file in the location C:\testdata.

## Source code for GUI

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.Ports;

using System.IO;

namespace TestPortsApp

{

public partial class TestForm : Form

{

private static SerialPort m\_serialPort;

private static SerialPort m\_serialPortReceive;

private const string Tx = "Tx >> ";

private const string Delimiter = "\r";

int m = 1;

public TestForm()

{

InitializeComponent();

LoadPorts();

}

public void LoadPorts()

{

string[] availablePorts = {"COM1","COM2","COM3", "COM4", "COM5", "COM6", "COM7", "COM8", "COM9", "COM10" };// SerialPort.GetPortNames();

string[] availableReceivePorts = { "COM1", "COM2","COM3", "COM4", "COM5", "COM6", "COM7", "COM8", "COM9", "COM10" };// SerialPort.GetPortNames();

if (availablePorts.Count() > 0)

{

ListPort.DataSource = availablePorts;

listReceivePort.DataSource = availableReceivePorts;

}

}

private void button1\_Click(object sender, EventArgs e)

{

string strInput = txtInput.Text;

string[] values = strInput.Split(',');

foreach (string input in values)

{

//label2.Text += input;

if (ConnectSendPort(ListPort.SelectedValue.ToString(), 9600))

{

SendData(Decimal.Parse(input));

//label2.Text += "Ports Available" + ListPort.SelectedValue.ToString() ;

}

else

{

//label2.Text += "No Ports Available";

}

}

// Receive Data

ConnectReceivePort();

}

// Send Data

void SendData(decimal strValue)

{

if ((m\_serialPort != null) && m\_serialPort.IsOpen)

{

m\_serialPort.DiscardOutBuffer();

m\_serialPort.Write(strValue + Delimiter);

//OnDataSend(Tx + strValue + Delimiter);

}

}

//Connect port

public bool ConnectSendPort(string portName, int baudRate)

{

if (m\_serialPort == null)

{

m\_serialPort = new SerialPort

{

PortName = portName,

BaudRate = baudRate,

Parity = Parity.None,

StopBits = StopBits.One,

DataBits = 8,

Handshake = Handshake.None,

ReadTimeout = 500, //500 ms

};

}

else

{

CloseSerialPort(m\_serialPort);

m\_serialPort.PortName = portName;

m\_serialPort.BaudRate = baudRate;

}

OpenSerialPort();

return IsPortOpen;

}

public void ConnectReceivePort()

{

m\_serialPortReceive = new SerialPort(listReceivePort.SelectedValue.ToString());

m\_serialPortReceive.BaudRate = 9600;

m\_serialPortReceive.Parity = Parity.None;

m\_serialPortReceive.StopBits = StopBits.One;

m\_serialPortReceive.DataBits = 8;

m\_serialPortReceive.Handshake = Handshake.None;

m\_serialPortReceive.RtsEnable = true;

m\_serialPortReceive.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler);

m\_serialPortReceive.Open();

string indata = m\_serialPortReceive.ReadExisting();

txtDataReceive.Text += indata;

}

private void DataReceivedHandler(

object sender,

SerialDataReceivedEventArgs e)

{

SerialPort sp = (SerialPort)sender;

string indata = sp.ReadExisting();

txtDataReceive.Text += indata;

//Console.WriteLine("Data Received:");

//Console.Write(indata);

}

private static void OpenSerialPort()

{

try

{

m\_serialPort.DataReceived += SerialPortDataReceived;

m\_serialPort.Open();

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex.Message + ex.StackTrace);

}

}

private void CloseSerialPort(SerialPort portName)

{

if (portName.IsOpen)

{

portName.DataReceived -= SerialPortDataReceived;

portName.Close();

}

}

private static void SerialPortDataReceived(object sender, SerialDataReceivedEventArgs e)

{

var serialPort = (SerialPort)sender;

var data = serialPort.ReadExisting();

//ProcessData(data);

}

/// <summary>

/// check is port open

/// </summary>

public static bool IsPortOpen

{

get

{

return (m\_serialPort != null && m\_serialPort.IsOpen);

}

}

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

{

}

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

{

ConnectReceivePort();

}

private void button2\_Click(object sender, EventArgs e)

{

if (m\_serialPort != null)

{

CloseSerialPort(m\_serialPort);

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (m\_serialPortReceive != null)

{

CloseSerialPort(m\_serialPortReceive);

}

}

private void button4\_Click(object sender, EventArgs e)

{

DateTime dt = DateTime.Now;

string strFileName = "OutPut\_" + dt.ToString("yyyyMMddHHmm");

TextWriter txt = new StreamWriter("C:\\testdata\\"+ strFileName + ".txt");

txt.Write(txtDataReceive.Text);

m++;

txt.Close();

}

}

}