Assignment report 2016

Candidates: Alexander Zhang Gjerseth

Title: C# coding assignment

Table of contents

Table of contents 2

1 Introduction 3

2 Results 4

2.1 Exercise 1 4

2.2 Exercise 2 4

2.3 Exercise 3 4

2.4 Exercise 4 5

2.5 Exercise 5 5

2.6 Exercise 6 5

2.7 Exercise 7 5

2.8 Exercise 8 6

2.9 Exercise 9 6

2.10 Exercise 10 6

2.11 Exercise 11 6

2.12 Exercise 12 7

2.13 Exercise 13 7

2.14 Exercise 14 7

2.15 Exercise 15 7

2.16 Exercise 8

3 Conclusion 9

Appendices 10

# Introduction

In this assignment the goal is to become familiar with object-oriented programming and the version control system git. The main task in the assignment is to develop an application that simulates a DAQ. This application will simulate sensor values and these values can be written to a CSV file.

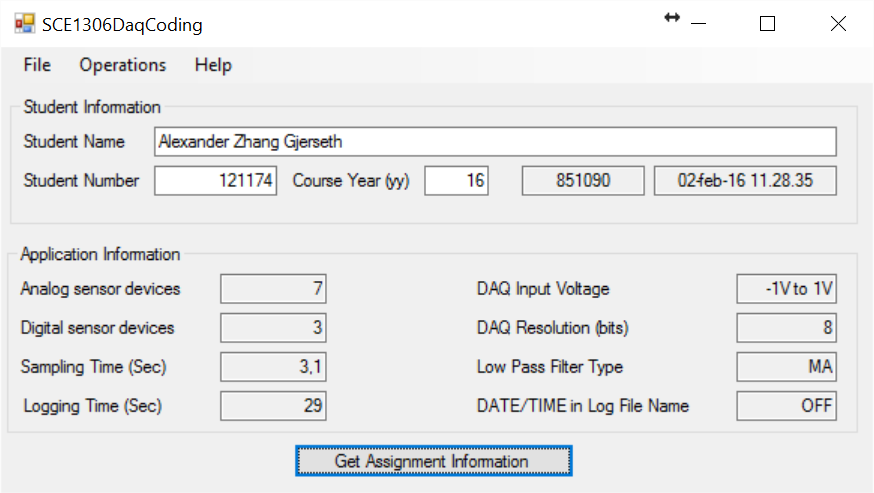


Figure 1.1 Screenshot of SCE1306 DaqCoding application containing the specifications for the DAQ simulator.

In Figure 1.1 the specifications for the DAQ simulator is given. The simulator is going to have 7 analog sensors with a signal between -1 and 1 volt and this signal is sampled with an 8 bit resolution. It will also have 3 digital sensors, which means that the signal is high or low. The sampling time is 3.1 seconds and logging time interval is 29 seconds.

# Results

## Exercise 1

A screenshot of the SCE1306 DaqCoding application is shown in Figure 1.1

## Exercise 2

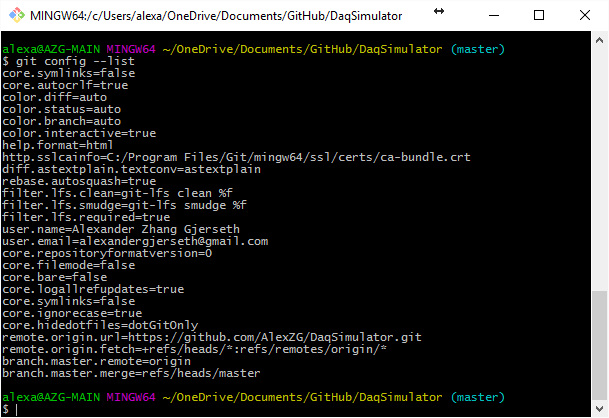


Figure 2.1 Screenshot of git bash showing current git configuration

Created a new folder inside the GitHub folder and configured the repository. The current git configuration is can be seen in Figure 2.1.

## Exercise 3



Figure 2.2 Screenshot of application files and path.

Figure 2.2 is showing the project file and project folder generated by the IDE. This file and folder are located inside the repository directory.

## Exercise 4

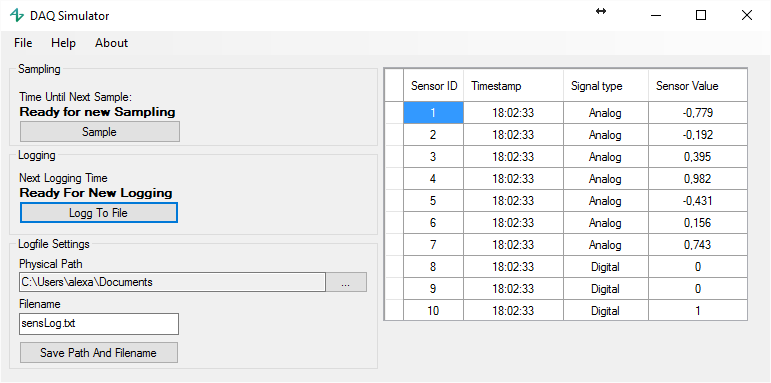


Figure 2.3 Screenshot of the GUI of the application.

Figure 2.3 is a screenshot of the graphical user interface for the application. This GUI is created according to the specifications given in the SCE1306 DaqCoding application. The text box described in the assignment is switched out with a data grid view, because it is a better way to display the data in an organized way.

## Exercise 5



Figure 2.4 Screenshot of GUI commit to GitHub.

To push the changes to git, Visual studio team explorer with GitHub was used. This is a graphical interface for git and needs no written commands. Figure 2.4 is a screenshot of the commit on the GitHub webpage.

## Exercise 6

Source code for the sensor class can be found in Appendix B on page …

## Exercise 7

Code for the sample button click event can be found in Appendix B on page… . This event calls methods created to get sensor values and sets the data source of the data grid view to the table contain the values. It also starts the countdown timer and disables itself. In the bottom of the method there is a call to an auto size method that enables a scrollbar if height of items in table is more than the height of the data grid view.

## Exercise 8

Digital sensors values is usually has two states, high and low. To sample this, the input is read and by the voltage, it is determined if the signal is high or low. Most commonly the signal is either 5 or 0 volt and then it is read as high and low respectively. It is also possible that the values are represented as a pulse train of high or low values with fixed pulse width.

## Exercise 9



Figure 2.5 Screenshot of git commit congaing the code for the sample button event

Figure 2.5 is a screenshot of the commit that contains the code for the sample button event.

## Exercise 10

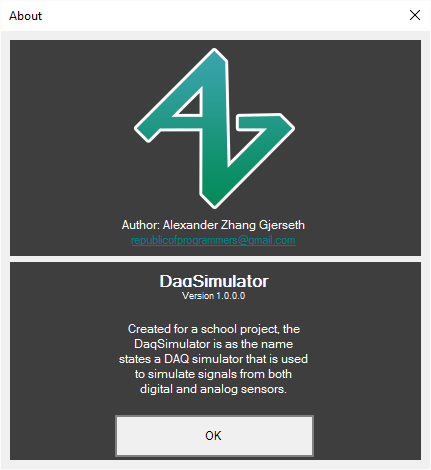


Figure 2.6 Screenshot of about form

Choose to create a about form. This can be opened from the main form by clicking the about button in the applications menu strip. A screenshot of this form is shown in Figure 2.6.

## Exercise 11

Code for the logging button event can be found in Appendix B on page .. . This event calls a method that sets the filename and path of the CSV file and then writes the values to a .txt file. It is possible to set the path and filename from the main form. This method also starts the countdown timer for logging and disables itself. This prevents the user from clicking the button until the timer reaches zero and the timer tick event enables the logging button again.

## Exercise 12

## Exercise 13



Figure 2.7 Screenshot of the latest git commit

Figure 2.7 contains a screenshot of the latest commit synced to the GitHub repository.

## Exercise 14

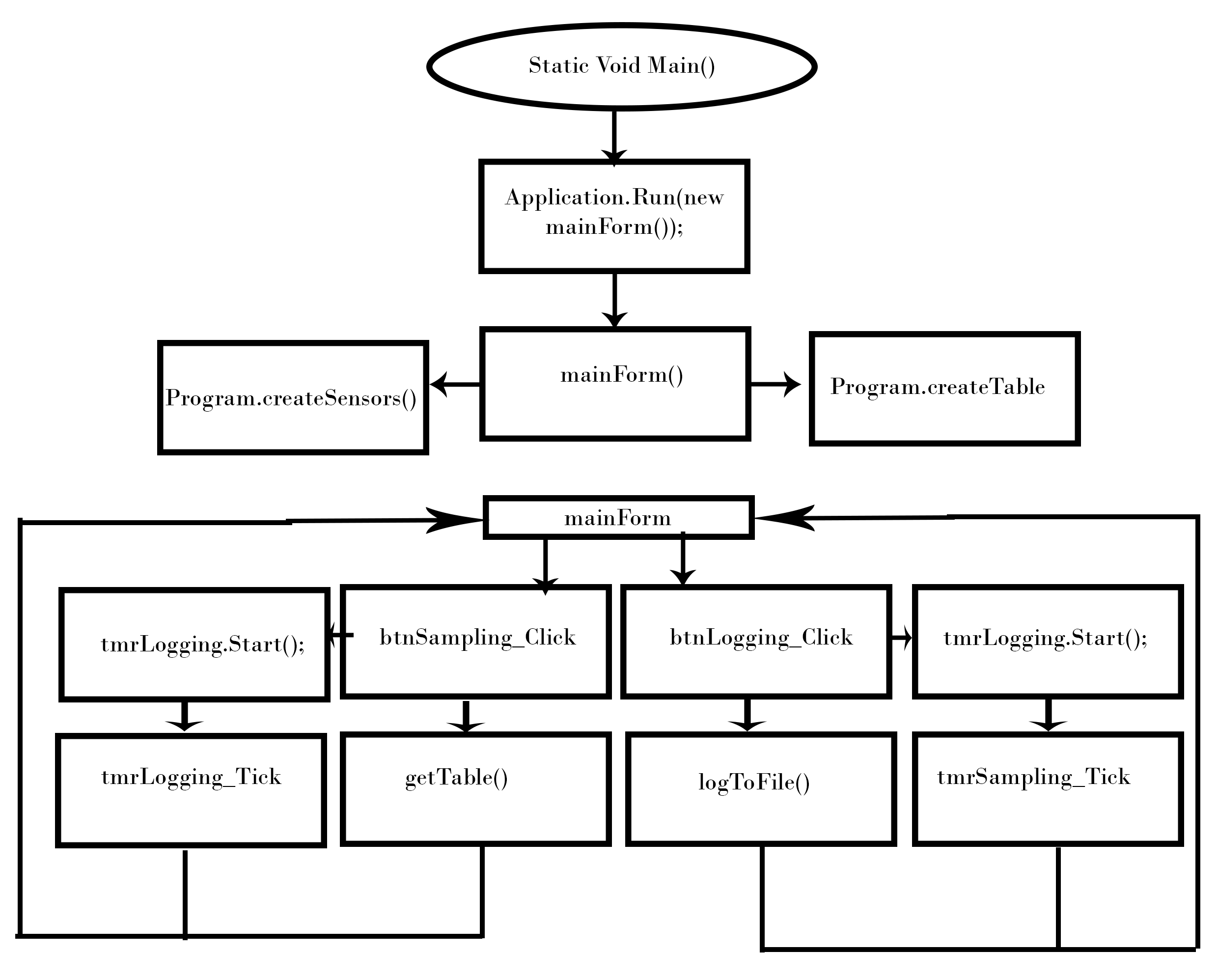


Figure 2.8 Flow diagram of the application

Figure 2.8 shows a basic sketch of a flow diagram for the application.

## Exercise 15

The classes in the program is Aboutbox.cs,GUI.cs,Program.cs,Sensor.cs and anaSensor and diSensor. anaSensor and digSensor is child classes of the sensor class. So 6 classes in total. The number of anaSensor and digSensor objects is 10. It is also a DataTable object in the application.

## Exercise 16

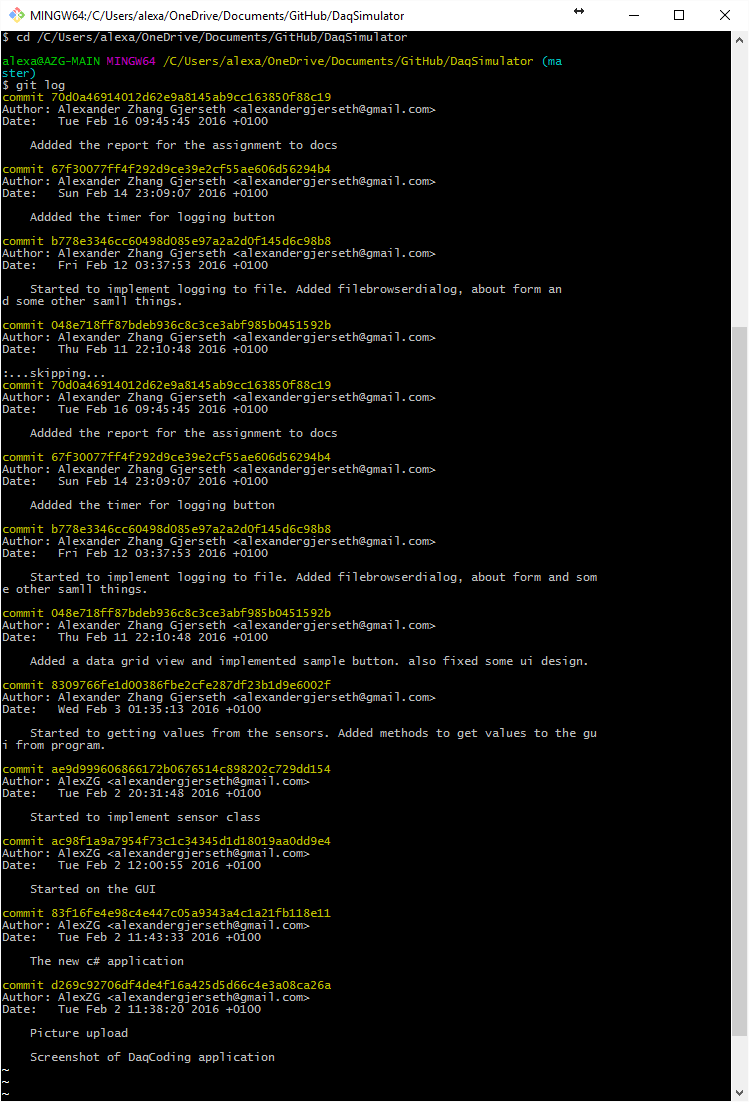


Figure 2.9 Screenshot of git repository commit history.

# Conclusion

The result of the assignment is a working DAQ simulator. It is possible to both create sensor values inside a given range, chose digital or analog sensor and write the values to a CSV file. The application has limited features, but it is possible to add more in the future. The application covers the specifications given in the assignment. The whole project can be found on GitHub: <https://github.com/AlexZG/DaqSimulator>

Appendices

Appendix 1: Source Code

Appendix 2: sensorLog.txt

Appendix A: Source Code

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 DaqSimulator.Properties;

namespace DaqSimulator

{

public partial class mainForm : Form

{

private Sensor[] sensObj;

private DataTable dataTbl;

private bool first;

private double dfltTimeSample;

private double dfltTimeLogging;

private double timeLeftSample;

private double timeLeftLogging;

private static string path;

private static string fileName;

public mainForm()

{

InitializeComponent();

first = true;

dfltTimeSample = Settings.Default.sampleTime\*10;

dfltTimeLogging = Settings.Default.dfltTimeLogging \* 10;

timeLeftSample = dfltTimeSample;

timeLeftLogging = 290;

sensObj = Program.createSensors();

dataTbl = Program.createTable();

path = Settings.Default.filePath;

fileName = Settings.Default.fileName;

setupDesign();

}

//When btn sampling has been clicked it calls the method getTable and sets the value of the

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

{

tmrCountSample.Start();

dataTbl = getTable(dataTbl);

dgvData.DataSource = dataTbl;

btnSampling.Enabled = false;

if (first == true)

{

autoHeightDGV();

}

}

//Gets values from program.filldata method

private DataTable getTable(DataTable table)

{

table.Clear();

table = Program.fillData(table, sensObj);

return table;

}

//Sets up the design of the gui

private void setupDesign()

{

dgvData.DataSource = dataTbl;

dgvData.AllowUserToAddRows = false;

dgvData.AllowUserToResizeColumns = false;

dgvData.AllowUserToResizeRows = false;

dgvData.EditMode = DataGridViewEditMode.EditProgrammatically;

dgvData.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

dgvData.ScrollBars = ScrollBars.None;

dgvData.RowHeadersWidth = 20;

dgvData.RowHeadersDefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

dgvData.Columns[0].Width = 60;

dgvData.Columns[0].HeaderText = "Sensor ID";

dgvData.Columns[1].Width = 100;

dgvData.Columns[1].HeaderText = "Timestamp";

dgvData.Columns[2].Width = 85;

dgvData.Columns[2].HeaderText = "Signal type";

dgvData.Columns[3].Width = 100;

dgvData.Columns[3].HeaderText = "Sensor Value";

txtPath.Text = Settings.Default.filePath;

txtFileName.Text = Settings.Default.fileName;

}

//Method for rezise the data grid view

private void autoHeightDGV()

{

int sum = this.dgvData.ColumnHeadersHeight;

foreach (DataGridViewRow row in dgvData.Rows)

{

sum += row.Height;

}

if (sum + 60 >= this.Height)

{

dgvData.Height = this.Height - 100;

dgvData.ScrollBars = ScrollBars.Vertical;

}

else

{

dgvData.Height = sum;

}

first = false;

}

//Timer tick event for the count sample timer, this checks if time left is bigger than zero

private void tmrCountSample\_Tick(object sender, EventArgs e)

{

if (timeLeftSample > 0)

{

timeLeftSample = timeLeftSample - 1;

lblCountSample.Text = (timeLeftSample / 10).ToString("F1") + " Seconds";

}

else

{

tmrCountSample.Stop();

timeLeftSample = dfltTimeSample;

lblCountSample.Text = "Ready for new Sampling";

btnSampling.Enabled = true;

}

}

//Just tried to implement help box

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

{

if(itmEnableHelp.Checked == false)

{

itmEnableHelp.Checked = true;

}

else

{

itmEnableHelp.Checked = false;

}

}

//Event if the about item is clicked, opens the about form

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

{

AboutBox aboutWindow = new AboutBox();

aboutWindow.ShowDialog();

}

//Fills a string with the content of datatable

private static string dataTableToCSV(DataTable table,bool newFile)

{

StringBuilder sb = new StringBuilder();

char seperator = ',';

try {

if(newFile == true)

{

int count = 0;

foreach (DataColumn dr in table.Columns)

{

sb.Append(dr.ColumnName);

if (count < table.Columns.Count -1)

{

sb.Append(seperator);

}

count++;

}

sb.AppendLine();

newFile = false;

}

foreach (DataRow row in table.Rows)

{

string value = null;

for (int i = 0; i < table.Columns.Count; i++)

{

value = row[i].ToString();

if (value.Contains(","))

{

value = value.Replace(",", ".");

}

sb.Append(value);

if (i < table.Columns.Count - 1)

{

sb.Append(seperator);

}

}

sb.AppendLine();

}

}

catch(Exception ex)

{

MessageBox.Show(ex.ToString());

}

return sb.ToString();

}

//Event when button logging is clicked

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

{

tmrCountLogging.Start();

logToFile(dataTbl);

}

//Method that writes a string to file

private static void logToFile(DataTable dt)

{

try

{

path = Settings.Default.filePath;

fileName = Settings.Default.fileName;

if (string.IsNullOrEmpty(path))

{

path = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

}

if (string.IsNullOrEmpty(fileName))

{

fileName = "sensorLog.txt";

}

string filePath = Path.Combine(path, fileName);

string logTxt = null;

if (!File.Exists(filePath))

{

logTxt = dataTableToCSV(dt,true);

}

else

{

logTxt = dataTableToCSV(dt, false);

}

using (StreamWriter sw = File.AppendText(filePath))

{

sw.Write(logTxt);

}

}

catch(Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

//Code when file dialog browser is clicked, saves the selected path

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

{

DialogResult result = fbdPath.ShowDialog();

if(result == DialogResult.OK)

{

Settings.Default.filePath = fbdPath.SelectedPath;

txtPath.Text = fbdPath.SelectedPath;

}

}

private void mainForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Settings.Default.Save();

}

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

{

Settings.Default.filePath = fbdPath.SelectedPath;

Settings.Default.fileName = txtFileName.Text;

Settings.Default.Save();

MessageBox.Show(Settings.Default.fileName);

}

private void tmrCountLogging\_Tick(object sender, EventArgs e)

{

if (timeLeftLogging > 0)

{

timeLeftLogging--;

lblCountLogging.Text = (timeLeftLogging / 10).ToString("F1") + " Seconds";

}

else

{

tmrCountLogging.Stop();

timeLeftLogging = dfltTimeLogging;

lblCountLogging.Text = "Ready for new Sampling";

btnLogging.Enabled = true;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data;

using DaqSimulator.Properties;

namespace DaqSimulator

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

///

private static int nAna { get; set; }

private static int nDig { get; set; }

private static int nSen { get; set; }

[STAThread]

static void Main(string[] args)

{

nAna = Settings.Default.numAna;

nDig = Settings.Default.numDig;

nSen = nAna + nDig;

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new mainForm());

}

public static Sensor[] createSensors()

{

Sensor[] senObj = new Sensor[nSen];

if (nAna != 0 && nDig != 0)

{

for (int i = 0; i < nAna; i++)

{

senObj[i] = new anaSensor();

}

for (int j = nAna; j < nAna + nDig; j++)

{

senObj[j] = new digSensor();

}

}

return senObj;

}

public static DataTable createTable()

{

DataTable table = new DataTable();

table.Columns.Add("sensorID", typeof(int));

table.Columns.Add("timeStamp", typeof(string));

table.Columns.Add("sensorType", typeof(string));

table.Columns.Add("sensorValue", typeof(double));

return table;

}

public static DataTable fillData(DataTable filltable, Sensor[] sensObj)

{

int count = 0;

foreach (Sensor element in sensObj)

{

DataRow row = filltable.NewRow();

row["sensorID"] = sensObj[count].GetSensId;

row["timeStamp"] = DateTime.Now.ToString("HH:mm:sss", new System.Globalization.CultureInfo("en-GB"));

row["sensorType"] = sensObj[count].getSensType();

row["sensorValue"] = sensObj[count].GetValue();

filltable.Rows.Add(row);

count++;

}

return filltable;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DaqSimulator.Properties;

namespace DaqSimulator

{

class Sensor

{

private static int currentID;

protected int sId { get; set; }

protected string sensType { get; set; }

protected double sVal { get; set; }

protected Random rSensVal { get; set; }

public Sensor()

{

sId = 0;

sensType = "Unspecified";

rSensVal = new Random(sId);

sVal = 0.0F;

}

static Sensor()

{

currentID = 0;

}

protected int getNextId()

{

return ++currentID;

}

public virtual double GetValue()

{

return sVal;

}

public int GetSensId

{

get

{

return sId;

}

}

public virtual string getSensType()

{

return sensType;

}

}

///////

class anaSensor : Sensor

{

protected double maxV { get; set; }

protected double minV { get; set; }

public anaSensor()

{

sId = getNextId();

sensType = "Analog";

rSensVal = new Random(sId);

sVal = 0.0F;

maxV = Settings.Default.maxVoltLim;

minV = Settings.Default.minVoltLim;

}

public override double GetValue()

{

sVal = Math.Round(rSensVal.NextDouble() \* (maxV - minV) + minV,3);

return sVal;

}

}

///////

class digSensor : Sensor

{

public digSensor()

{

sId = getNextId();

sensType = "Digital";

rSensVal = new Random(sId);

sVal = 0.0F;

}

public override double GetValue()

{

sVal = rSensVal.Next(0,2);

return sVal;

}

}

}

Appendix B: sensorLog.txt

sensorID,timeStamp,sensorType,sensorValue

1,17:10:28,Analog,-0.503

2,17:10:28,Analog,0.542

3,17:10:28,Analog,-0.413

4,17:10:28,Analog,0.632

5,17:10:28,Analog,-0.323

6,17:10:28,Analog,0.722

7,17:10:28,Analog,-0.234

8,17:10:28,Digital,1

9,17:10:28,Digital,0

10,17:10:28,Digital,1