using System;

using System.Windows.Forms;

using TChart = System.Windows.Forms.DataVisualization.Charting;

using System.Drawing;

using Npgsql;

namespace PiezoMath

{

public class PiezoParameters

{

/// <summary>

/// Gets or sets the type of the open sorting.

/// </summary>

/// <value>

/// The type of the open sorting.

/// </value>

public string OpenSortingType { get; set; }

/// <summary>

/// Gets or sets the type of the save sorting.

/// </summary>

/// <value>

/// The type of the save sorting.

/// </value>

public string SaveSortingType { get; set; }

/// <summary>

/// Gets or sets the measuring frequency.

/// </summary>

/// <value>

/// The measuring frequency.

/// </value>

public int MeasuringFrequency { get; set; }

….

public PiezoParameters()

{

gg = 0;

thicknes = 0.1;

diametr = 1.000;

MeasuringFrequency = 1000;

fr1 = 0;

fa1 = 0;

fr3 = 0;

….

/// <summary>

/// Maximum number of element from array.

/// </summary>

/// <param name="array\_in">The array in.</param>

/// <returns></returns>

public double MaxNumFromArray(double[] array\_in)

{

double max = array\_in[0];

int j=0;

for (int i = 0; i < array\_in.Length; i++)

{

if (max < array\_in[i])

{

j = i;

}

}

return j;

}

/// <summary>

/// Find the frequences for measurments from range.

/// </summary>

/// <returns>Frequency array</returns>

public double[] FindFreqs(int start, int end, int pointC)

{

double[] freqs = new double[pointC];

int x = (end - start) / pointC;

freqs[0] = start;

for (int i = 1; i < pointC; i++)

{

freqs[i] += x;

}

return freqs;

}

…

/// <summary>

/// Calculation d31

/// </summary>

/// <param name="eps">The eps.</param>

/// <param name="ro">The ro.</param>

/// <param name="Kp">The kp.</param>

/// <param name="d">The d.</param>

/// <param name="fr1">The fr.</param>

/// <returns></returns>

public double d31(double eps, double ro, double Kp, double d, int fr1)

{

double g, t;

t = (0.188 \* 2 \* Kp \* 1E-5) / (d \* fr1);

g = Math.Sqrt(eps / ro);

return (t \* g\*1e12);

}

/// <summary>

/// Databases the create table SQL command.

/// </summary>

/// <param name="DG">The dg.</param>

/// <param name="ConnectingString">The connecting string.</param>

/// <param name="TableName">Name of the table.</param>

/// <returns></returns>

public string DBCreateTableSQLCommand(DataGridView DG, string TableName)

{

string s = "";

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

{

if (DG.Columns[i].HeaderText == "id")

{

s = s + DG.Columns[i].HeaderText + " SERIAL PRIMATY KEY, ";

}

if (DG.Columns[i].HeaderText == "id\_section" ||

DG.Columns[i].HeaderText == "Direct" ||

DG.Columns[i].HeaderText == "Meas\_type" ||

DG.Columns[i].HeaderText == "operator")

{

s = s + DG.Columns[i].HeaderText + " char(255), ";

}

if (DG.Columns[i].HeaderText == "id\_sample")

{

s = s + DG.Columns[i].HeaderText + " INT, ";

}

if (DG.Columns[i].HeaderText == "Tsint\_K" ||

DG.Columns[i].HeaderText == "t\_cm" ||

DG.Columns[i].HeaderText == "d\_cm" ||

DG.Columns[i].HeaderText == "T\_K" ||

DG.Columns[i].HeaderText == "T\_K" ||

DG.Columns[i].HeaderText == "C\_pF" ||

DG.Columns[i].HeaderText == "e\_re" ||

DG.Columns[i].HeaderText == "tgd1e2" ||

DG.Columns[i].HeaderText == "e\_im" ||

DG.Columns[i].HeaderText == "tgd" ||

DG.Columns[i].HeaderText == "Y" ||

DG.Columns[i].HeaderText == "Ubias\_V" ||

DG.Columns[i].HeaderText == "Hbias\_T" ||

DG.Columns[i].HeaderText == "Timer\_disp"

)

{

s = s + DG.Columns[i].HeaderText + " double precision, ";

}

if (DG.Columns[i].HeaderText == "f\_Hz" ||

DG.Columns[i].HeaderText == "Cycle" ||

DG.Columns[i].HeaderText == "Timer")

{

s = s + DG.Columns[i].HeaderText + " INT, ";

}

if (DG.Columns[i].HeaderText == "Time")

{

s = s + DG.Columns[i].HeaderText + " time, ";

}

if (DG.Columns[i].HeaderText == "Date")

{

s = s + DG.Columns[i].HeaderText + " date, ";

}

}

string sql = "Create table " + TableName + " (" + s + " description text);";

return sql;

}

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.Threading;

using System.Windows.Forms;

using PiezoMath;

using System.Timers;

using Npgsql;

using System.IO;

using System.Diagnostics;

using System.Windows.Forms.DataVisualization;

using TChart = System.Windows.Forms.DataVisualization.Charting;

//это типа define

using ExcelObj = Microsoft.Office.Interop.Excel;

using NationalInstruments.NI4882;

namespace Kalipso

{

/// <summary>

/// Class for main form

/// </summary>

/// <seealso cref="System.Windows.Forms.Form" />

public partial class frmMain : Form

{

/// <summary>

/// The pp

/// </summary>

public PiezoParameters PP = new PiezoParameters();

/// <summary>

/// The COM

/// </summary>

public frmComPort Com = new frmComPort();

/// <summary>

/// The FRM database connection

/// </summary>

public frmDataBase frmDBConnection = new frmDataBase();

/// <summary>

/// The FRM m opt

/// </summary>

public frmMeasTempOpt frmMOpt = new frmMeasTempOpt();

/// <summary>

/// The FRM gpib

/// </summary>

public frmGPIBConfig frmGPIB = new frmGPIBConfig();

/// <summary>

/// Flag of hand measurments

/// </summary>

public frmMain()

{

InitializeComponent();

cmbTreatment.SelectedIndex = 0;

frmMOpt.Enabled = true;

hand = false;

}

….

private void btnCalcPiezo\_Click\_1(object sender, EventArgs e)

{

try

{

if (cmbOperator.SelectedIndex == -1)

{

MessageBox.Show("You must choice operator");

}

for (int i = 0; i < dGridPiezo.RowCount - 1; i++)

{

PiezoMathCalculation pm = new PiezoMathCalculation();

//расчет диэлектрической проницаемости до поляризации

if (dGridPiezo["C\_pF", i].Value.ToString() != "" &&

dGridPiezo["t\_cm", i].Value.ToString() != "" &&

dGridPiezo["d\_cm", i].Value.ToString() != "")

{

dGridPiezo["e\_re", i].Value = pm.e\_re(Convert.ToDouble(dGridPiezo["t\_cm", i].Value.ToString()),

Convert.ToDouble(dGridPiezo["d\_cm", i].Value.ToString().ToString()),

Convert.ToDouble(dGridPiezo["C\_pF", i].Value.ToString().ToString())).ToString();

}

….}

/// <summary>

/// Commons the import.

/// </summary>

/// <param name="OpenFileDNON">The open file dnon.</param>

public void Common\_Import(string OpenFileDNON)

{

int cst = 0;

DataTable dt = new DataTable();

toolStripStatusLabel2.Text = "Open file: " + OpenFileDNON;

string fileN1 = OpenFileDNON;

DataRow row;

FileJob FJ = new FileJob();

string[] s;

try

{

s = FJ.ReadF(OpenFileDNON);

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

return;

}

for (int i = 0; i < s.GetLength(0); i++)

{

ParseStringTab p = new ParseStringTab();

ParseStringTab p1 = new ParseStringTab();

p.AddString(s[i]);

//создание заголовков таблицы

if (i == 0)

{

p.AddString(s[i]);

IEnumerable<string> pdis = p.Distinct();

cst = pdis.Count();

foreach (string st in pdis)

{

p1.Add(st);

}

for (int j = 0; j < p1.Count; j++)

{

p1[j] = FJ.CorrectHead(p1[j]);

dt.Columns.Add(p1[j]);

}

}

else

{

//цикл для сортировки

int x = 0;

row = dt.NewRow();

for (int j = 0; j < p.Count(); j++)

{

try

{

row[x] = p[j];

++x;

if (x == cst)

{

dt.Rows.Add(row);

row = dt.NewRow();

x = 0;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

}

p.Clear();

}

dTreatmentIn.DataSource = dt;

}

….

public void MainMeasuringUnderBiasU()

{

//C(dU)\_man

//C(dU)\_auto

//C(dU\_df)

//C(U)\_relaxation

//C(dU\_df\_dT)

//C(dU\_dT)

switch (frmMOpt.cWorkMode.Text)

{

case "C(dU)\_man":

{

MeasTime.Start();

frmGPIB.WriteCommandeSync(PP.BiasAgilent4980 + PP.BiasUCurrent.ToString());

SendCommandMeasAtUbias();

MeasTemp(PP.MeasuringFrequency.ToString(), PP.CelSel);

++PP.CelSel;

this.Refresh();

break;

}

case "C(dU)\_auto":

{

for (int j = 0; j < frmMOpt.tVoltageList.Lines.Count() - 1; j++)

{

PP.BiasUCurrent = Convert.ToDouble(frmMOpt.tVoltageList.Lines[j]);

this.txtUbias.Text = PP.BiasUCurrent.ToString();

this.txtCurFreq.Text = PP.MeasuringFrequency.ToString();

SendCommandMeasAtUbias();

MeasTemp(PP.MeasuringFrequency.ToString(), PP.CelSel);

++PP.CelSel;

this.Refresh();

}

this.Refresh();

break;

}

….

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PiezoMath;

namespace PiezoMath.Test

{

[TestClass]

public class PiezoMathCalculationTests

{

[TestMethod]

public void Udelobemelsopr\_test()

{

//arrange

double d = 1.000;

double Rizol = 1e10;

double h = 0.1;

double expected = 78500000000;

//act

PiezoMathCalculation PM = new PiezoMathCalculation();

double actual = PM.Udelobemelsopr(Rizol, d, h);

//assert

Assert.AreEqual(expected, actual);

}

….

/// <summary>

/// Class list of string

/// </summary>

/// <seealso cref="System.Collections.Generic.List{String}" />

class ParseStringTab :List<string>

{

/// <summary>

/// Pars string by \t and add the string.

/// </summary>

/// <param name="s">The s.</param>

public void AddString(string s)

{

string sq="";

for (int i = 0; i < s.Length; i++)

{

if (s[i] != '\t')

{

sq = sq + s[i];

}

if (s[i] == '\t')

{

this.Add(sq);

sq = "";

}

}

this.Add(sq);

}

/// <summary>

/// Adds the meas string.

/// </summary>

/// <param name="s">The s.</param>

public void AddMeasString(string s)

{

string ss="";

for (int i = 0; i < s.Length; i++)

{

if (s[i] != 44)

{

ss += s[i];

}

if (s[i]==44)

{

this.Add(ss);

ss = "";

}

}

}