# Real Time Processing Monitoring

# Project Report

**ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY**

**Information Processing**

**Group 23**

Nguyễn Thiện Tuấn

Phạm Nhật Nam

Supervisor

**Prof. Pech**

Second Supervisor

**Mr. Ha Ngoc An**

# Table of Contents

# Introduction

## Background

Information Processing is a standard module included in the bachelor program of Electrical Engineering and Information Technology at the Vietnamese-German University, Ho Chi Minh city, Vietnam in association with the Frankfurt University of Applied Sciences, Frankfurt, Germany. This report is the technical report of the theme project provided by the professor in charge and acts as a part of quality assessment of the course. The score of the report will indicate the completeness of the module of the students whose names are written at the front page. The project is due on May 3rd, 2017 accomplished by this written document submission.

## Project description

The project was conducted in a group consisted of two students. Students are asked to develop an application run on personal computer running Windows 8 and above with the following criteria:

- The application must be developed using Visual C# as the base language, target at personal computer running x86/x64 versions of Windows 8 and above.
- The application must have a graphical user interface (GUI) designed using Windows Form, included as a part of the Microsoft .NET Framework.
- The data are fetched from a web server.

# The application

## The User Interface

The application consisted of a single window with two tabs. By default, the User Interface started on the first tab.

The first tab allowed the user to fetch a number of parameters, which are industrial processes being surveyed and streamed on a web server, then display them in a graphical chart with selectable duration.

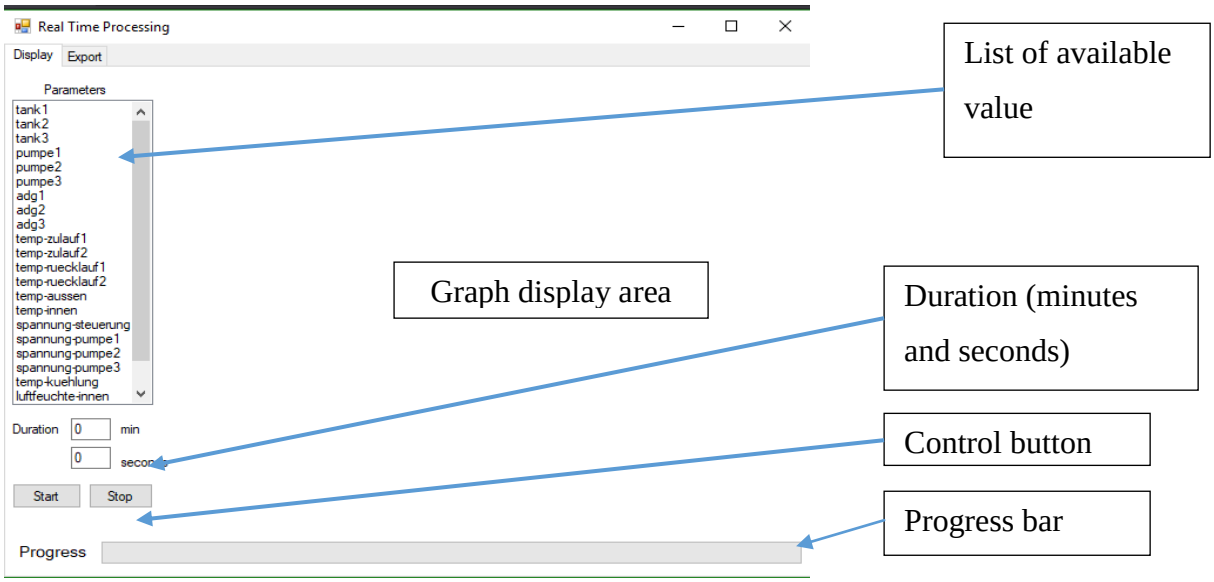



Figure 1. The “Display” tab

The second tabs allowed the user to export the surveyed data of a process into a Microsoft Excel file, with selectable time range and time interval. The data exported is printed line by line in four columns, which from leftmost to rightmost, came in the order of: maximum value, minimum value of the respected variable, the actual data recorded, and a timestamp to signify when was the data being recorded.

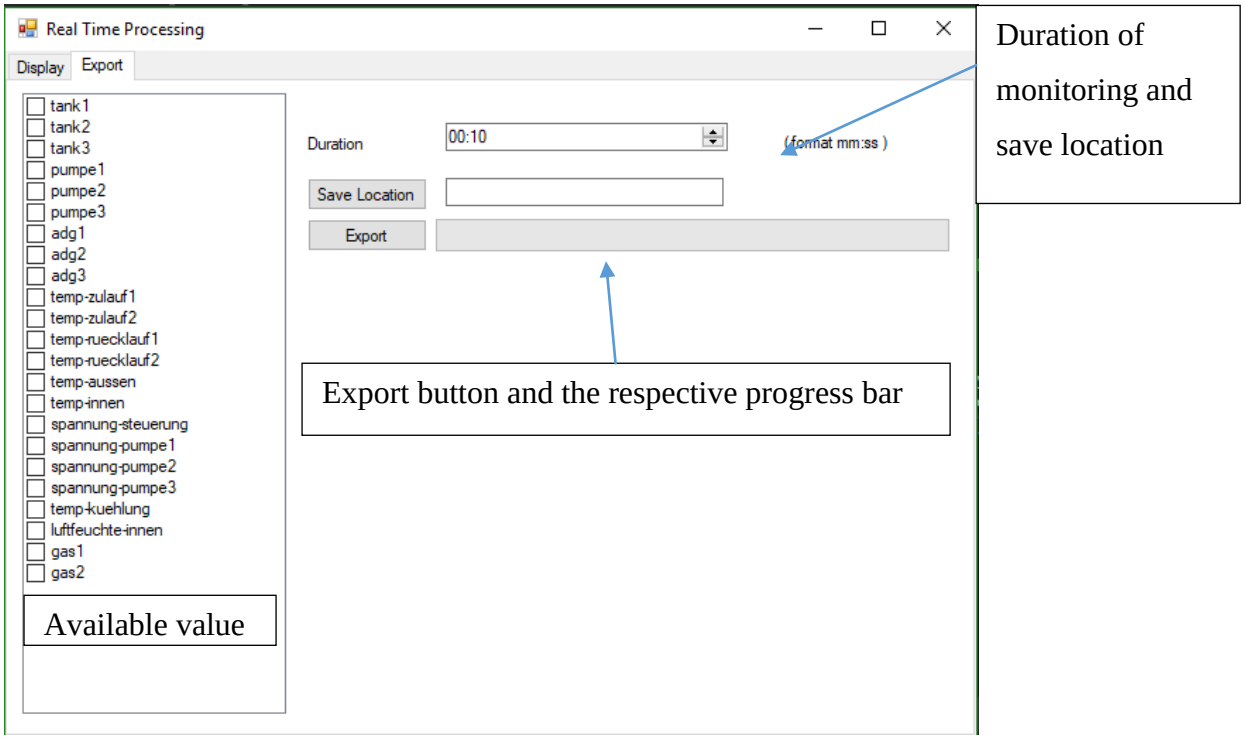



Figure 2. The "Export" tab

The application was developed using Visual Studio 2015 Community Edition in C# language and .NET Framework 4.5.2. The result was captured by compiling and running the source code (named "Solution" by Visual Studio standard) using Visual Studio 2015. For the "Export" function, a copy of Microsoft Excel installed on the machine is required.

## How to run the program

As soon as the "Run" button is clicked, the source code is compiled and run. Then, the user interface pop up and wait for the instruction of the user.

The first action, which is done automatically by design, is to check if connections to the aforementioned websites could be established. If a connection is possible, the application will connect to the websites and fetch the list of available process in which data could be mapped by the application standard. This process requires an Internet connection to the server which hosted the website.

### Plot the parameters

The parameters available would be displayed in a drop list to the left of the window, which the user can access by clicking the arrow button to view the list. The user must click the desirable parameters, enters the values for durations, and then click "Start".

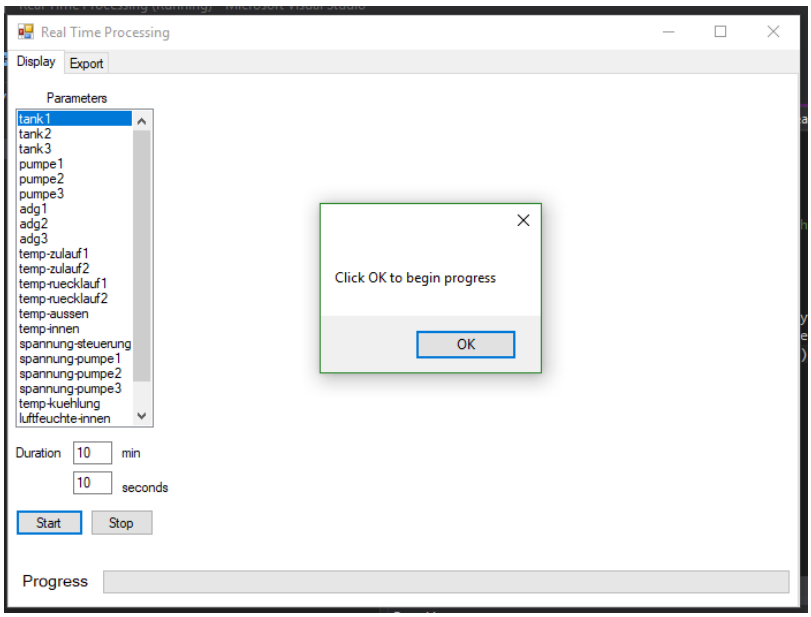



Figure 3. A plotting operation of parameter ***tank 1*** with duration of ***10 minutes 10 seconds***.

A dialogue box will appear which prompt the user to click “OK” to begin the process.

If the user want to stop the process, simply click on the “Stop” button and click on the “Stop” button on the pop up dialogue box. The part of the chart had been plotted would not disappeared.

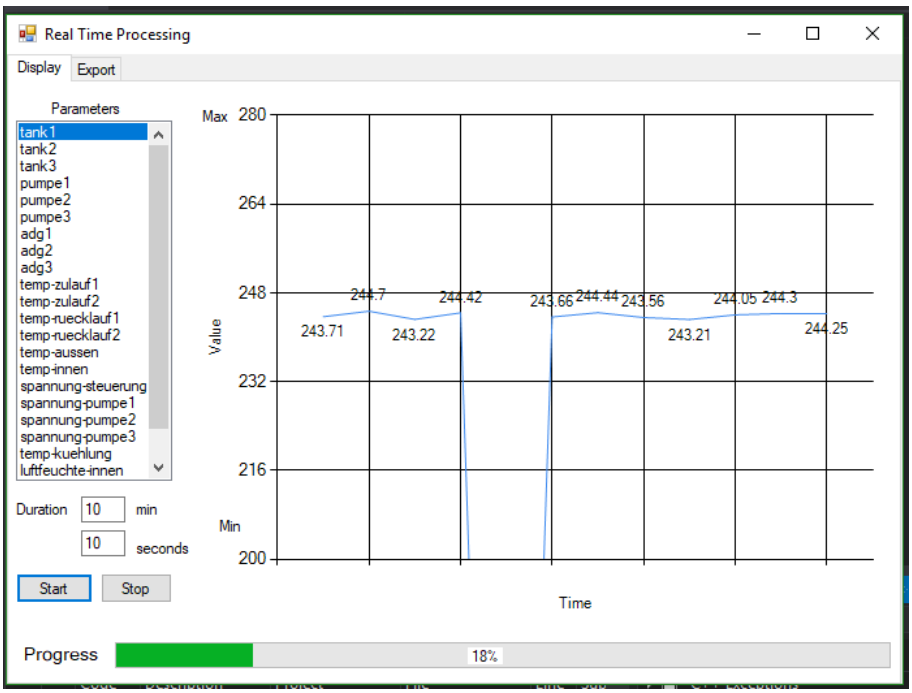



Figure 4. The chart for value “tank 1”, with a duration of 10 minutes 10 seconds. Here the process is 18% completed.

After the process completed, a dialogue box would appeared, notify the user that the process of graphing had finished and the program is at rest, waiting for the next request from the user. The chart would remain until the next graphing.

### Export the variable into an Excel file

Starting from the idle state, the user must chose the second tab titled “Export”. A new interface appeared. In this page the left is a list of values could be export into the Excel file. The user would choose one of the values, then choose the duration of the process, while the interval is 10 seconds.

The location for the target output file could be input in two ways: either the user input it directly into the “Save Location” box, or the user could click on the “Save Location” button to open the Explorer window, and choose the location of the file as well as name it in the “File name” field. Then click the button “Start” to begin.

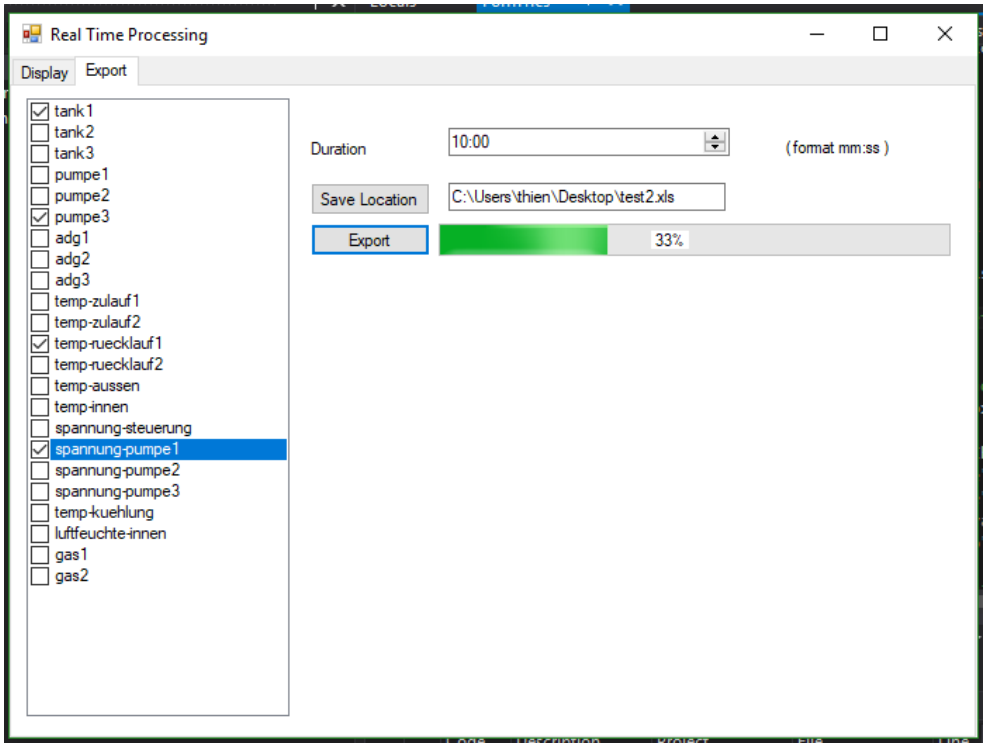
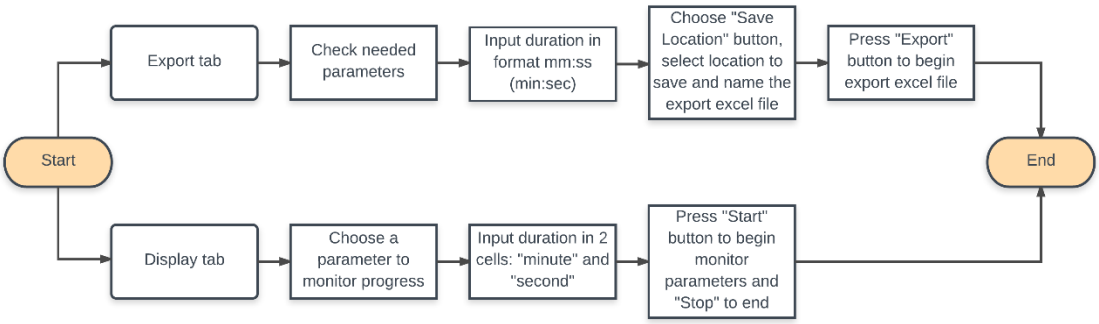



Figure 5. An export request in progress. In this example, four values are chosen to be exported: “tank1”, “pumpe3”, “temp-ruecklauf1” and “spanning-pumpe1”. The duration is set to be 10 minutes, export into file “test2.xls” locates in C:\Users\thien\Desktop\test2.xls. The process was 33% completed.

After finished exporting, a dialogue box would appear to notify the user, click “OK” to close it.

# Code description

## The general idea




The program has two different function with a similar input.

The input is data fetched from the web server by a web client, which are industrial processes monitored in real time. The data is serialized in JSON format for the ease of transmitting, so before we can use it, we must de-serialized it to usable data formats.

The first type of output is a chart, which is drawn by taking the input and plots them on a scale chosen by the user. The user must enter the duration desired for the chart.

The second type of output takes the input and writes them into an Excel workbook in a location chosen by the user.

## Setting up the program

```
1   using System;
2   using System.IO;
3   using System.Net;
4   using System.Text;
5   using System.Windows.Forms;
6   using Newtonsoft.Json;
7   using Excel = Microsoft.Office.Interop.Excel;
8   using System.Text.RegularExpressions;
9
10  namespace Real_TIme_Processing
11  {
12      public partial class Form1 : Form
13      {
14          WebRequest request = WebRequest.Create("http://ivlab.azurewebsites.net/getData"
            );//Create request to get info from web
15
16
17          public Form1()
18          {
37
38          ////////////////////////////////////////////////////////////
39          WebClient wc = new WebClient(); //Create Web Client variable to request info
40          int durationMinute, durationSecond, count;
41          string selected;
```

Figure 7. Setting up the program: including libraries, initialize the Form1 object and contact the server for the list of available values.

Notable among the included libraries are

- The **Newtonsoft.Json** library provides us the ability to convert the JSON objects sent from the webserver into useable data formats.
- The **Microsft.Office.Interop.Excel** allows us to export data into Excel files. For the sake of simplicity, we named it as the class **Excel** in this program.

On line 12, the Form1 inherited the class Form, a standard Windows Form class library. A web request go to the website http://ivlab.azurewebsites.net/getData is called on line 14, which is where the data are draws from.

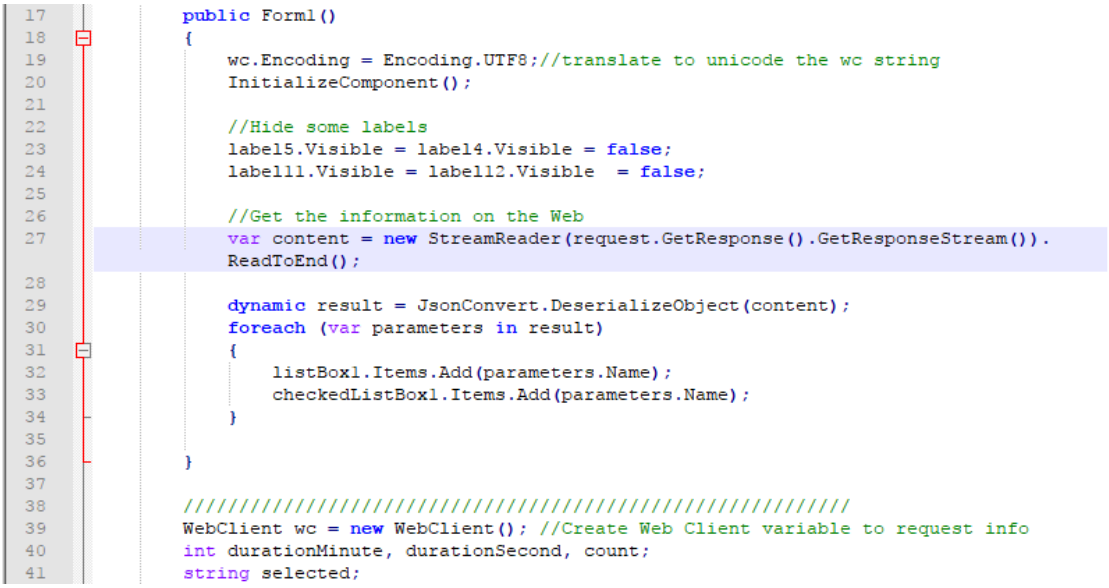



Figure 8. Connect with the web server, get the JSON data stream and create the list of variables available to be process.

Line 19 was to ensure the data read from the web server would be read in UTF-8, because the stream was not in Unicode and we use method *DownloadString* instead of a *SreamReader*.

On line 27, the object **content** was waiting to receive the data stream from the web server. The variable **result** on line 29 was defined as the storage for the converted JSON data read from **content**. As we did not know exactly what would be the data, we gave it the type *dynamic*[1].

The loop ***foreach***[2] from line 30 to line 34 put any **parameters** values we stored in **result** would be put into the **listBox1** and **checkedListBox1**, which denote the list of variables available for charting (**listBox1** in tab Display) and writing into Excel file (**checkedListBox1** in tab Export**)**. On line 39-41, we finish the setup with the creation of the object **wc** as a web client to get the data from the web server, and define four variables to be used later in the whole programs.

## Plotting with tab Display

```
42          //Start Button
43          private void Start_Click(object sender, EventArgs e)
44          {
45              //Clear the old chart
46              chart1.Series[0].Points.Clear();
47              //Check if any item in listBox1 is selected
48              if (listBox1.SelectedItems.Count!= 0)
49              {
71              else
72              {
73                  MessageBox.Show("Choose a parameter");
74              }
75          }
```

Figure 9. The outline of the chart is defined, with the duration entered.

On line 46, **chart1** performed a clear procedure to prepare for the plotting operation. The code from line 47 to line 74 checked if an item is chosen in the list of variables. If it does, the plotting operation constructs the outline of the chart with pre-entered parameters. If not, a prompt appeared, remind the user to choose one item to plot [3].

```
87          //Timer for Display tab (tab 1)
88          private void timer1_Tick(object sender, EventArgs e)
89          {
90
91              progressBar2.Value = count;
92              label11.Text = ((count * 100) / (durationMinute * 6 + durationSecond / 10)).
                ToString()+'%';
93              label11.Visible = true;
94              //Each 10s (set interval), count will +1, durationMinute,durationSecond=10s
                (same as set interval)
95              if (count++ == (durationMinute*6 +durationSecond/10))
96              {
97                  //Add stop time to the chart
98                  chart1.Series[0].Points.AddXY(DateTime.UtcNow.ToString("hh:mm:ss"), (
                    double)JsonConvert.DeserializeObject<dynamic>(wc.DownloadString("
                    http://ivlab.azurewebsites.net/getData"))[selected][5]["actual"]);
99                  timer1.Stop();
100                 MessageBox.Show("Progress finished");
101             }
102             else
103             {
104                 chart1.Series[0].Points.AddXY(" ", (double)JsonConvert.DeserializeObject<
                    dynamic>(wc.DownloadString("http://ivlab.azurewebsites.net/getData"))[
                    selected][5]["actual"]);
105             }
106         }
107
```

Figure 10. The data is read from the de-serialized JSON stream into the chart.

Despite being called **timer1_Tick** object, it included the actual plotting operation. **Timer1** calculate whether the duration is on time or not. If the duration was not ended yet, **wc** takes values from the web server as a stream of string, convert it from JSON into *dynamics* data, and then passes it to **chart1** to plot it on the chart. If the duration ended, **timer1** told **chart1** to stop plotting and display a dialogue box “Progress finished” [3].

A notable feature is the use of methods **DownloadString** instead of **StreamReader**. While the latter was more universal and was offered as an example by the professor, we found that the latter is shorter, had to deal with less exceptions and was overall more reliable [4].

## Exporting to Excel file

```
107
108         //Create variable used in tab Export
109         int count2;
110         int time;
111         Excel.Application exapp; //variable to run Excel Application
112         Excel.Workbook wb; //variable to create new Excel workbook
113         Excel.Sheets ws; //Variable to create new Excel Worksheet
114         //Export button
115         private void ExportButton_Click(object sender, EventArgs e)
116         {
117             if (textBox2.Text!="")
118             {
155             else { MessageBox.Show("Choose a file location"); }
156         }
157
158         //Export tab duration choosing
159         int dt;
160         //Set that each time press arrow to change value in dateTimePicker1, value
            increase 10 units
161         private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
162         {
174
175         //Save Location Button
176         private void SaveLocation_Click(object sender, EventArgs e)
177         {
184
185         //Timer for Export tab
186         private void timer2_Tick(object sender, EventArgs e)
187         {
214
215
216     }
217 }
```

Figure 11. The entire code for the Excel exporting feature, folded for ease of visibility.

The ability to export the entire duration of a parameters into an Excel file was done through external wrappers and methods which are not explicitly showed. However ones did not need to understand the mechanics, as the methods are rather straight forward. Methods **ExportButton_click** worked similar to method **timer1_Tick**: takes values of parameters from de-serialized JSON data fetched from the web server by the web client **wc**. Instead of plot them on a chart, the method write them in an Excel file **ws** located in a user-defined location [3].

Despite using a reliable method, it is not error-free. Due to different point in new workbooks of different Excel edition, error may arise.

```
115         private void ExportButton_Click(object sender, EventArgs e)
116         {
117             if (textBox2.Text!="")
118             {
119                 exapp = new Excel.Application();//Run Excel Application
120                 wb = exapp.Workbooks.Add();//Create new workbook
121                 ws = wb.Worksheets;//Create new worksheet in workbook
122
123                 var x = JsonConvert.DeserializeObject<dynamic>(wc.DownloadString("
                    http://ivlab.azurewebsites.net/getData"));
124                 // dateTimePicker set time interval of 10 seconds, so the minute must
                    multiply by 6 (not 60) to transfer to real equivalent second and the
                    second must be divided by 10 to transfer to real equivalent second.
125                 time = dateTimePicker1.Value.Minute * 6 + dateTimePicker1.Value.Second /
                    10;
126                 count2 = 0;
127                 //Set Maximum of progress bar
128                 progressBar1.Maximum = time;
129                 //Run timer 2
130                 timer2.Start();
131                 MessageBox.Show("Click OK to begin progress");
132                 //Old Microsoft Excel has 3 sheets at beginning, so if worksheets less
                    than 3, we dont need new sheets
133                 if (checkedListBox1.CheckedIndices.Count > 1) //Check if checked items
                    in checkedlistBox is more than 3
134                 {
135                     for (int i = 1; i <= checkedListBox1.CheckedIndices.Count -1; i++)
136                     {
137                         ws.Add();//Add new worksheet
138                     }
139                 }
```

Figure 12. The line of code from 134 to 138 is fixed for Excel 2016. For earlier editions of Excel, *checkedListBox1.CheckedIndices.Count* must be check with 3 instead of 1.

The highlighted code (line 132-138) checks if the file is a newly created one by checking the number of worksheets it currently holds. If such file is existing, the program simply add a new worksheet to it and proceed on the new worksheet [5][6].

In Excel 2010 and previous editions, the standard number of worksheets created in a new workbook was always three. However, Excel 2013 and later editions only have one worksheets in a newly created workbook. If the number is not fixed, exceptions are not properly taken care of and may crash the program, as shown below.

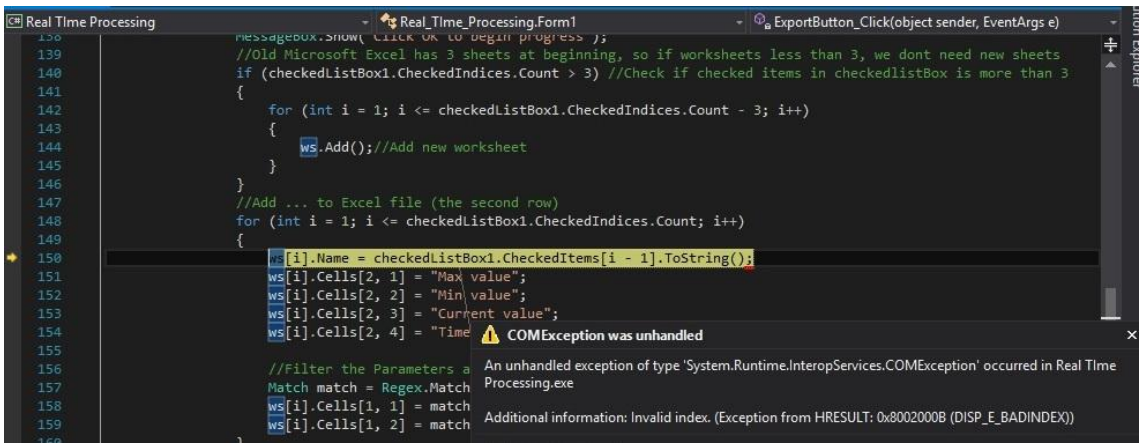



Figure 13. Unhandled exception if the variables are tested with 3 sheets as standard, on a system with Excel 2016 installed.

# Conclusion

This paper explained the idea behind a possible solution to the project “Real time Processing Monitoring” using C# and Windows Form. Our solution worked for two required output, one required us taking the data from a predefined web server and plot it on a chart, and the other required us taking the data from the same predefined web server and export it into an Excel workbook.

The primary objective was to gain a further understand in project planning and executing. By doing this project, not only we gain experience in working with C# and graphical user interface using Windows Form, but we also gained experience from dealing with possible events that are often encountered during a software development project.

# Reference

[1] JSON de-serializing guide

http://www.newtonsoft.com/json/help/html/SerializingJSON.htm

[2] foreach reference

https://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx

[3] Chart Class reference

https://msdn.microsoft.com/en-us/library/system.windows.forms.datavisualization.charting.chart(v=vs.110).aspx

[4] Using *DownloadString* in conjunction with WebClient class

https://stackoverflow.com/questions/6656451/c-sharp-read-webpage-content-streamreader

[5] Microsoft.Office.Interop.Excel Reference

https://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel.aspx

[6] How to: Use COM Interop to Create an Excel Spreadsheet

https://msdn.microsoft.com/en-us/library/ms173186(v=vs.80).aspx