OPC Lab

Isak Skeie, 245362

Table of Contents

1 Introduction

The Lab work focuses on OPC, this includes OPC DA and OPC UA. The assignment walks through how to build an OPC server in LabView, while using real world sensor readings as the basis of data transfer.

1.1 Background

OPC is a communication standard, much used in the industry of automation. This makes them important to learn as an automation engineer. OPC includes among other things OPC-DA and OPC-UA. It could be stated that DA is outdated and not that relevant anymore. But for educational purposes its useful to start with this version, and then expand to OPC-UA, which includes a suite of functionality.

1.2 Temperature Sensor

The assignment specifies a temperature sensor to be used a the basis of data being transferred by the OPC servers. This is something that needs to be performed at the School. Because this lab is done at home and at work, an alternative to the temperature sensor is needed. The readings are therefore simulated with a random number generator in LabView's. Which is then scaled to realistically represent temperature reading (disregarding the deviations!).

2 OPC DA

2.1 OPC-DA Server

The OPC DA server used for this project is the one that came with LabView. NI OPC SERVERS. It comes with premade channels that emulate industrial hardware. Channel 1, Device 1 is used, as it hosts two tags made for testing purposes. Tag1 and Tag 2 are configured as doubles, as the simulated temperature readings are decimal values.

2.2 OPC DA Read/Write

LabView are used to create a client for the OPC DA server. This is done by using the Datasocket library that's included in the LabView installation. The block diagram starts with a module for finding the right server and node. The next module opens up a connection with the server in Write mode. Within the While-loop, a random number between 20 and 30 is created, shown in a GUI in LabView, and written to the server. Figure 2.1 shows the block diagram.

For reading from the OPC server, another LabView program is used, as seen in figure 2.1. This works in the same way as the Writing client, with the exception of the Temperature simulator and mode of connection. Figure 2.2 also shows another OPC DA client, provided from NI. This is used to verify that the tag in the OPC server is being written to.

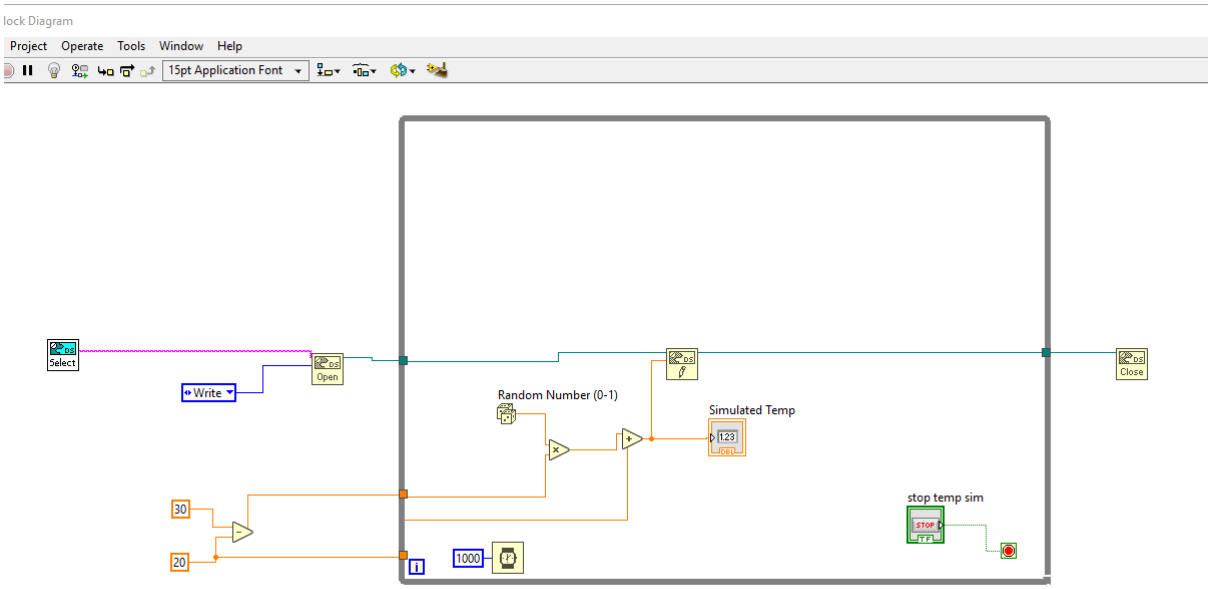

Figure 2-1: OPC client writing simulated temperature values to server

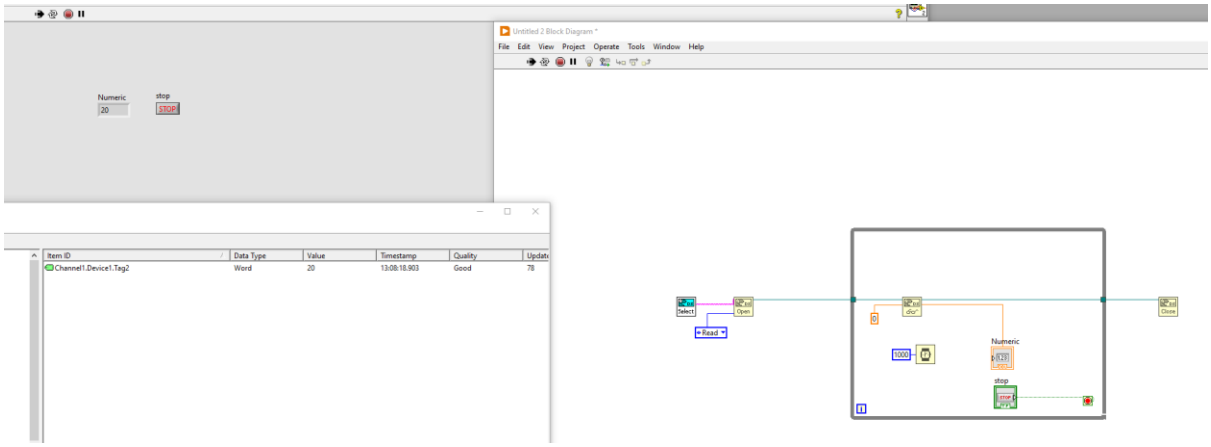

Figure 2-2: OPC client Reading values from tag in DA server

2.3 Datalogging

To visualize the generated temperature data, it is being logged to a text file through a OPC client in LabView as seen in figure 2.3. The text file is the opened in Excel and graphed to display the readings in a clearly manner. A excerpt from the excel file is shown in figure 2.3

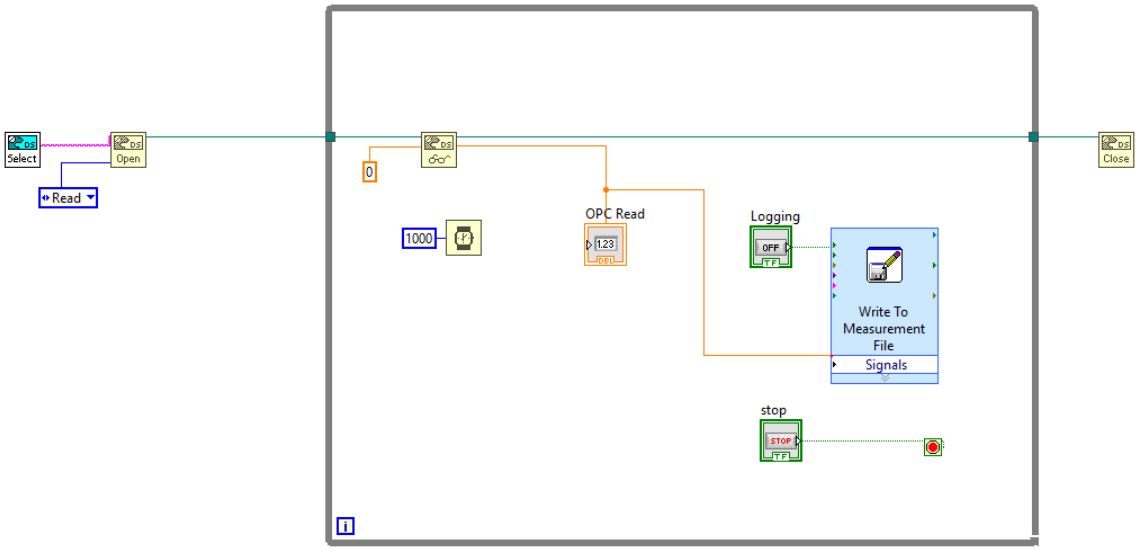

Figure 2-3: OPC client logging values every second

0	0
0	26
2	29
3	21
4	28
5	27
6	23
8	22
9	23
10	25
11	23
12	29
13	25
14	27
16	20
17	22
18	28
19	25
21	30
22	22
23	24
24	25
25	27
26	30

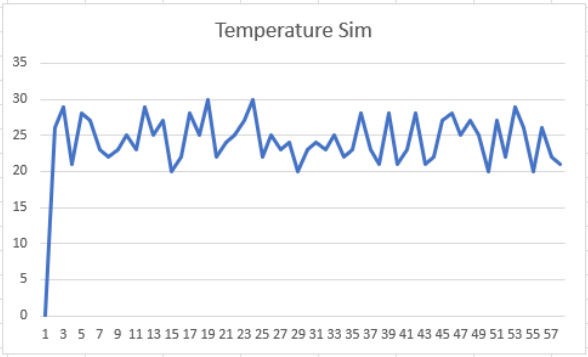

Figure 2-4: A cutout from the log-file

3 OPC UA

Compared to OPC DA, comes OPC UA with more functionality and a more modern way of communicating [1]. This makes the setup more advanced, and more considerations needed to be made.

3.1 OPC-UA Server

LabView is used as a OPC server. This is done by downloading external packages that gives LabView the functionality it needs. The block diagram for the server is shown in figure 3.1

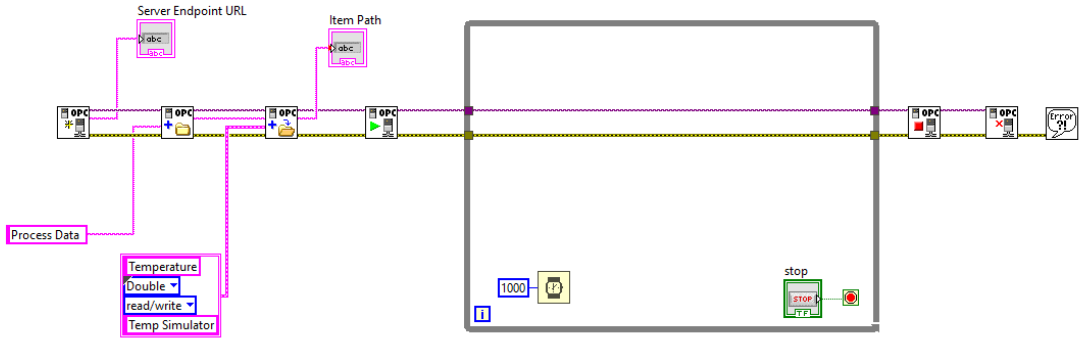

Figure 3-1: LabView Block Diagram for the OPC UA server

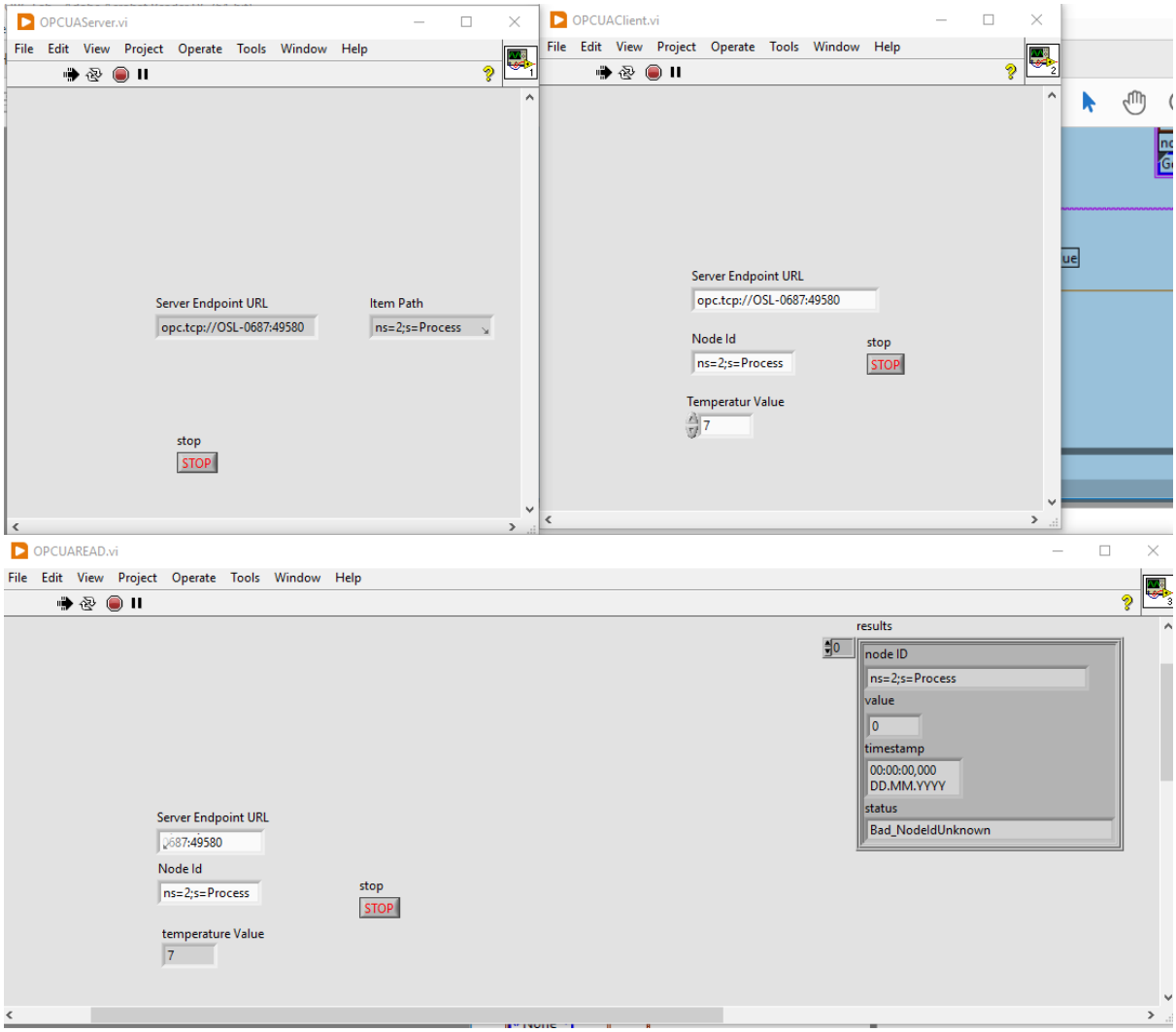

Figure 3-2: UI for OPC-UA server and clients

3.2 OPC-UA Clients

Two clients are created for the OPC UA server, one for reading data, and one for writing data.

The LabView client program that writes data to the server has the GUI shown to the top right in figure 3.2. The user are able to change the number that gets written to the server. The other client program, reads data from the server and displays it in the GUI for the user. From the Figure below its possible to verify that the Clients/servers are working as they are supposed to. In the figure, the top left window displays the GUI for the server, the top right writing client, and the bottom one the reading client.

3.3 C# Client

From the assignment, suggestions for alternative OPC DA clients was provided. These client alternatives came with their own challenges, as both OPC DA, and the client alternatives is outdated (For example the python library, which supports an older and outdated python version). The solution is to create an alternative OPC UA client with C#. The OPC Foundation have provided an example of

a C# client on GitHub. The repo is cloned and executed in Visual Studio 2019. The program provides a GUI a seen in figure 3.3. By using the client to connect to the LabView OPC UA client, we can verify that both of them work. As seen in the figure.

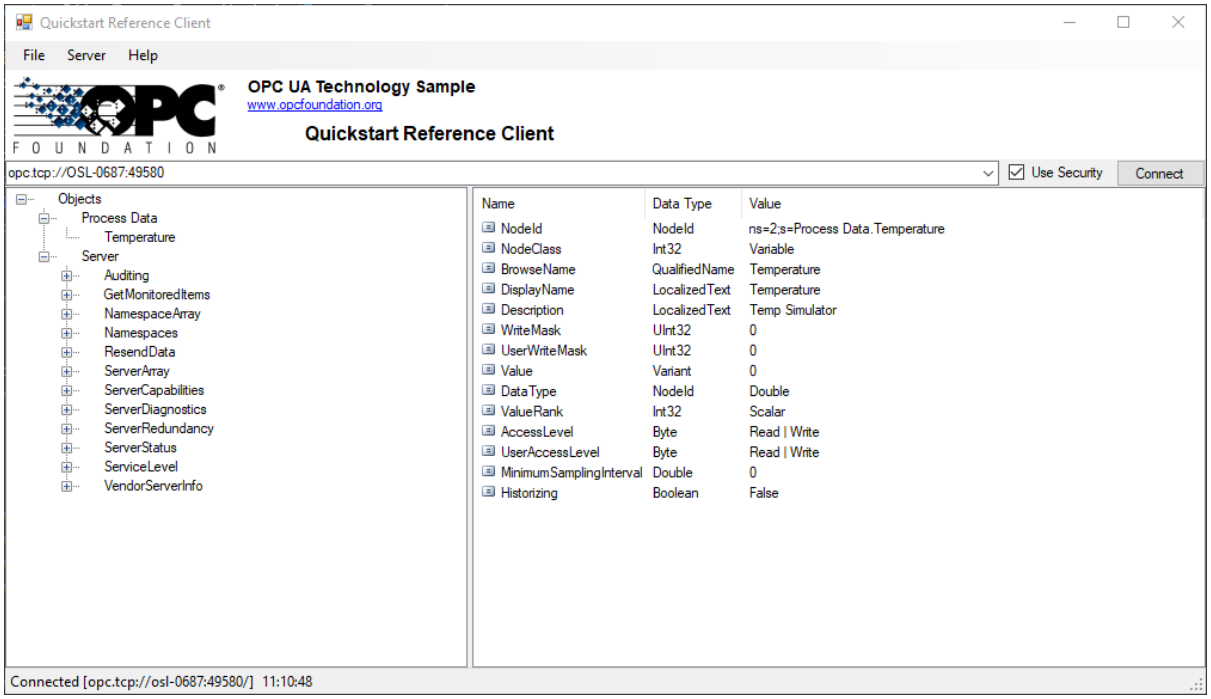

Figure 3-3: OPC UA client created from C# script

4 Conclusion

OPC is a widely used communication protocol. Utilized in industrial environments for automation and plant control. By doing this lab, you get an overview of what this protocols capable of, as well as the many differences and between OPC-DA and OPC-UA. This includes highlighting the capabilities of OPC-UA and the functionality contained within this standard.

By performing tasks with LabView, you get a visual representation of the core functionality of communication with OPC, this includes both OPC-DA and OPC-UA. With the addition of having to deploy an alternative client, useful skills for working in the automation and IT industry is obtained.

5 Bibliography

[1] TheAutomization, "opc-ua-vs-opc-da," 23 02 2022. [Online]. Available: https://theautomization.com/opc-ua-vs-da/.

6 References

- OPCFoundation/UA-.NETStandard: OPC Unified Architecture .NET Standard (github.com)
-

```
/*
============================================================================
* Copyright (c) 2005-2020 The OPC Foundation, Inc. All rights reserved.
*
* OPC Foundation MIT License 1.00
*
* Permission is hereby granted, free of charge, to any person
* obtaining a copy of this software and associated documentation
* files (the "Software"), to deal in the Software without
* restriction, including without limitation the rights to use,
* copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the
* Software is furnished to do so, subject to the following
* conditions:
*
* The above copyright notice and this permission notice shall be
* included in all copies or substantial portions of the Software.
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
* NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
* FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
* OTHER DEALINGS IN THE SOFTWARE.
```

```
 *
 * The complete license agreement can be found here:
 * http://opcfoundation.org/License/MIT/1.00/
 *
==========================================================================*/

using System;
using System.Windows.Forms;
using Opc.Ua;
using Opc.Ua.Client.Controls;
using Opc.Ua.Configuration;

namespace Quickstarts.ReferenceClient
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // Initialize the user interface.
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            ApplicationInstance.MessageDlg = new ApplicationMessageDlg();
            ApplicationInstance application = new ApplicationInstance();
            application.ApplicationName = "UA Reference Client";
            application.ApplicationType   = ApplicationType.Client;
            application.ConfigSectionName = "Quickstarts.ReferenceClient";

            try
            {

                // load the application configuration.

                application.LoadApplicationConfiguration(false).Wait();

                // check the application certificate.
                var certOK = application.CheckApplicationInstanceCertificate(false, 0).Result;
                if (!certOK)
                {
                    throw new Exception("Application instance certificate invalid!");
                }

                // run the application interactively.
                Application.Run(new MainForm(application.ApplicationConfiguration));
            }
            catch (Exception e)
            {
                ExceptionDlg.Show(application.ApplicationName, e);
```

```
            }
        }
    }

    /// <summary>
    /// The <b>ReferenceClient</b> namespace contains classes which implement a Quickstart Client.
    /// </summary>
    /// <exclude/>
    [System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    public class NamespaceDoc
    {
    }
}

```