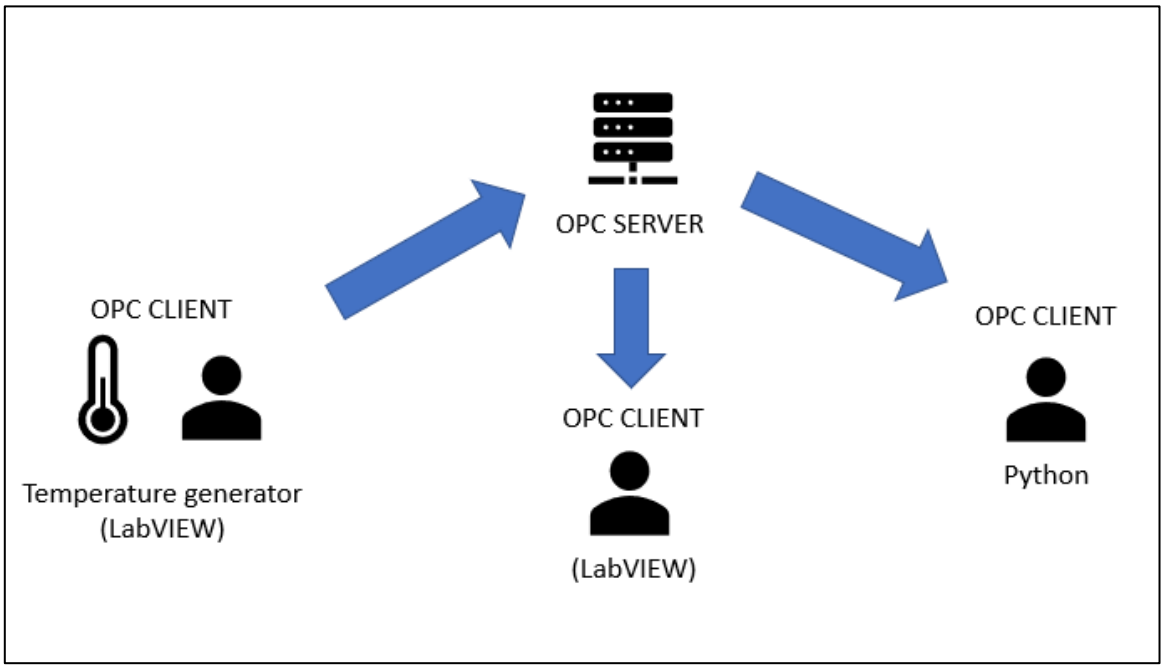
OPC Lab

Lars Rikard Rådstoga | 223786

2022-02-17



1 Introduction

The OPC technology is a very popular toolset used to handle data in process and manufacturing industry. There are many ways to implement this technology through a multitude of different technology stacks. The following report will investigate technologies that can be used with OPC: Matrikon OPC server, LabVIEW and Python. Matrikon will be used as is, LabVIEW will be used alongside the OPC UA Toolkit, and Python will be used with the asyncua library.

There are two generations of OPC: the classic DA/Data access and the new UA/Unified Architecture. It is good to know about both, because a lot of legacy systems can still exist using DA. First OPC DA will be investigated using Matrikon for the server and LabVIEW for the clients. Then LabVIEW will be used to create an OPC UA server with both LabView and Python clients.

2 Results

The following subchapters will contain screenshots of the lab activities and commentary.

2.1 Read temperature data into LabVIEW

As an industry master student, I will come into the lab later to complete this step.

2.2 OPC DA: Send temperature data using LabVIEW to OPC server

An OPC server was quickly initialized using MatrikonOPC. Using the explorer, see Figure 2-1, it was easily possible to connect to the server and add a tag to it.

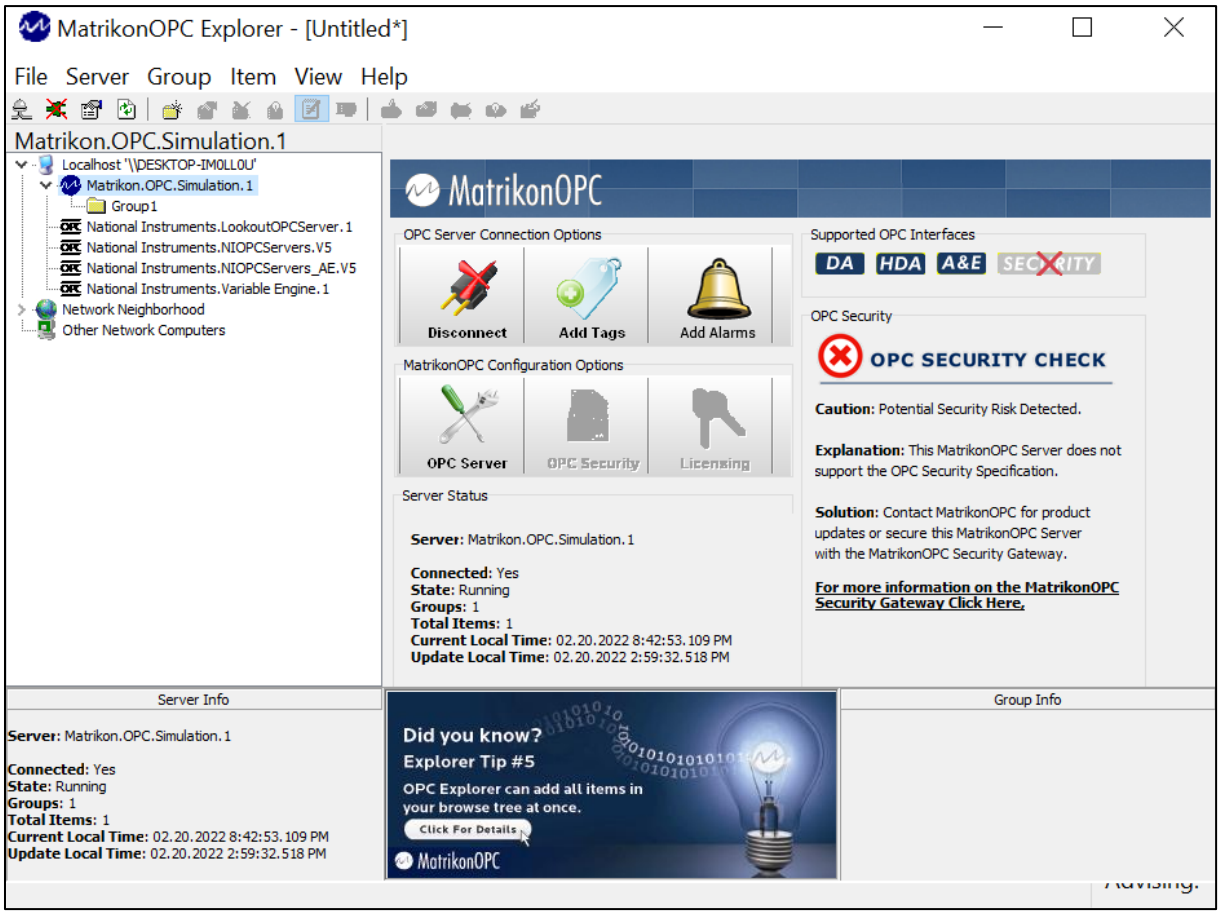


Figure 2-1 MatrikonOPC Explorer.

To generate some data to send to the OPC server, a random generator was used as suggested, see Figure 2-2.

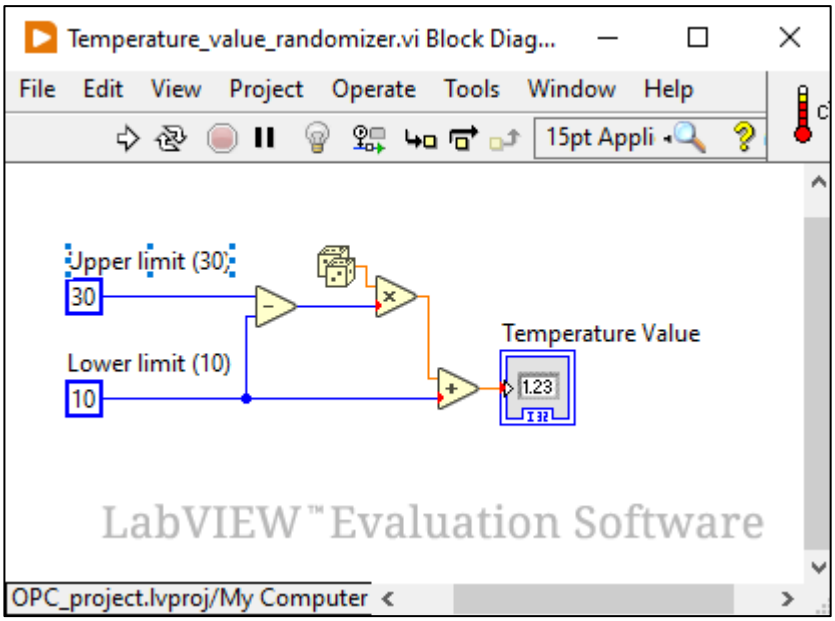


Figure 2-2 Random temperature generator.

An OPC client was then implemented incorporating the random generator. The block diagram seen in Figure 2-3 includes an endpoint address on the left, connected to a data socker opener. A loop is then responsible for writing data to the tag using the random generator and a data socker writer. Finally a data socker closer is executed when the loop is stopped.

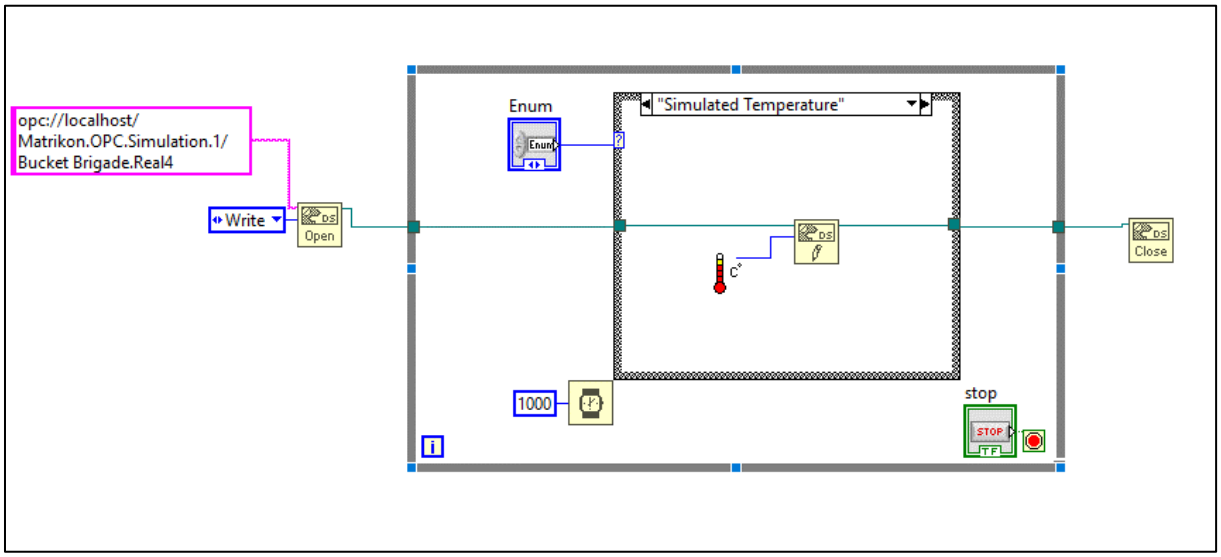


Figure 2-3 block diagram of OPC client used to write data to a tag.

The program successfully updates the tag, which is observed in the MatrikonOPC explorer, as seen in Figure 2-4.

Item ID	Value	Quality	Timestamp	Status
Bucket Brigade.Real4	23	Good, non...	02.20.2022 1:51:57.666 PM	Active

Figure 2-4 The tag value is updating according to the MatrikonOPC explorer.

2.3 OPC DA: Read temperature data from OPC Server using LabVIEW

Another OPC client was created in LabVIEW, see Figure 2-5 for the block diagram. Similarly, a data socker opener is added first on the left with a tag endpoint address. A loop is also used here checking the tag's value each second and writing it to an indicator.

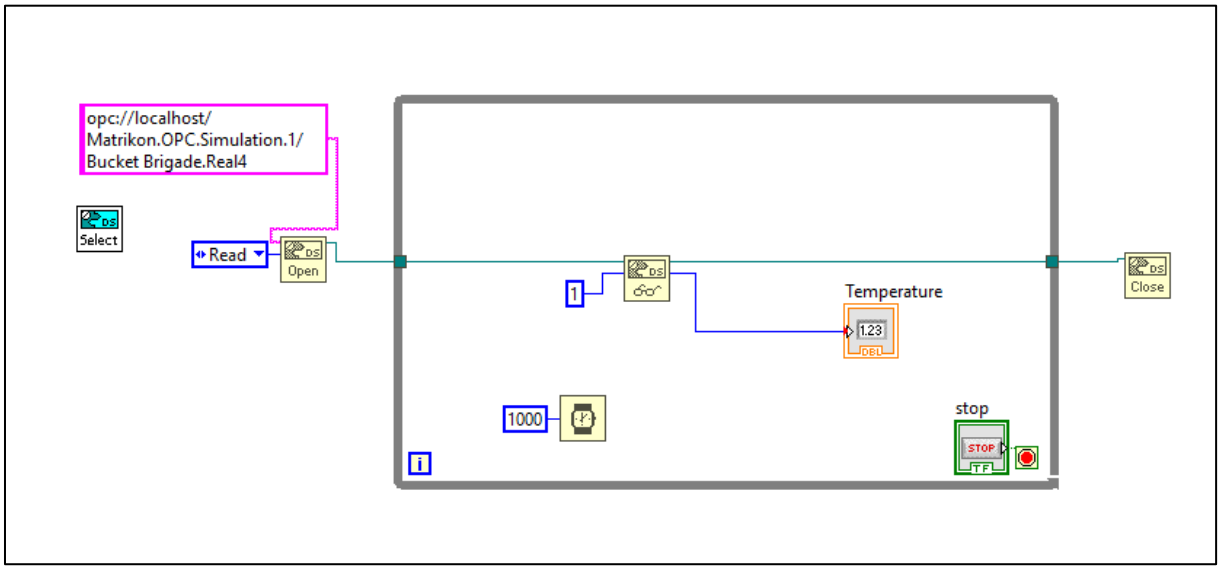


Figure 2-5 OPC client for reading.

Figure 2-6 Shows the front panel of the program, which also contains a button to toggle data logging.

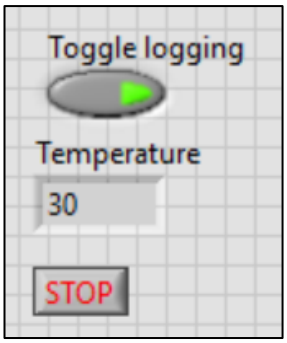


Figure 2-6 OPC client front panel.

The data logging functionality was added to the block diagram as seen in Figure 2-7. Data read from the OPC can now be saved to a file. Alternatively, it could have been just as easily saved to a database.

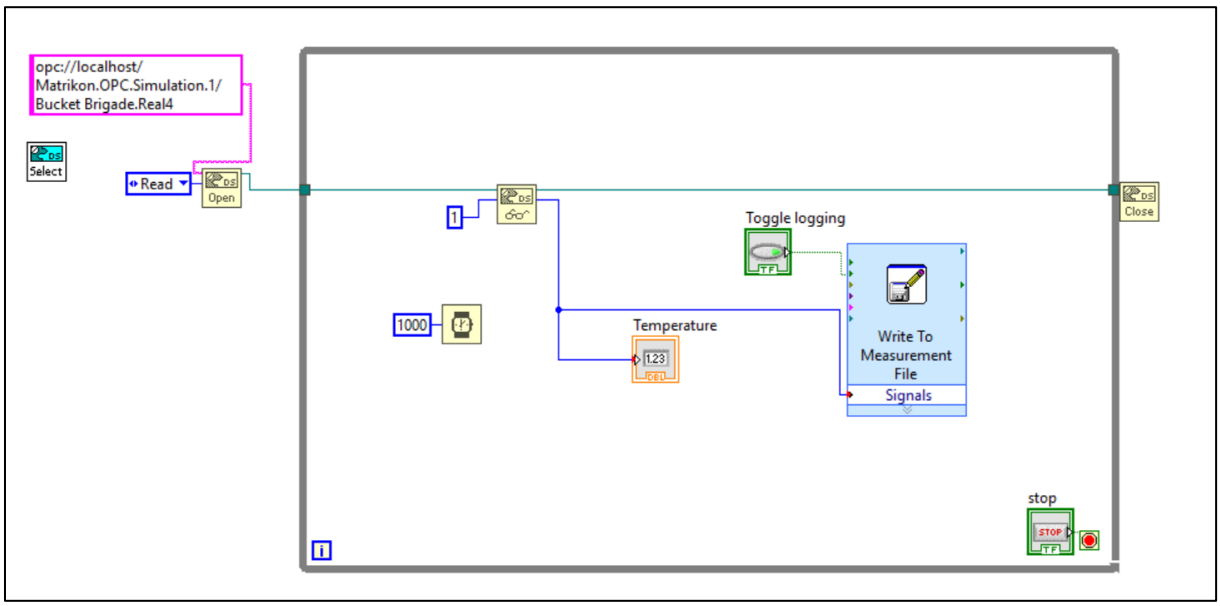


Figure 2-7 Updated OPC client with logging.

Figure 2-8 shows a section of the saved data, and Figure 2-8Figure 2-9 shows the data plotted in Excel.

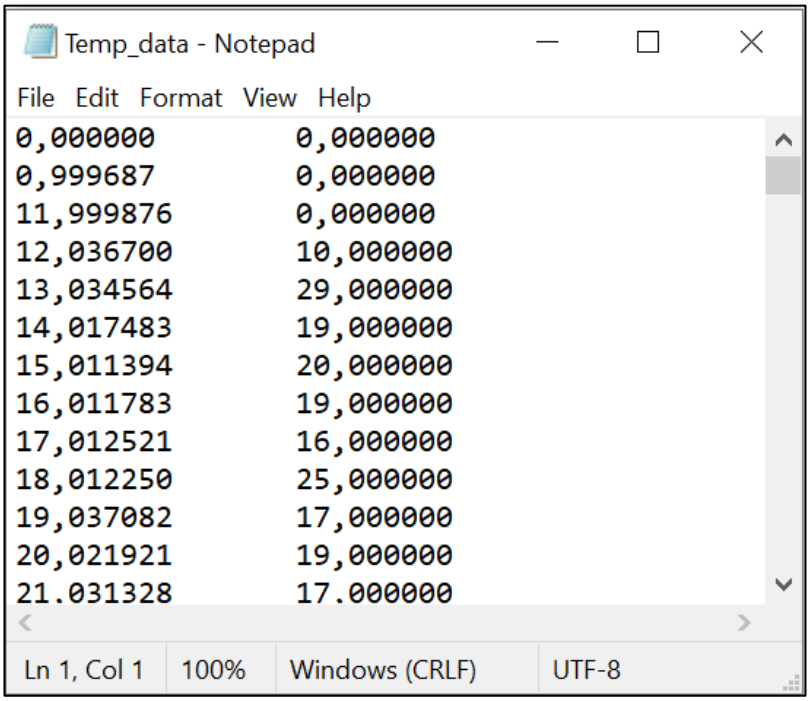


Figure 2-8 Data logged with data from the OPC server.

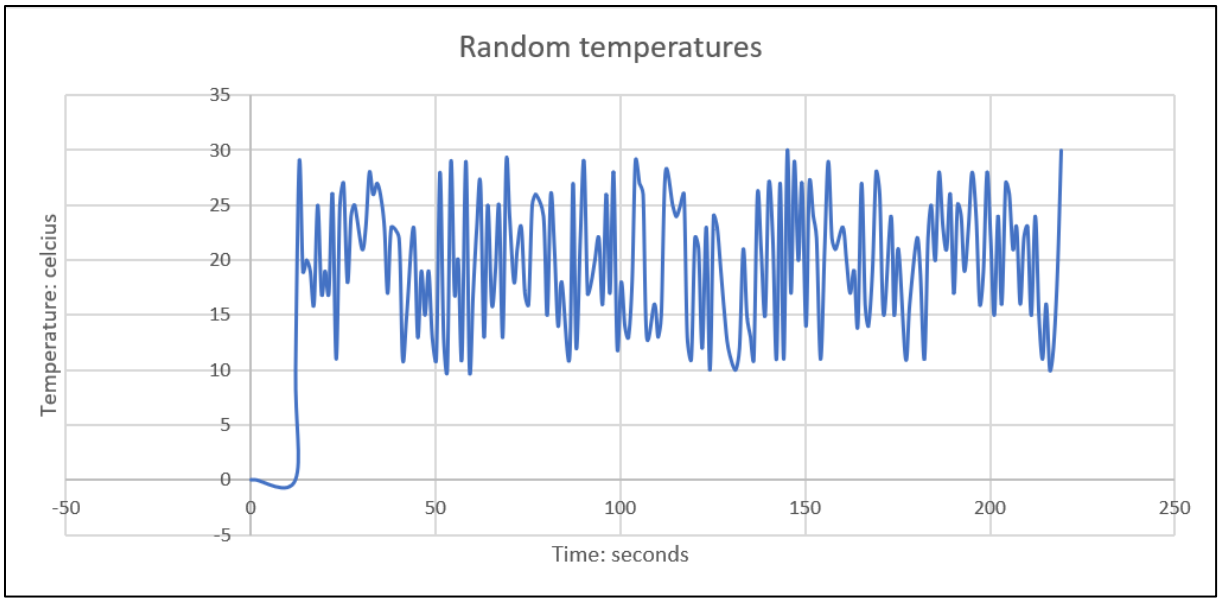


Figure 2-9 A plot of random data; generated, written to, and read from the OPC server.

2.4 OPC UA using LabVIEW

An OPC UA Server was created in LabVIEW using the OPC UA Toolkit. The front panel shown in Figure 2-10 shows server endpoint URL and a node Id for a tag created on the server. Figure 2-11 shows the according block diagram.

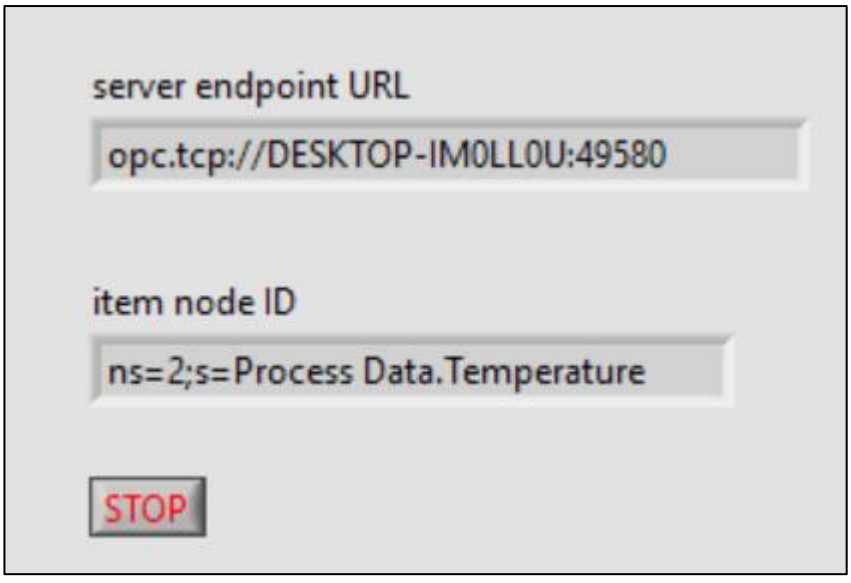


Figure 2-10 server and node info for the OPC UA server.

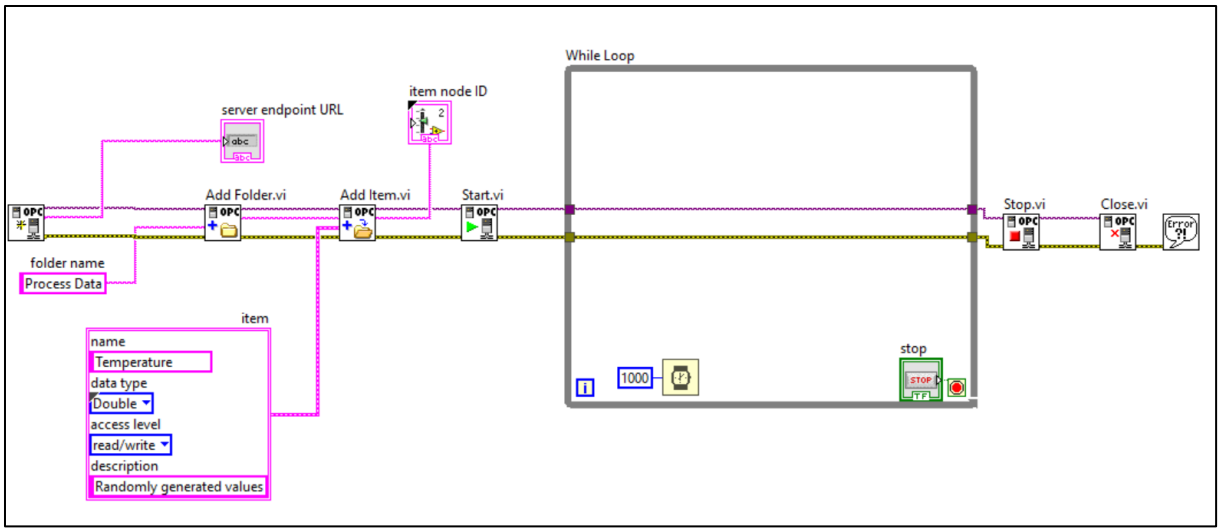


Figure 2-11 block diagram of the OPC UA server.

An OPC client for this server was also created as seen in Figure 2-12. This client takes random temperature data, in a similar fashion to the one created for the OPC DA client, and writes it to the tag.

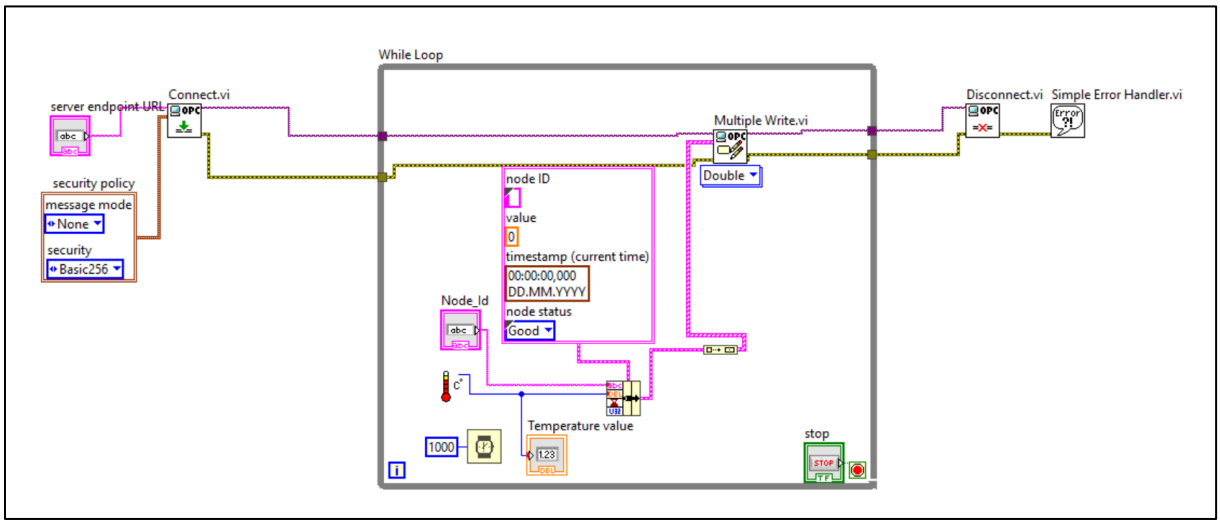


Figure 2-12 block diagram of an OPC UA client used to write data to the server.

Another client was created, used to read data from the OPC UA server. Figure 2-13 Shows the front panel and Figure 2-14 shows the block diagram for this client.

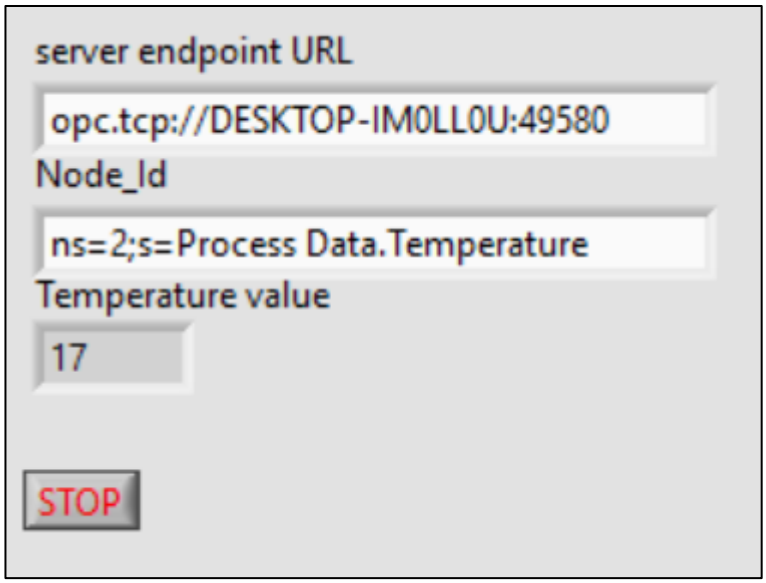


Figure 2-13 Front panel of the OPC UA client used for reading.

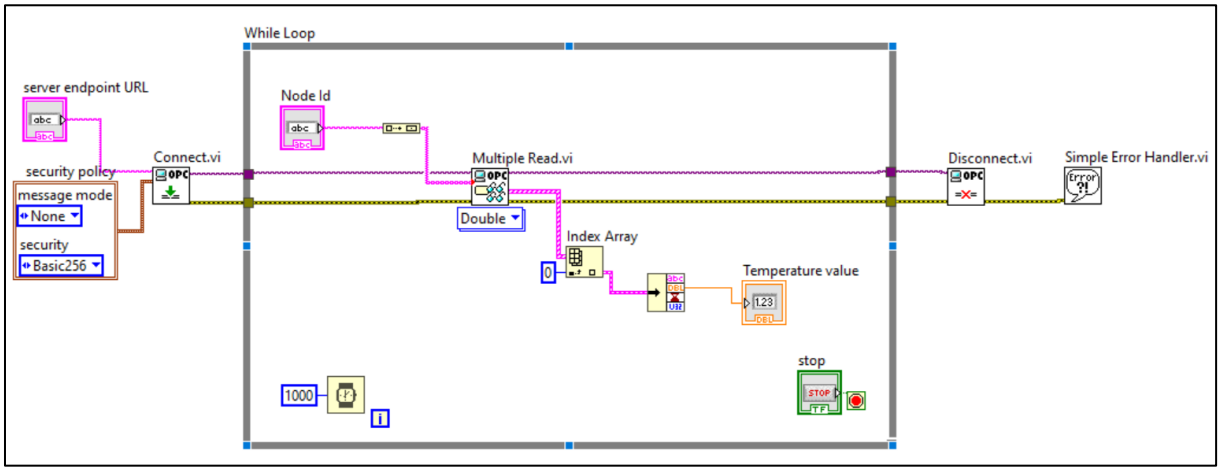


Figure 2-14 Block diagram of the OPC UA client used for reading.

2.5 OPC with Python

A final experiment was conducted to see if it was possible to read data using python. Using a library called “asyncua”. Data was easily read from the OPC server as seen in Figure 2-15. The code used for this can be found in Appendix A: OPC Client with Python.

```
tag1 is: ns=2;s=Process Data.Temperature with value 19.0
tag1 is: ns=2;s=Process Data.Temperature with value 25.0
tag1 is: ns=2;s=Process Data.Temperature with value 10.0
tag1 is: ns=2;s=Process Data.Temperature with value 17.0
tag1 is: ns=2;s=Process Data.Temperature with value 18.0
tag1 is: ns=2;s=Process Data.Temperature with value 20.0
tag1 is: ns=2;s=Process Data.Temperature with value 27.0
tag1 is: ns=2;s=Process Data.Temperature with value 16.0
tag1 is: ns=2;s=Process Data.Temperature with value 27.0
tag1 is: ns=2;s=Process Data.Temperature with value 19.0
tag1 is: ns=2;s=Process Data.Temperature with value 15.0
```

Figure 2-15

3 Summary

OPC client and server programs were used and created for both the OPC DA and UA versions.

Interoperability was no problem between MatrikonOPC and LabVIEW, and between LabVIEW and Python. There was very little overhead needed to communicate between the platforms.

Appendices

Appendix A: OPC Client with Python

4 Appendix A: OPC Client with Python

```
import asyncio
from asyncua.client import Client
import time

async def read_file():
    """https://github.com/FreeOpcUa/opcua-
asyncio/blob/master/examples/client_to_kepware.py"""
    url = "opc.tcp://DESKTOP-IM0LL0U:49580"
    async with Client(url=url) as client:
        while True:
        # Do something with client
            node = client.get_node('ns=2;s=Process Data.Temperature')
            print(f"tag1 is: {node} with value {await node.read_value()} ")
            time.sleep(1)
if __name__ == "__main__":
    asyncio.run(read_file())
```