

Esse artigo foi escrito pelo engenheiro Ismael Lopes da Silva, exclusivamente para o site "www.embarcados.com.br". OPC UA (Open Platform Communications Unified Architecture) é um protocolo para comunicação industrial, padronizado na IEC 62541. Pablo Melo escreveu um artigo para o Embarcados, vê-lo no link <https://www.embarcados.com.br/introducao-ao-opc-ua/>, que é uma ótima introdução. Minha intenção não é ser repetitivo, então, recomendo primeiramente ler o artigo citado. Nesse artigo quero focar no FreeOPCUA que é um projeto para implementar um Stack (pilha) OPC UA de código aberto (open source) e ferramentas associadas. Atualmente, FreeOpcUa é composto por uma biblioteca de Client e Server Python OPC-UA, escrita inteiramente em Python, que está disponível no Github.

Segue os principais links para familiarização:

- <http://freeopcua.github.io/>
- <https://github.com/FreeOpcUa/python-opcua>

O que temos que instalar e como, em ambas as máquinas (Desktop e Raspberry):

- 1) Python3
- 2) pip install freeopcua
- 3) pip install cryptography

Para exemplificar o uso do OPC UA Server e Client, utilizarei a configuração ilustrada na figura 1.

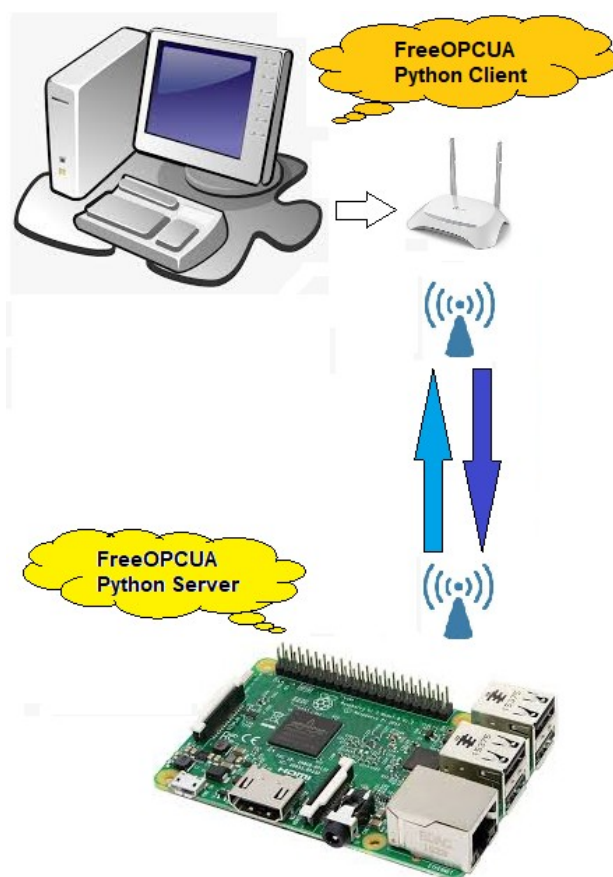


Figura 1 – Raspberry Server e PC Client

Segue o programa em Python escrito para o OPC UA Server, que rodará no Raspberry PI 3, com o sistema operacional Raspbian. O arquivo foi nomeado como Server.py, conforme mostrado a seguir:

```
# This code from https://github.com/FreeOpcUa
from opcua import ua, Server
import sys
import time
import datetime
sys.path.insert(0, "..")

if __name__ == "__main__":
    # setup our server
    server = Server()
    server.set_endpoint("opc.tcp://192.168.0.44:4840")
    # setup our own namespace, not really necessary but should as spec
    uri = "Ismael Lopes da Silva"
    idx = server.register_namespace(uri)
    # get Objects node, this is where we should put our nodes
    objects = server.get_objects_node()
    # populating our address space
    myobj = objects.add_object(idx, "MyObject")
    myData1 = myobj.add_variable(idx, "MyData1", 0)
    myDataDatetime = myobj.add_variable(idx, "MyDataDatetime", 0)
    myData1.set_writable() # Set MyVariable to be writable by clients
    myDataDatetime.set_writable() # Set MyVariable to be writable by clients
    # starting!
    server.start()
    try:
        count = 0
        while True:
            time.sleep(2)
            count = myData1.get_value()
            count += 0.1
            myDataDatetime.set_value(datetime.datetime.now())
            myData1.set_value(count)
    finally:
        #close connection, remove subscriptions, etc
        server.stop()
```

Segue o programa em Python escrito para o OPC UA Client, que rodará no Desktop, com o sistema operacional Windows 10. O arquivo foi nomeado como Client.py, conforme mostrado a seguir:

```
# This code from https://github.com/FreeOpcUa
from opcua import Client
import time
import sys
sys.path.insert(0, "..")

if __name__ == "__main__":
    client = Client("opc.tcp://192.168.0.44:4840")
```

```

try:
    # connecting!
    client.connect()
    # Client has a few methods to get proxy to UA nodes that should always be in address space
    root = client.get_root_node()
    myData1 = root.get_child(["0:Objects", "2:MyObject", "2:MyData1"])
    myDataDatetime = root.get_child(["0:Objects", "2:MyObject", "2:MyDataDatetime"])
    obj = root.get_child(["0:Objects", "2:MyObject"])
    print("myobj is: ", obj)
    print("myData1 is: ", myData1)
    print("myDataDatetime is: ", myDataDatetime)
    while True:
        print("myData1 = %4.1f" %client.get_node("ns=2;i=2").get_value())
        print("myDataDatetime = ", client.get_node("ns=2;i=3").get_value().strftime("%Y-%m-%d
        %H:%M:%S"))
        time.sleep(2)
finally:
    client.disconnect()

```

Nesta simples aplicação duas variáveis serão disponibilizadas pelo Server, que são "myData1" e "myDataDatetime". A variável "myData1" é inicializada com zero, e incrementada em 0.1 a cada dois segundos. A variável "myDataDatetime" é a data e hora de cada incremento. Primeiramente rode o Server, conforme ilustrado na figura 2, e depois rode o Client, figura 3.

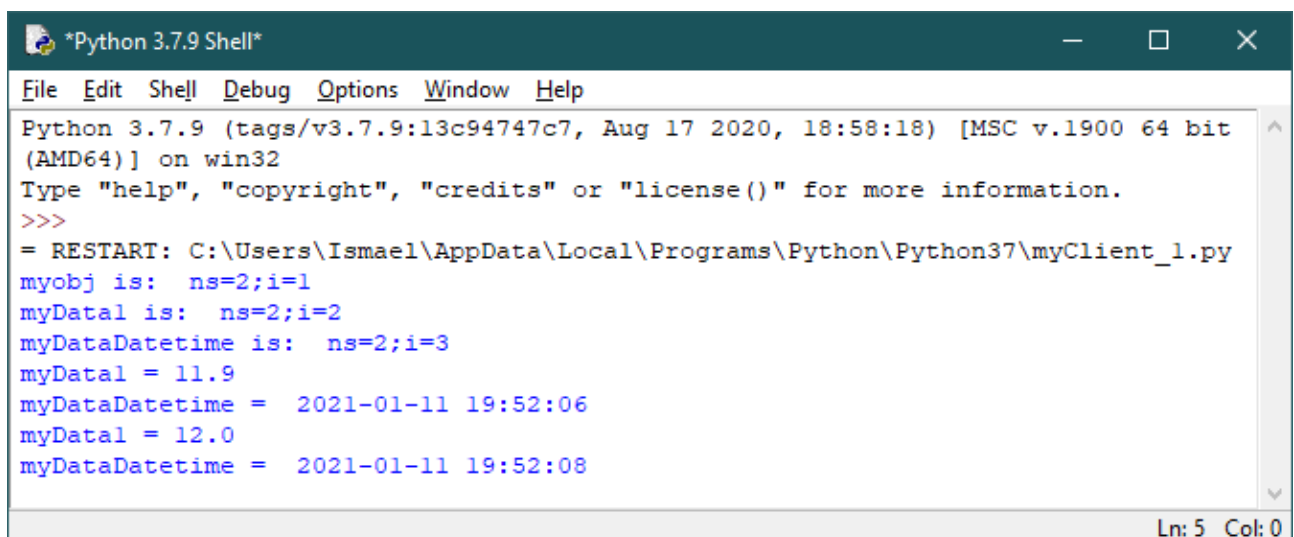


```

Running: Server.py
Endpoints other than open requested but private key and certificate are not set.
Listening on 192.168.0.44:4840

```

Figura 2 – Server rodando no Raspberry PI 3 Raspbian



```

Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ismael\AppData\Local\Programs\Python\Python37\myClient_1.py
myobj is: ns=2;i=1
myData1 is: ns=2;i=2
myDataDatetime is: ns=2;i=3
myData1 = 11.9
myDataDatetime = 2021-01-11 19:52:06
myData1 = 12.0
myDataDatetime = 2021-01-11 19:52:08

```

Figura 3 – Client rodando no Desktop Windows 10

Observe que é simples estabelecer uma conexão entre o Server e o Client. Com esses recursos podemos criar as mais variadas aplicações de forma simples e segura. O limite será sua criatividade.

Uma outra opção para validar o Server é baixar o software UAExpert Client, configurar e estabelecer uma conexão com o Server. Não vou detalhar os passos para configurar o UAExpert Client, porque não é escopo desse artigo, mas, tem alguns tutoriais disponíveis na internet, ver ilustração na figura 4.

O que podemos ver na figura 5, é a conexão estabelecida, entre o Client e Server, e as duas variáveis sendo consumidas pelo UAExpert Client. Acredito que temos o suficiente para dar uma boa noção de uma aplicação OPC UA Server e Client.

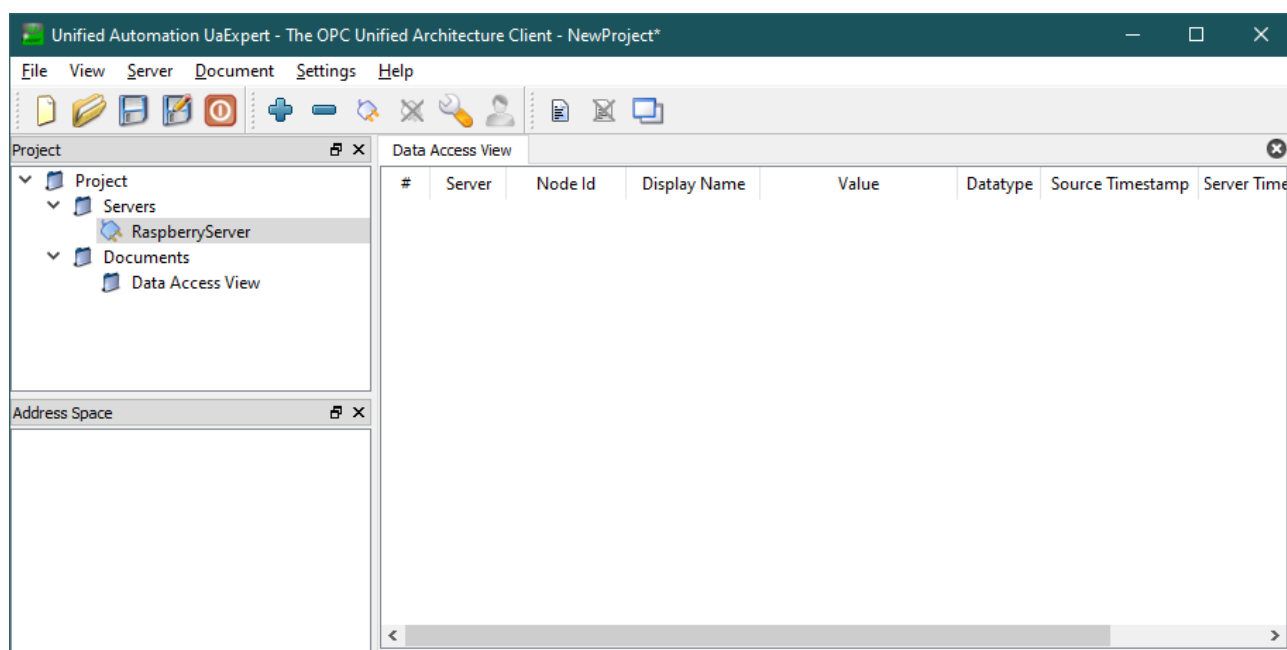


Figura 4 – UAExpert servidor configurado

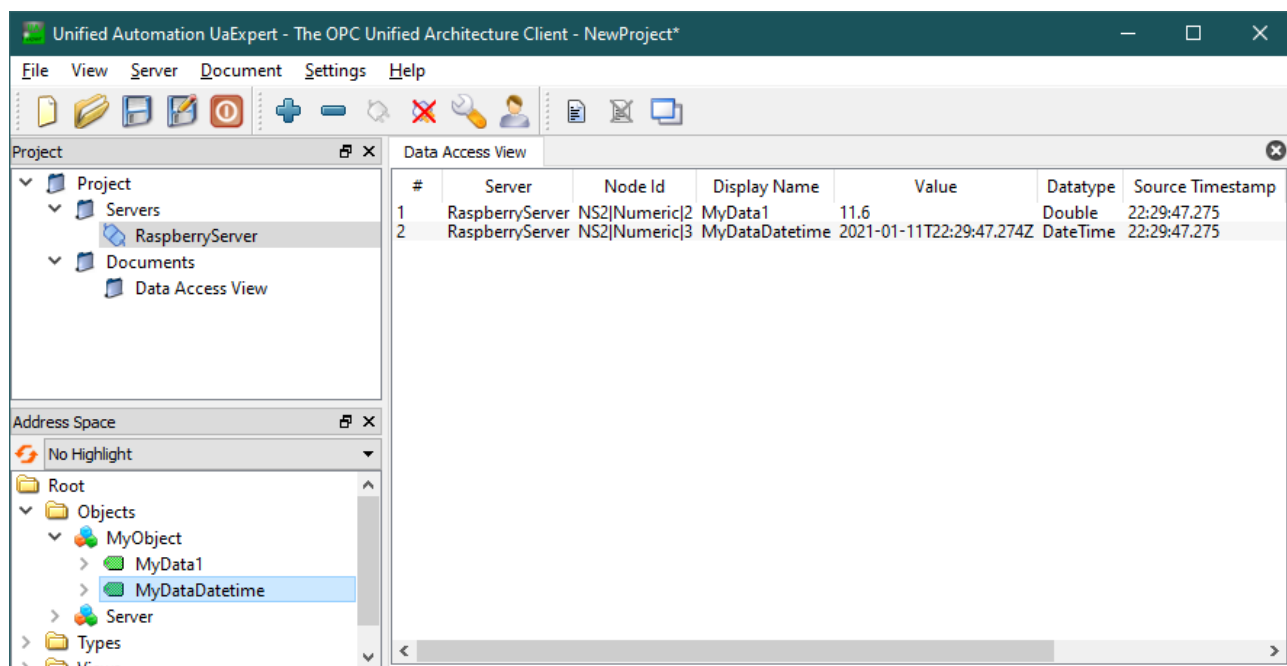


Figura 5 – Dados sendo lidos no Client (UAExpert)