How to connect and integrate an OPC UA server

by Marius Schmeding

Prerequisites

This lesson assumes that you want to integrate the **OPC Unified Architecture** (a.k.a OPC UA) protocol with the Cybus Connectware. To understand the basic concepts of the Connectware, please checkout the Technical Overview lesson. To follow along with the example, you will also need a running instance of the Connectware. If you don't have that, learn How to install the Connectware. Although we focus on OPC UA here, we will ultimately access all data via MQTT. So you should also be familiar with MQTT. If in doubt, head over to our MQTT Basics lesson.

Introduction

This article will teach you the integration of OPC UA servers. In more detail, the following topics are covered:

- Browsing the OPC UA address space
- Identifying OPC UA datapoints
- Specifying MQTT mappings
- Creating the Commissioning File
- Establishing a connection via the Connectware Admin UI
- Verifying data in the Connectware Explorer

The Commissioning Files used in this lesson are made available in the Example Files Repository on GitHub.

Selecting the tools

OPC UA server

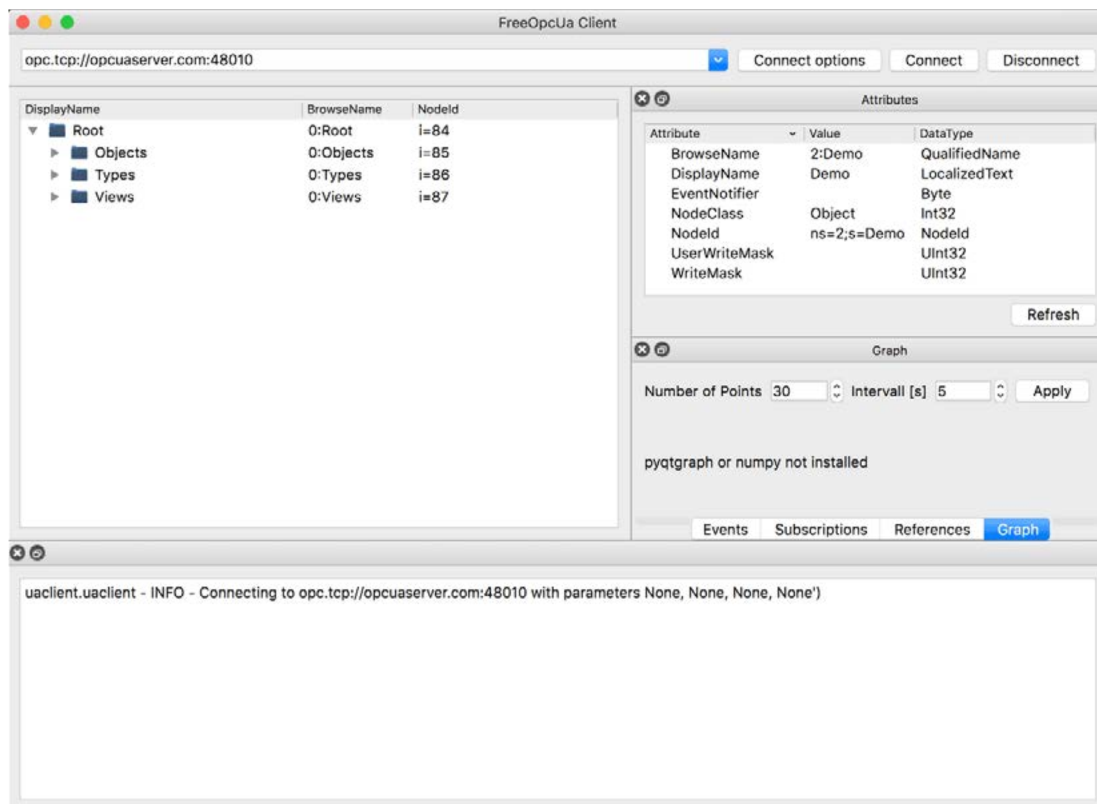
Our example utilizes the public OPC UA server at opcuaserver.com. This makes it very easy to get started, but please be warned, that a OPC UA server used in production should neither be exposed to the Internet, nor should allow anonymous access. In case you bring your own device instead (e.g. a Simatic S7-1500 PLC), that's perfectly fine. If so, please make sure that a basic TCP connection is possible.

OPC UA browser

We will use FreeOpcUa's Simple OPC UA GUI client for exploration in this guide. It is open source and available for all major OSs. If you feel more comfortable working on the terminal, go for Etienne Rossignon's opcua-commander. Or, if you prefer online tools, try One-Way Automation's OPC UA Web Client. It is free to use, but will ask you to sign up first and you won't be able to connect to servers on your local network. The choice is yours.

Exploring the OPC UA address space

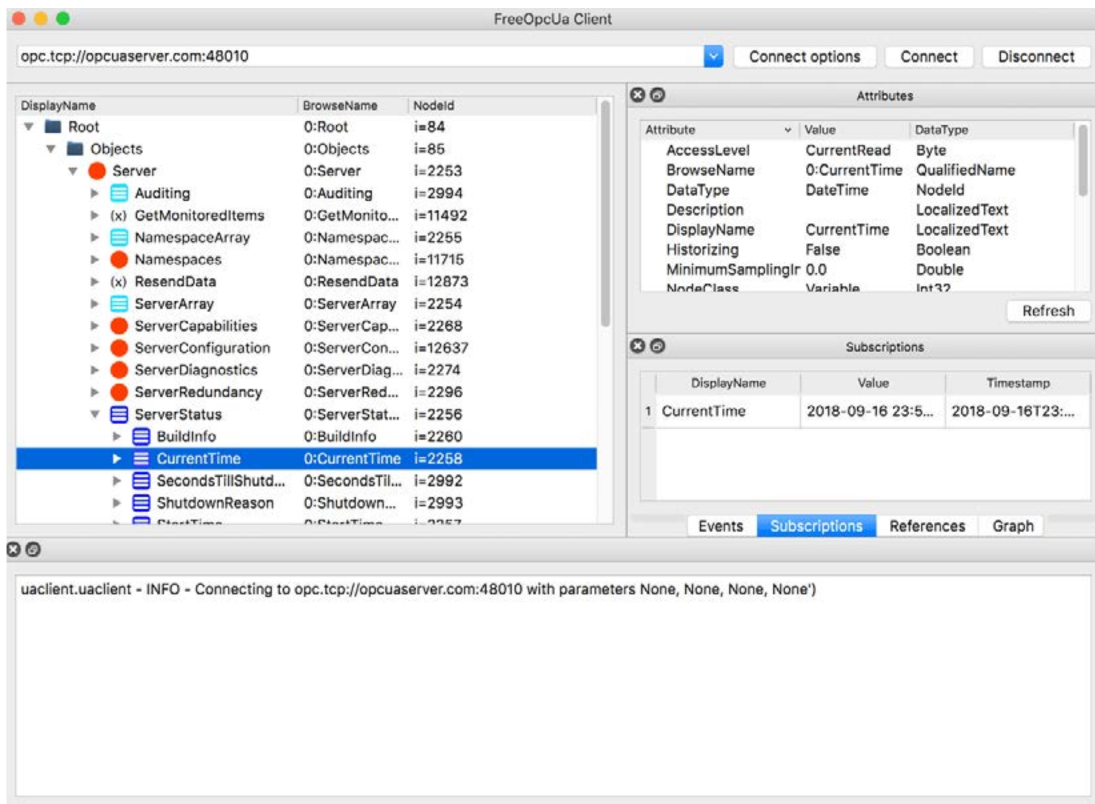
Now that we have assembled all our tools, let's get started! Run your OPC UA browser and connect to the endpoint `opc.tcp://opcuaserver.com:48010` (or whatever endpoint applies to you). Tada! – regardless of the chosen server/client, the three characteristic root nodes Objects, Types, and Views should show up, once the connection gets established. Explaining the OPC UA address space is certainly out of scope for this lesson – so we will only deal with the Objects tree here. But if you want to go further, then the Address Space Concepts documentation by Unified Automation is a good starting point.



Selecting datasource nodes

Go ahead and explore everything beneath Objects until you find the node displayed as `CurrentTime`. Select it and take a look at the Attributes pane – according to the NodeClass attribute we are dealing with a `Variable` here which implies, that there is a Value attribute as well. The Commissioning File, which we are about to create in the next step, will deal with `Variable` nodes. They allow us to read and write data, depending on their designated AccessLevel attribute.

Proceed now with a right click on `CurrentTime` and select subscribe to data change. The Subscriptions pane then shows the actual value and it is updating at a fast rate.



Now we know, that `Variable` nodes in the OPC UA address space are the datasources, but to utilize them with the Connectware, how do we reference them? There are two choices:

1. By `NodeId`, which is a node's unique identifier. For example, the `CurrentTime` variable's NodeId is `i=2258.`
2. By `BrowsePath`, which is the path of `BrowseName` when you navigate the treeview. For example, the `CurrentTime` variable's `BrowsePath` assembles to `/0:Objects/0:Server/0:ServerStatus/0:CurrentTime`.

Both approaches have their pros and cons. In general, the NodeId is less clumsy than the BrowsePath, but also less descriptive. In this example, we prefer the NodeId and preserve some of the tree structure semantics when we create the MQTT mappings.

Let's pick some more variables from the address space. Below you find a table of all the variables and their relevant data that will be used in the example. Pay attention to the last variable in the table. The Temperature Setpoint not only allows us to read data, but to also write data! That's pretty cool, since we can actually control something.

```
|Variable            |NodeId                                      |BrowsePath
|AccessLevel             |
|--------------------|--------------------------------------------|------------------------
---------------------------------------------------|-------------------------|
|CurrentTime         |i=2258                                      |/0:Objects/0:Server/0:Ser
verStatus/0:CurrentTime                           |CurrentRead              |
|Humidity            |ns=3;s=AirConditioner_1.Humidity            |/0:Objects/3:BuildingAuto
mation/3:AirConditioner_1/3:Humidity              |CurrentRead              |
|Power Consumption   |ns=3;s=AirConditioner_1.PowerConsumption    |/0:Objects/3:BuildingAuto
mation/3:AirConditioner_1/3:PowerConsumption      |CurrentRead              |
|Temperature         |ns=3;s=AirConditioner_1.Temperature         |/0:Objects/3:BuildingAuto
mation/3:AirConditioner_1/3:Temperature           |CurrentRead              |
|Temperature Setpoint |ns=3;s=AirConditioner_1.TemperatureSetPoint |/0:Objects/3:BuildingAuto
mation/3:AirConditioner_1/3:TemperatureSetPoint |CurrentRead, CurrentWrite|
```

Writing the Commissioning File

The Commissioning File contains all connection and mapping parameters and is read by the Connectware. To understand the file's anatomy in detail, please consult to the Reference docs. To get started, open a text editor and create a new file, e.g. `opcua-example-commissioning-file.yml`. The Commissioning File is in the YAML format, perfectly readable for human and machine! We will now go through the process of defining the required sections for this example:

- Description
- Metadata
- Parameters
- Resources

Description and Metadata

These sections contain more general information about the commissioning file. You can give a short description and add a stack of metadata. Regarding the metadata, only the name is required while the rest is optional. We will just use the following set of information for this lesson:

```
description: >
  OPC UA Example Commissioning File
  Cybus Learn - How to connect and integrate an OPC UA server
  https://learn.cybus.io/lessons/how-to-connect-and-integrate-an-opcua-server/

metadata:
  name: OPC UA Example Commissioning File
  version: 1.0.1
  icon: https://www.cybus.io/wp-content/uploads/2019/03/Cybus-logo-Claim-lang.svg
  provider: cybus
  homepage: https://www.cybus.io
```

Parameters

Parameters allow the user to customize Commissioning Files for multiple use cases by referring to them from within the Commissioning File. Each time a Commissioning File is applied or reconfigured in the Connectware, the user is asked to enter custom values for the parameters or to confirm the default values.

parameters:

```
  opcuaHost:
    type: string
    description: OPC UA Host Address
    default: opcuaserver.com

  opcuaPort:
    type: integer
    description: OPC UA Host Port
    default: 48010
```

We are defining the host address details of our OPC UA server as parameters, so they are used as default, but can be customized in case we want to connect to a different server.

Resources

In the resources section we declare every resource that is needed for our application. The first resource we need is a connection to the OPC UA server.

Cybus::Connection

```
resources:

  opcuaConnection:
    type: Cybus::Connection
    properties:
      protocol: Opcua
      connection:
        host: !ref opcuaHost
        port: !ref opcuaPort
        #username: myUsername
        #password: myPassword
```

After giving our resource a name – for the connection it is `opcuaConnection` – we define the `type` of the resource and its type-specific `properties`. In case of `Cybus::Connection` we declare which `protocol` and connection parameters we want to use. For details about the different resource types and available protocols, please consult the Reference docs. For the definition of our connection we reference the earlier declared parameters `opcuaHost` and `opcuaPort` by using `!ref`. If you are using a username and password, you could also create parameters, to make them configurable. In our case the server does not require credentials.

Cybus::Endpoint

The next resources needed are the datapoints that we have selected earlier. Let's add each OPC UA node by extending our list of resources with some endpoints.

```
currentTime:
    type: Cybus::Endpoint
    properties:
      protocol: Opcua
      connection: !ref opcuaConnection
      subscribe:
        nodeId: i=2258

  Humidity:
    type: Cybus::Endpoint
    properties:
      protocol: Opcua
      connection: !ref opcuaConnection
      subscribe:
        browsePath: /0:Objects/3:BuildingAutomation/3:AirConditioner_1/3:Humidity

  PowerConsumption:
    type: Cybus::Endpoint
    properties:
      protocol: Opcua
      connection: !ref opcuaConnection
      subscribe:
        nodeId: ns=3;s=AirConditioner_1.PowerConsumption
```

```
Temperature:
  type: Cybus::Endpoint
  properties:
    protocol: Opcua
    connection: !ref opcuaConnection
    subscribe:
      nodeId: ns=3;s=AirConditioner_1.Temperature

TemperatureSetpointSub:
  type: Cybus::Endpoint
  properties:
    protocol: Opcua
    connection: !ref opcuaConnection
    subscribe:
      nodeId: ns=3;s=AirConditioner_1.TemperatureSetPoint

TemperatureSetpointWrite:
  type: Cybus::Endpoint
  properties:
    protocol: Opcua
    connection: !ref opcuaConnection
    write:
browsePath:/0:Objects/3:BuildingAutomation/3:AirConditioner_1/3:TemperatureSetPoint
```

Each resource of the type `Cybus::Endpoint` needs a definition of the used protocol and on which connection it is rooted. Here you can easily refer to the previously declared connection by using its name. Furthermore we have to define to which OPC UA node the endpoint should subscribe or write by giving the `nodeId` or the `browsePath`.

Cybus::Mapping

To this point we are already able to read values from the OPC UA server and monitor them in the Connectware Explorer or on the default MQTT topics related to our service. To achieve a data flow that would satisfy the requirements of our integration purpose, we may need to add a mapping resource to publish the data on topics corresponding to our MQTT topic structure.

mapping:

```
type: Cybus::Mapping
 properties:
   mappings:
     - subscribe:
         endpoint: !ref currentTime
       publish:
         topic: 'server/status/currenttime'
     - subscribe:
         endpoint: !ref Humidity
       publish:
         topic: 'building-automation/airconditioner/1/humidity'
     - subscribe:
         endpoint: !ref PowerConsumption
       publish:
         topic: 'building-automation/airconditioner/1/power-consumption'
     - subscribe:
         endpoint: !ref Temperature
       publish:
         topic: 'building-automation/airconditioner/1/temperature'
     - subscribe:
         endpoint: !ref TemperatureSetpointSub
       publish:
         topic: 'building-automation/airconditioner/1/temperature-setpoint'
     - subscribe:
         topic: 'building-automation/airconditioner/1/temperature-setpoint/set'
       publish:
         endpoint: !ref TemperatureSetpointWrite
```

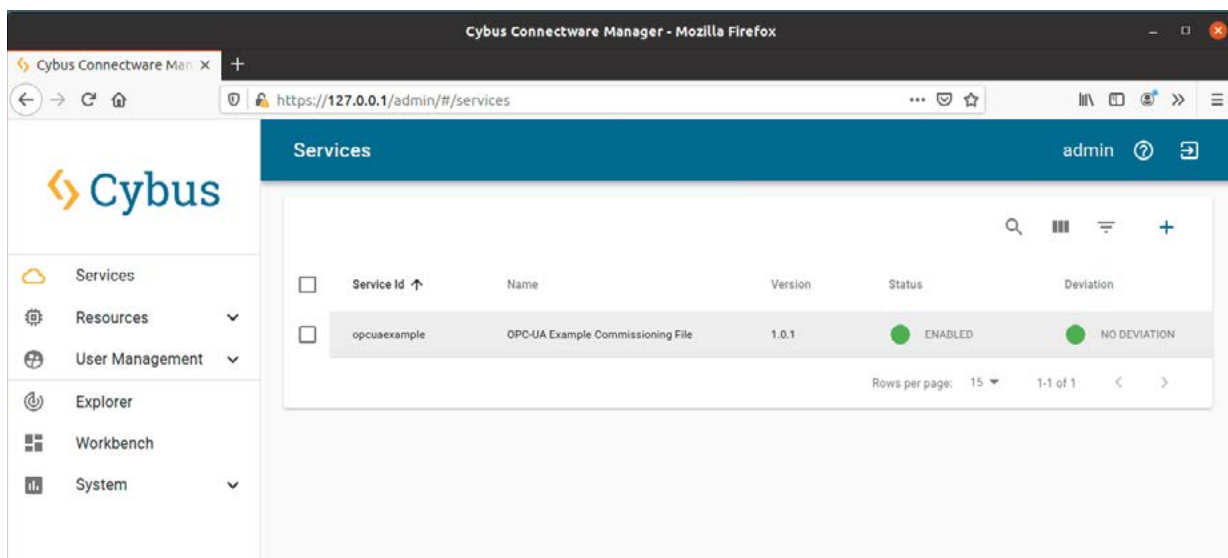
In this case the mapping defines which endpoints value is published on which MQTT topic. But as you may have noticed already, the Temperature Setpoint mapping is a bit different from the previous ones – here we have two endpoints: One monitoring the variable and the other writing to the variable, when a new setpoint is received via MQTT. To make the write operation work, we simply have to reverse the data flow and map it from the MQTT topic to the according endpoint. Then you can write to the endpoint by simply publishing on this MQTT topic. Please note that write messages have to be in JSON format containing the value as follows:

```
{
  "value": true
}
```

This message has the obligatory key „value", which can contain a value of any data type (which should match the concerning endpoint): For instance boolean values just use `true`/`false`, integers are represented by theirselves like `3`, decimals similarly appear as `10.45` and strings are written in `"quotes"`.

Installing the Commissioning File

You now have the Commissioning File ready for installation. Head over to the `Services` tab in the Connectware Admin UI and hit the `(+)` button to select upload the Commissioning File. You will be asked to set values for each member of the section `parameters` or confirm the default values. With a proper written Commissioning File, the confirmation of this dialog will result in the installation of a Service, which manages all the resources we just defined: The connection to the OPC server, the endpoints collecting data from the server and the mapping controlling where we can access this data. After enabling this Service you are good to go on and see if everything works out!



Verifying the data

Now that we have a connection established between OPC UA server and Connectware, we can go to the `Explorer` tab, where we see a tree structure of our newly created datapoints. Hover an entry and select the eye icon on the right – this activates the live view and you should see data coming in.

On MQTT topics the data is provided in JSON format and applications consuming the data must take care of JSON parsing to pick the desired properties. Though this JSON structure is not represented by the Explorer view. Taking our `currentTime` topic as an example, the raw output would look as follows:

```
{
  "value":"2020-07-14T11:43:06.632Z",
  "timestamp":1594726986632
}
```


Summary

We learned quite a few things here. First, we used an OPC UA client application to browse the address space. This enabled us to pick the variables of our interest and reference them by their NodeId, or by BrowsePath. Given that information, we created a Commissioning File and installed the Service on the Connectware. In the Explorer we finally saw the live data on corresponding MQTT topics and are now ready to go further with our OPC UA integration.

Where to go from here

The Connectware offers powerful features to build and deploy applications for gathering, filtering, forwarding, monitoring, displaying, buffering, and all kinds of processing data… why not build a dashboard for instance? For guides checkout more of Cybus Learn.

Cybus is a specialist for secure IIoT Edge software, headquartered in Germany. Cybus Connectware serves smart factories as a universal Edge and DevOps hub. Machine builders and providers of IIoT services use the Cybus Connectware as a software-based gateway. As early as 2017, Cybus published the first secure industrial connector for machine data according to today's DIN SPEC 27070 standard. Industry analyst Gartner named Cybus a worldwide "Cool Vendor". Today, the company counts medium-sized and large companies from numerous industrial sectors such as mechanical engineering, automotive and aviation among its customers.

Cybus GmbH · Osterstraße 124 · 20255 Hamburg · Germany · www.cybus.io · hello@cybus.io · (+49) 40 228 58 68 51