Haute Ecole Arc - Ingénierie

Guide d’utilisateur  
Connected Factory

Travail d’Automne 2016 – Projet 15DLM-TA655

Superviseur : Huber Droz

# Table des matières

[1. Table des matières 3](#_Toc473653639)

[2. Introduction 4](#_Toc473653640)

[3. Client 4](#_Toc473653641)

[3.1. Handshake 4](#_Toc473653642)

[3.2. Lecture et modification de variables 4](#_Toc473653643)

[3.3. Abonnement 5](#_Toc473653644)

[3.4. Appel d’une fonction serveur 7](#_Toc473653645)

[3.5. Historique d’une valeur 7](#_Toc473653646)

[4. Server 7](#_Toc473653647)

[4.1. Configuration générique 8](#_Toc473653648)

[4.2. Instanciations d’un nœud 8](#_Toc473653649)

[5. Références 9](#_Toc473653650)

# Introduction

Ce guide détaille l’implémentation de la démonstration liée au projet Connected Factory. Le but est de facilité l’accès à la spécification OPC UA par l’étudiant qui continuera ce projet. Le langage n’a pas un grand impacte sur la spécification donc ce document reste pertinent même si l’utilisateur futur souhaite utiliser une implémentation avec un autre langage. L’implémentation de base utilisée ici est celle du projet FreeOpcUa[[1]](#footnote-1).

La démonstration est composée de deux fichiers : client\_demo.py et server\_demo.py.  
La première partie de ce guide détaille ces deux fichiers.  
La deuxième partie donne des astuces liées à l’utilisation de spécifique de l’implémentation python.

# Client

Le client est instancié au travers d’une classe générique Client qui permet de centraliser et de faire abstraction de toutes les opérations liées au réseau.

## Handshake

Premièrement le client doit se connecter au serveur. Le constructeur de la classe Client prend l’adresse http du serveur en paramètre.

client **=** Client**(**"opc.tcp://localhost:4840/connected-factory/server/"**)**client**.**connect**()**

## Lecture et modification de variables

Le client peut ensuite récupérer le nœud racine du serveur. A partir de là tout est possible car le nœud racine permet d’accéder à tous les objets et variables du serveur.

root **=** client**.**get\_root\_node**()**

Dans mon exemple je veux accéder à l’objet robot. Le chemin relatif de l’objet à partir du nœud racine sera 0 :Objects puis 2 :Robot1.

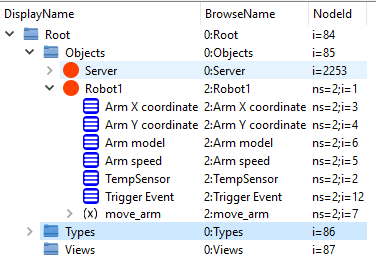


Figure 1 Structure du serveur de démonstration sur le client graphique[[2]](#endnote-1)

robot **=** root**.**get\_child**([**"0:Objects"**,** "2:Robot1"**])**

La fonction Node.get\_child() permet de récupérer la référence sur un nœud spécifique. La référence robot chez le client pointe sur l’instance de l’objet 1 :Robot1 sur le serveur. Le client peut donc directement modifier l’instance 1 :Robot1. Si le serveur modifie l’instance sans prévenir le client, le récupérera automatiquement les nouvelles valeurs. En effet, les méthode d’accès au valeurs des variables et propriétés de l’objet récupère leur valeur sur le serveur et non pas dans le cache du client.

L’accès et la modification d’une variable se fait de la manière suivante :

Remarque : si la variable est protégée en écriture l’exception  « BadUserAccessDenied » sera levée.

arm\_speed **=** robot**.**get\_child**([**"2:Arm speed"**])**

new\_speed **=** 15

arm\_speed**.**set\_value**(**new\_speed**)**

## Abonnement

Afin de suivre la modification d’une variable, il faut s’abonner à celle-ci. Le serveur enverra une notification pour chaque modification de la variable pendant le temps de l’abonnement.

Premièrement il faut déclarer un handler qui est une classe implémente l’interface SubHandler qui comprends les fonctions la fonction :

datachange\_notification**(**node**,** val**,** data**)**

event\_notification**(**event**)**

status\_change\_notification**(**status**)**

La fontion datachange\_notification est appelée quand la variable à laquelle le handler est abonnée est modifiée.  
L’argument node est une référence au nœud dont la valeur a été modifiée.  
L’argument val est la nouvelle valeur du nœud.  
L’exemple suivant montre un handler qui affiche un message dans la console lorsque la température est modifié.

**class** **TemperatureSubHandler(**object**):**

*@staticmethod*

**def** datachange\_notification**(**node**,** val**,** data**):**

**print(**"The Temperature of the robot is now {0:3.1f}."**.**format**(**val**))**

La fonction event\_notification est appelée lorsque que le serveur auquel le handler est abonné génère un événement.  
L’argument event permet de récupérer le nom de l’événement ainsi que ses propriétés.  
L’exemple ci-dessous affiche un message dans la console qui dépend de la valeur d’une propriété de cet événement.

**class** **ServerEventSubHandler(**object**):**

*@staticmethod*

**def** event\_notification**(**event**):**

**if** event**.**IsInUse**:**

**print(**"Event received from {0}: {1} Current power level is {2}% and is being used."

**.**format**(**event**.**SourceName**,** event**.**Message**,** event**.**PowerLevel**))**

La fonction status\_change\_notification est utilisée pour suivre les changements d’état du serveur.

Maintenant, nous devons instancier un abonnement qui utilise une de nos classes handler. Pour ce faire, on utilise la fonction Client.create\_subscritpion(periode, handler). L’argument periode n’a pas d’effet à ma connaissance. Finalement, l’objet sub est lié à la variable qu’elle doit monitorer.

handler **=** TemperatureSubHandler**()**

sub **=** client**.**create\_subscription**(**1**,** handler**)**

handle **=** sub**.**subscribe\_data\_change**(**root**.**get\_child**([**"0:Objects"**,** "2:Robot1"**,** "2:TempSensor"**]))**

sub**.**unsubscribe**(**handle**)**

Cette exemple illustre l’abonnement à la variable TempSonsor.  
La variable handle est uniquement utilisée pour se désabonner d’une variable.  
  
L’abonnement aux événement du serveur se fait de façon similaire.

low\_power\_event **=** root**.**get\_child**([**"0:Types"**,** "0:EventTypes"**,** "0:BaseEventType"**,** "2:Low Power Event"**])**

handler **=** ServerEventSubHandler**()**

sub **=** client**.**create\_subscription**(**500**,** handler**)**

handle **=** sub**.**subscribe\_events**(**robot**,** low\_power\_event**)**

La fonction subscribe\_events(sourcenode, evtype) prend en paramètre le nœud qui a généré l’événement. L’argument evtype permet de spécifier le type de l’événement. Le type de l’événement est en réalité la même information que son emplacement dans l’adresse de nom du serveur ( voir Figure 1).

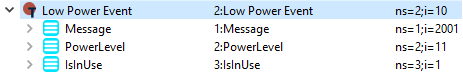


Figure 3 Evénement personlalisé comprennant trois propriétés.

## Appel d’une fonction serveur

L’appel d’une fonction se fait à l’aide de la fonction Node.call\_method(methodid, \*args). Elle prend en argument l’identifiant de la méthode et les arguments de la fonction.

robot**.**call\_method**(**"2:move\_arm"**,** x**,** y**)**

Code 1 Appel la fonction move\_arm du nœud Robot1

## Historique d’une valeur

La fonction Node.read\_raw\_history(starttime=None, endtime=None, numvalues=0) permet de récupérer l’historique des modifications du nœud courant. Il est possible de choisir le début et la fin de l’échantillonnage ainsi que le nombre de mesures. Les valeurs par défaut permettent de récupérer tous les enregistrements de cette variable par le serveur jusqu’au moment de l’appel. Elle retourne une liste d’objet possédant les propriétés suivantes : Encoding, ServerPicoseconds, ServerTimestamp, SourcePicoseconds, SourceTimestamp, StatusCode, Value. Le code illustre l’utilisation de certaines de ces propriétés.

history = temp\_sensor.read\_raw\_history(numvalues=5)

for temp in reversed(history):

print("Temperature at {0} was {1:3.1f}°.".format(str(temp.SourceTimestamp), temp.Value.Value))

# Server

Le server est représenté par la classe Server. Elle fait abstraction de toutes les fonctionnalités reseaux et permet d’accéder à l’espace d’adressage du serveur. Celui-ci contient l’arborescence des nœuds.

## Configuration générique

Il faut attribuer une adresse IP au serveur ainsi qu’une URI[[3]](#footnote-2). L’URI n’est pas nécessaire au fonctionnement du serveur mais fait partie de la spécification et permet d’avoir accès à une page internet donnant des informations sur le serveur.

server **=** Server**()**

server**.**set\_endpoint**(**"opc.tcp://0.0.0.0:4840/connected-factory/server/"**)**

uri **=** "https://github.com/thegazou/connected-factory"

idx **=** server**.**register\_namespace**(**uri**)**

## Instanciations d’un nœud

Tout d’abord il nous faut récupérer le nœud Objet de base puis nous allons y ajouter des variables, objets ou propriétés.

# Références

1. <https://github.com/FreeOpcUa/python-opcua> [↑](#footnote-ref-1)
2. projet python [FreeOpcUa/opcua-client-gui](https://github.com/FreeOpcUa/opcua-client-gui) [↑](#endnote-ref-1)
3. [Identifiant une ressource sur un réseau.](https://fr.wikipedia.org/wiki/Uniform_Resource_Identifier) [↑](#footnote-ref-2)