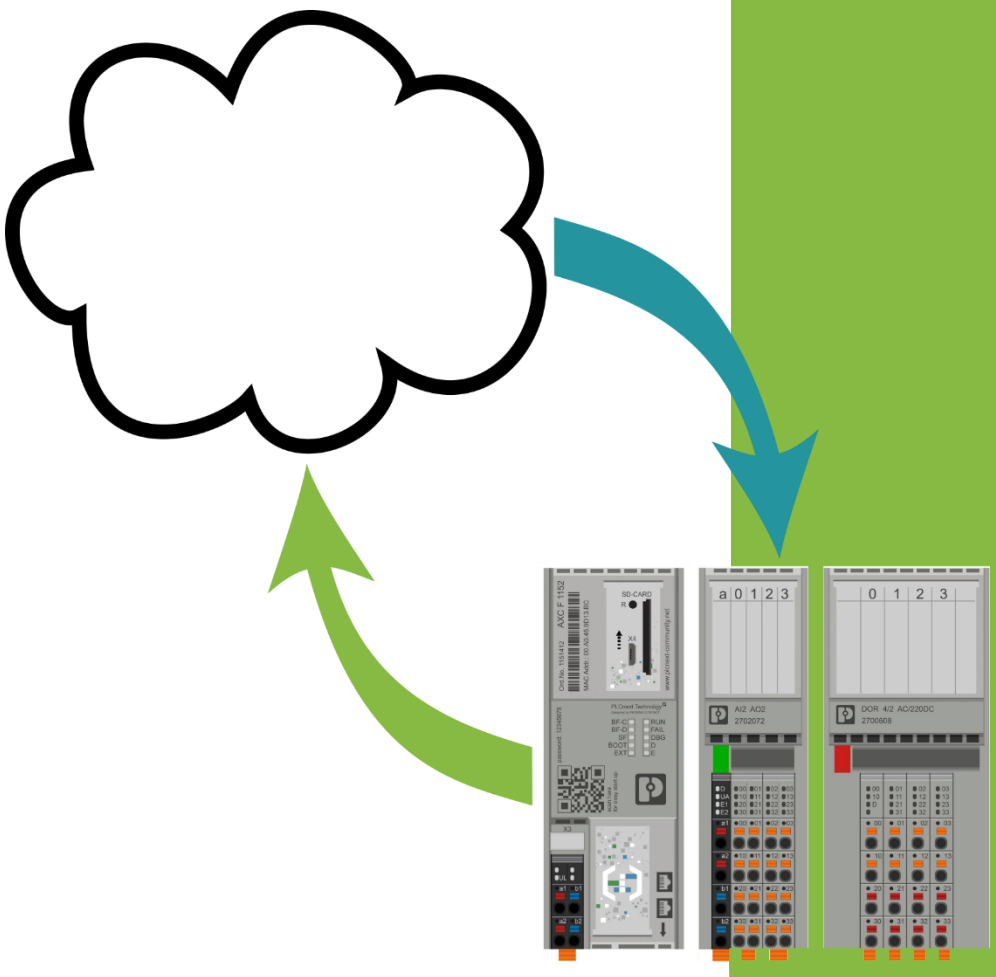
# IIOT Cloud-Integration einer SPS mit OPC UA und Proficloud.io



Tim Aschwanden

Bachelor-Thesis

Hochschule Luzern

09.06.2022

# Bachelor-Thesis an der Hochschule Luzern - Technik & Architektur

| | |
|---|---|
| **Titel** | **IIOT Cloud-Integration einer SPS mit OPC UA und Proficloud.io** |
| **Diplomandin/Diplomand** | **Aschwanden, Tim Benjamin** |
| **Bachelor-Studiengang** | **Bachelor Elektrotechnik und Informationstechnologie** |
| **Semester** | **FS22** |
| **Dozentin/Dozent** | **Kasten, Oliver** |
| **Expertin/Experte** | **Wegmüller, Marc** |


**Abstract Deutsch**
Im Zuge der Digitalisierung und der stetig intensiveren Automation in der Industrie sollen die einzelnen Maschinen und Prozesse miteinander vernetzt werden, um die Produktivität und die Effektivität zu steigern. In dieser Bachelor-Thesis wurde dafür eine SPS von Phoenix Contact in das Industrial Internet of Things (IIoT) integriert. Auf der einen Seite wurde diese Integration mit dem plattformunabhängigen Kommunikations-Service OPC UA gemacht und auf der anderen Seite über den Cloudservice Proficloud.io. Für beide dieser Kommunikations-Möglichkeiten wurde jeweils eine C#-Konsolenanwendung erstellt, um grundlegende Befehle wie das Schreiben und Lesen von Werten, umsetzen zu können. Dabei lag der Fokus auf simpler und verständlicher Implementation sowie Dokumentation, damit der Code und die Dokumentation von Kunden und Studierenden für den Einstieg wiederverwendet werden kann. Zur Veranschaulichung der gesamten Kommunikationen wurde ausserdem ein Prototyp mit einer SPS, einem Lichtsensor und einer Steckdose erstellt.

**Abstract Englisch**
In the course of digitalization and the ever more increasing automation in the industry, individual machines and processes are networked among each other in order to increase productivity and effectiveness. In this Bachelor Thesis, a PLC from Phoenix Contact was integrated into the Industrial Internet of Things (IIoT). On the one hand, this integration was done with the platform-independent communication service OPC UA and on the other hand via the cloud service Proficloud.io. For both of these communication options, a C# console application was created to implement the basic commands like writing and reading values. The focus lied on simple and understandable implementation as well as documentation, so that the code and documentation can be reused by customers and students to get started. To illustrate all the communications, a prototype was also created with a PLC, a light sensor and a power socket.


Ort, Datum Seedorf, 09.06.2022

# Inhaltsverzeichnis

# 1. Abkürzungsverzeichnis

API ..... Application Programming Interface

GUID ..... Global Unique Identifier

HTTP ..... Hypertext Transfer Protocol

IIOT ..... Industrial Internet of Thigs

LD ..... Ladder

NOLD ..... Network Oriented Ladder

OPC UA ..... Open Platform Communications Unified Architecture

PLC ..... Programmable Logic Controller

REST ..... Representational State Transfer

SPS ..... Speicherprogrammierbare Steuerung

ST ..... Structured Text

UUID ..... Universally Unique Identifier

WBM ..... Web Based Management

WWW ..... World Wide Web

# 2. Einleitung

Diese Dokumentation enthält die Bachelor-Thesis von Tim Aschwanden über die IIOT Cloud-Integration einer SPS mit OPC UA und Proficloud.io.

Im Zuge der Digitalisierung und der stetig intensiveren Automation in der Industrie sollen die einzelnen Maschinen und Prozesse miteinander vernetzt werden, um die Produktivität und die Effektivität zu steigern. Sehr oft ist die Intelligenz beziehungsweise die Steuerung einer Maschine in der Industrie eine speicherprogrammierbare Steuerung (SPS), welche überwacht und gewartet werden muss. Dies geschieht mit unterschiedlichsten Technologien und Kommunikationsprotokollen, mit gewissen Vor- und Nachteilen.

In dieser Bachelor-Thesis liegt der Fokus auf einer Dokumentation, welche Kunden und Studierenden die Kommunikation mit einer SPS über OPC UA oder über eine Cloud mit REST API (Representational State Transfer Application Programming Interface) Befehlen näher bringen soll. Diese Dokumentation beinhaltet verschiedene Befehle zwischen der SPS und einem Computer sowie dessen Implementation und Anwendung. Ausserdem soll mit dieser Dokumentation die SPS-Konfiguration und Programmierung beschrieben werden. Des Weiteren soll ein simpler und benutzerfreundlicher Prototyp mit einer SPS und den passenden Ein- und Ausgangsmodulen von Phoenix Contact erstellt werden, um den Einstieg in die genannten Technologien zu veranschaulichen. Der Prototyp enthält einen Lichtsensor und eine Steckdose, um eine Beispiel-Anwendung zu präsentieren, welche die Steckdose bei einem gewissen Schwellwert mit einer Hysterese schaltet.

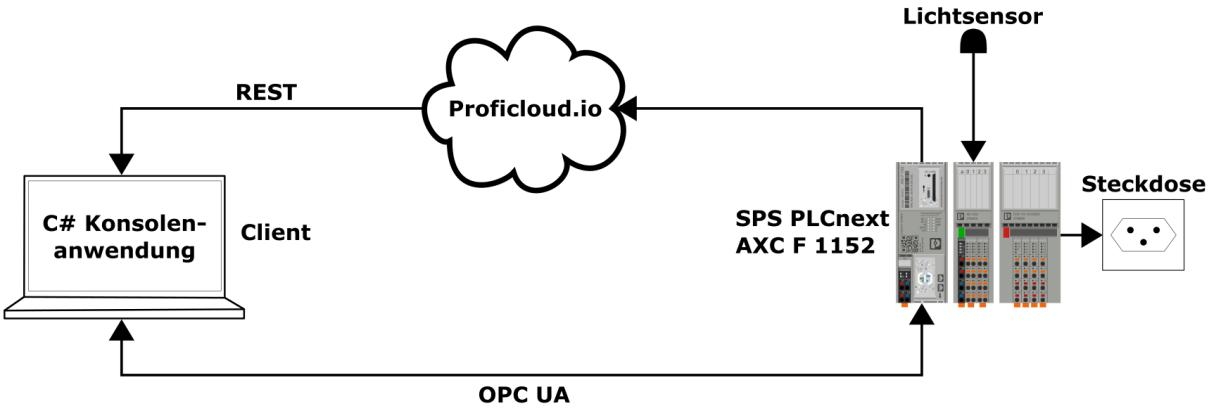



*Abbildung 1: Projektaufbau mit Client-Computer und SPS*

Der neuartige und immer wichtiger werdende plattformunabhängige Service «Open Platform Communications Unified Architectures» (OPC UA) bietet unzählige Möglichkeiten und Funktionen zur Implementierung einer Kommunikation zwischen zwei oder mehreren Teilnehmern in einem Netzwerk. Damit soll eine C#-Konsolenanwendung erstellt werden, mit welcher die Kommunikation zwischen dem OPC UA Server auf der SPS und dem OPC UA Client in der Konsolenanwendung hergestellt werden kann. Dabei sollen die wichtigsten Fähigkeiten dieses Service, wie zum Beispiel das Lesen und Schreiben von Variablen, nachvollziehbar implementiert und dokumentiert werden. Die in Abbildung 1 gezeigte Grafik soll den Aufbau des Projekts und des Prototyps verdeutlichen.

Um die Einsicht in die Prozessvariablen aus einem anderen Netz zu gewährleisten, soll Verwendung der Cloud Proficloud.io von Phoenix Contact implementiert und dokumentiert werden. Die SPS speichert dabei die Daten direkt auf der Cloud. Mit einer zweiten C#-Konsolenanwendung können dann die Daten mit REST API Befehlen eingesehen werden. Auch hier sollen die wichtigsten und Eigenschaften und Befehle nachvollziehbar implementiert und dokumentiert werden. In Abbildung 1 wird diese Verbindung durch den oberen Pfeil zwischen SPS und Computer repräsentiert.

# 3. Prototyp

In diesem Kapitel wird zum einen der Aufbau des Prototyps mit der benutzten SPS, den dazugehörenden Ein- und Ausgangsklemmen und der Schaltung zur Ansteuerung des Lichtsensors und der Steckdose beschrieben. Zum andern enthält dieser Abschnitt ein Beschrieb zur Inbetriebnahme der SPS, zur Konfigurierung der Klemmen und eine Erklärung der Beispielanwendung, welche auf der SPS läuft.

## 3.1. Prototyp Komponenten

Den Aufbau des Prototyps und die dafür verwendeten Komponenten werden in diesem Kapitel beschrieben. In einem ersten Schritt sind die bereits vorhandenen, beziehungsweise eingekauften Komponenten aufgelistet. Anschliessend wird die kleine, selbst hergestellte Schaltung zur Messung der Lichtintensität erläutert.

### 3.1.1. SPS PLCnext AXC F 1152

Abbildung 2 zeigt die verwendete Speicherprogrammierbare Steuerung PLCnext AXC F 1152 der Firma Phoenix Contact. Dabei handelt es sich um eine industrietaugliche Steuerung, welche für das simple und vielseitige IN/OUT-Klemmensystem Axioline F verwendet wird. Die SPS hat zwei Ethernet Anschlüsse und auf ihr läuft ein Linux Betriebssystem, was sie auch vielseitig einsetzbar macht. Das Datenblatt befindet sich in den technischen Dokumenten unter «Prototyp\SPS\Speisung». (Phoenix Contact, 2022)

*Abbildung 2: SPS PLCnext AXC F 1152*

### 3.1.2. Speisung STEP-PS/ 1AC/24DC/0.75

Um die Speisung der SPS, der Klemmen und der Beschaltung der Steckdose zu gewährleisten, wurde die Stromversorgung STEP-PS/ 1AC/24DC/0.75 von Phoenix Contact, welche in Abbildung 3 ersichtlich ist, verwendet. Bei einer Eingangsspannung von 100 V AC bis 240 V AC liefert diese Versorgung eine Ausgangsspannung von 24 V DC und einem maximalen Ausgangsstrom von 1.4 A. Das Datenblatt befindet sich in den technischen Dokumenten unter «Prototyp\SPS\Speisung». (Phoenix Contact, 2022)


*Abbildung 3: Speisung STEP-PS/ 1AC/24DC/0.75*

### 3.1.3. Analog-Klemme AXL F AI2 AO2 1H

Das analoge Ein- und Ausgabemodul AXL F AI2 AO2 1H des Klemmensystems Axioline F vom Hersteller Phoenix Contact ist in Abbildung 4 zu sehen. Diese Klemme hat zwei analoge Eingänge und zwei analoge Ausgänge, welche verwendet werden können, um Spannungen zwischen -10 V und 10 V auszugeben oder zu messen beziehungsweise Ströme zwischen -20mA und 20mA auszugeben oder zu messen. Das Datenblatt befindet sich in den technischen Dokumenten unter «Prototyp\SPS\ Klemmen». (Phoenix Contact, 2022)


*Abbildung 4: Analogklemme AXL F AI2 AO2 1H*

### 3.1.4. Relais-Klemme AXL F DOR4/2 AC/220DC 1F

Die Relaisklemme AXL F DOR4/2 AC/220DC 1F des Klemmensystems Axioline F vom Hersteller Phoenix Contact ist in Abbildung 5 zu sehen. Diese Klemme besitzt vier potentialfreie Schliesser-Relaisausgänge, um Spannungen bis zu 220 V DC oder bis zu 230 V AC zu schalten. Das Datenblatt befindet sich in den technischen Dokumenten unter «Prototyp\SPS\Klemmen». (Phoenix Contact, 2022)


*Abbildung 5: Relaisklemme AXL F DOR4/2 AC/220DC 1F*

### 3.1.5. Photodiode TEFD4300F

Zur Bestimmung der Lichtintensität, um anschliessend die Steckdose zu schalten, wurde die Photodiode TEFD4300F des Herstellers Vishay verwendet (Distrelec, 2022). Eine Photodiode wird in Sperrrichtung betrieben und in dieser Anwendung in Serie mit einem 560 kΩ Widerstand, um die Lichtänderung anhand der Spannungsänderung festzustellen. Durch die einfallenden Photonen des Lichtes werden in der Diode Ladungsträger erzeugt, welche aufgrund der angelegten Spannung transportiert werden und so zu einem Strom führen. Da dieser Strom sehr klein ist, muss ein grosser serieller Widerstand gewählt werden, um eine Spannungsänderung über dem Widerstand gut feststellen zu können. Grundsätzlich wird bei diesem Prinzip nicht die Lichtintensität in Lumen bestimmt, sondern einfach die Spannung über dem Widerstand, welche sich bei unterschiedlichen Lichtintensitäten verändert. Da das Dioden-Gehäuse nur UV-Strahlen durchlässt, muss sie mit Sonnenlicht bescheint werden. Das Datenblatt der Diode befindet sich in den technischen Dokumenten unter «Prototyp\Lichtsensor». (Ligitek, 2022)

*Abbildung 6: Photodiode TEFD4300F*

### 3.1.6. Gehäuse Photodiode

Damit die Photodiode und der serielle Widerstand sicher und gut sichtbar platziert werden können, wurde ein Komponentengehäuse von Phoenix Contact verwendet (Digitec, 2022). In diesem Gehäuse kann durch Anbringen von Klemmen auf einer passenden Leiterplatine die Schaltung platziert und einfach entfernt oder angepasst werden. Dazu wurde eine Laborkarte passend zugeschnitten (Abbildung 7 links) und dann mit den Komponenten und dem Deckel des Gehäuses bestückt (Abbildung 7 rechts).

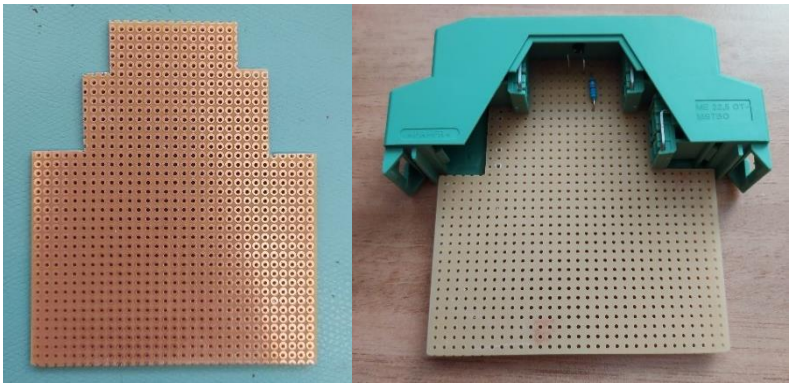


*Abbildung 7: Leiterplatine Gehäuse Photodiode*

Anschliessend kann diese Platine gemäss Abbildung 8 leicht in das Gehäuse geschoben werden und dann auf einer Schiene platziert werden.

*Abbildung 8: Gehäuse Photodiode*

## 3.2. Prototyp Aufbau

In diesem Kapitel wird gezeigt, wie der Aufbau des Prototyps am Schluss aussieht und wie die einzelnen Komponenten miteinander verbunden sind. Abbildung 9 zeigt eine Skizze aller verwendeten Komponenten und ihre Verdrahtung.

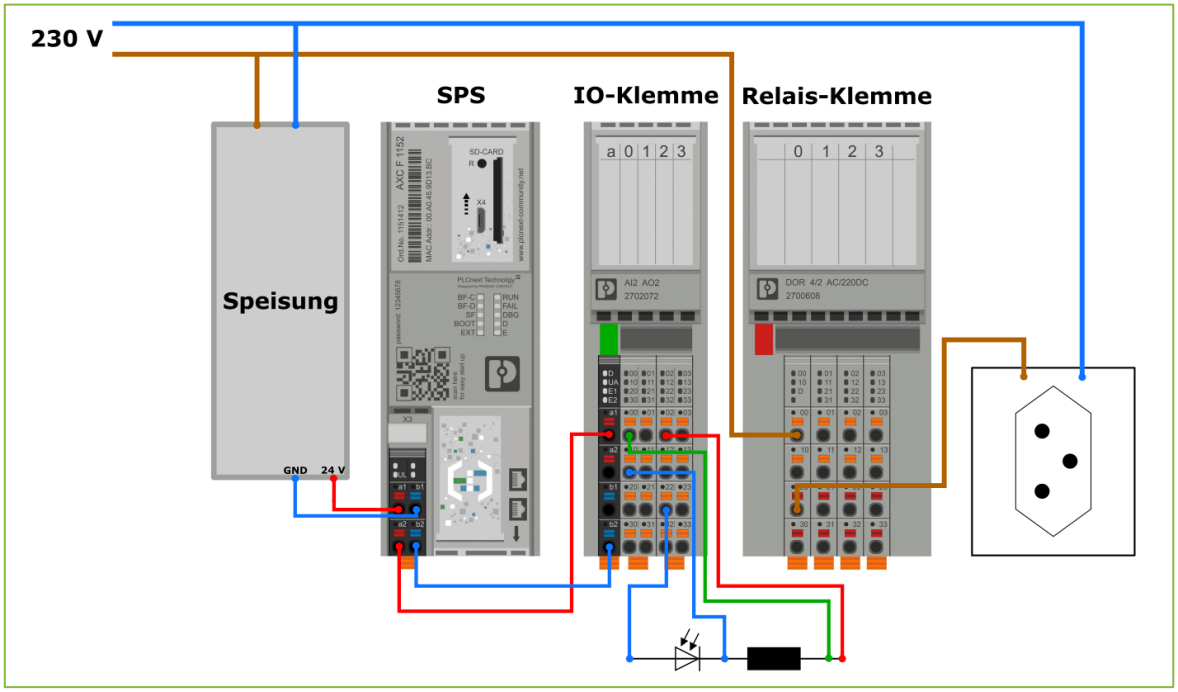



*Abbildung 9: Skizze Prototyp Aufbau*

Der fertig gestellte und voll funktionstüchtige Prototyp ist in Abbildung 10 zu sehen.

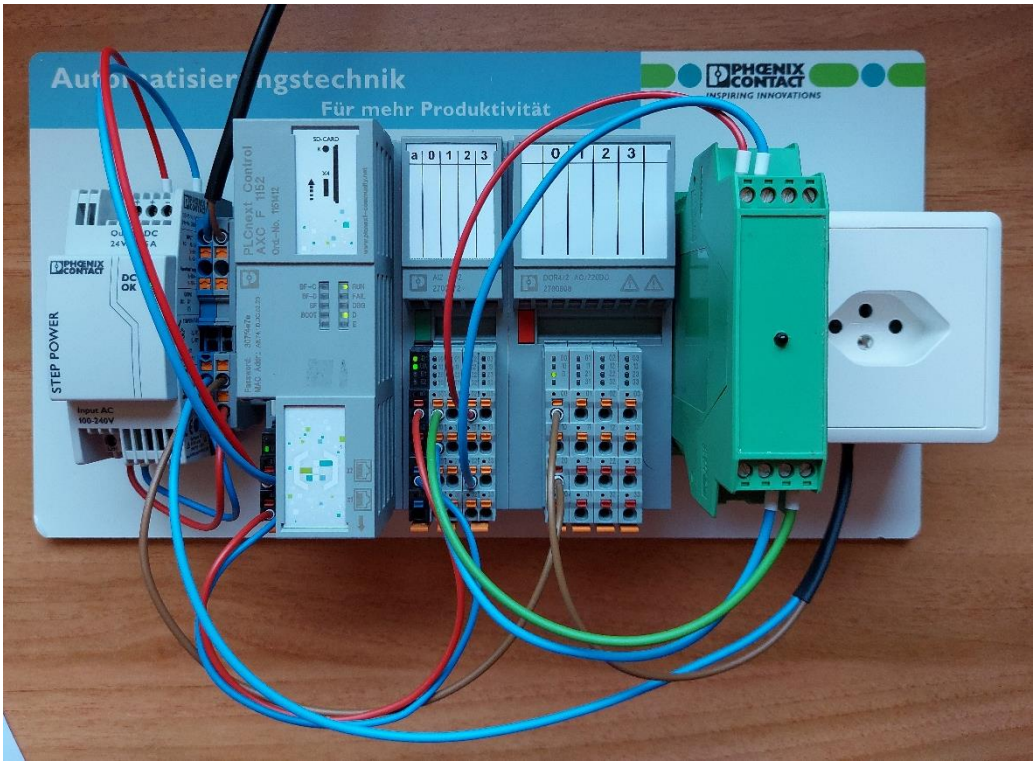



*Abbildung 10: Prototyp Aufbau*

## 3.3. Prototyp Beispielanwendung

In diesem Kapitel liegt der Fokus auf dem Einstieg in die Software zur SPS-Konfiguration und Programmierung PLCnext Engineer der Firma Phoenix Contact. Dafür werden in den folgenden Abschnitten die wichtigsten Funktionen von PLCnext Engineer kurz erklärt. Ausserdem wird die SPS mit den Klemmen konfiguriert und die Beispielanwendung darauf implementiert. Diese Anwendung soll die Spannung über dem Widerstand, welcher mit der Photodiode in Serie ist, messen und somit eine Lichtintensität erhalten. Anhand dieser Intensität soll dann die Steckdose über die Relais-Klemme mit einer Hysterese bei Dunkelheit eingeschaltet beziehungsweise bei Helligkeit ausgeschaltet werden. Die folgende Grafik (Abbildung 11) soll das Verhalten der Steckdose mit einer Hysterese bei unterschiedlichen Lichtintensitäten verdeutlichen. Sobald die Lichtintensität (grün) unter die «w_HystOn» Grenze geht, wird die Steckdose (gelb) eingeschaltet. Sobald die Lichtintensität dann wieder über die «w_HystOff» Grenze steigt, wird die Steckdose ausgeschaltet. Mit solch einer Hysterese wird verhindert, dass die Steckdose innert kürzester Zeit mehrmals ein- und ausschaltet, wie es bei einem einfachen Schwellwert der Fall wäre, wenn die Lichtintensität gerade in diesem Bereich befindet. Hierbei muss beachtet werden, dass in dieser Grafik nur jede Sekunde ein Datenpunkt vorhanden ist und dazwischen eine lineare Interpolation vorliegt und deshalb Hysterese-Grenzen nicht exakt mit den kontinuierlichen Kurven übereinstimmen.

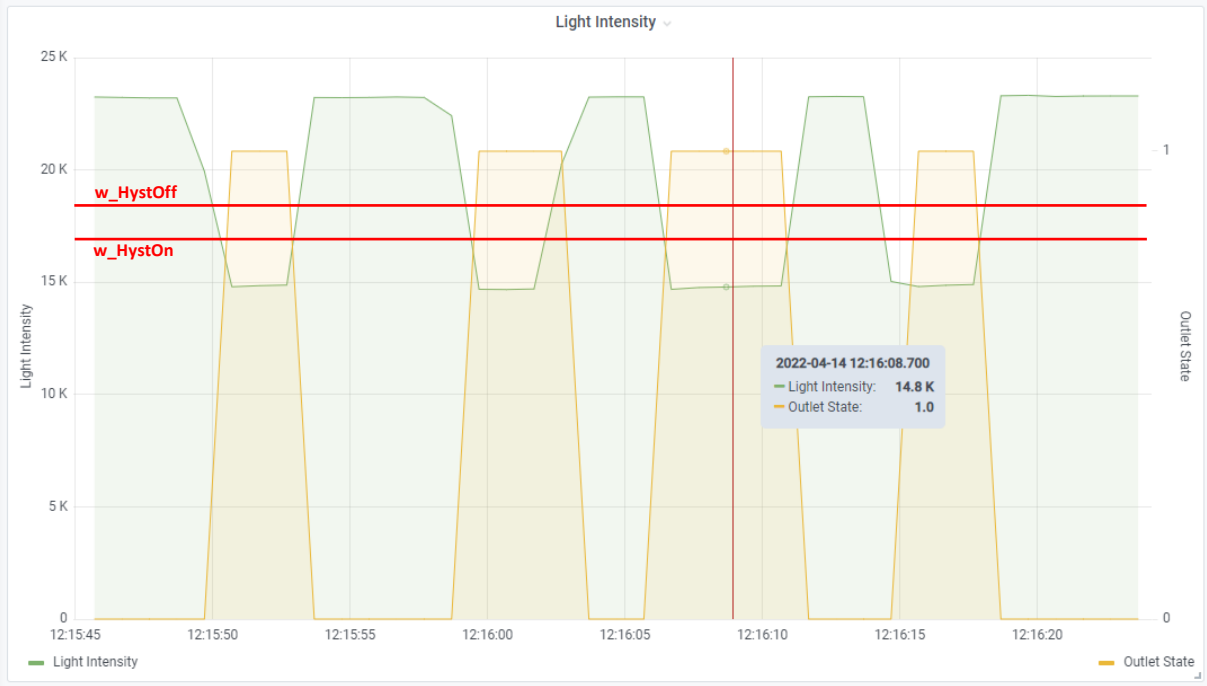



*Abbildung 11: Prototyp Beispielanwendung Schaltung Steckdose mit Hysterese*

In dieser Dokumentation wird die Version 2022.0.1 Build 5.0.546.0 von PLCnext Engineer verwendet. Um Probleme bei der Konfiguration der SPS zu vermeiden, wird im Folgenden die SPS direkt über ein Ethernet Kabel mit dem Computer verbunden. Dabei soll das Ethernet Kabel an der SPS beim Stecker X1 eingesteckt werden.

### 3.3.1. Erstellung eines Projekts

Als erstes wird ein neues Projekt in PLCnext Engineer erstellt. Dafür kann auf der Start Page, welche beim Öffnen von PLCnext Engineer erscheint (Abbildung 12, links), die Projektvorlage für unterschiedliche SPS gewählt werden. In diesem Fall wird die Vorlage «AXC F 1152 v00/2021.90» gewählt, welche in Abbildung 12 im linken Bild mit rot markiert ist. Das gesamte PLCnext Engineer Projekt, welches von diesem Punkt an erstellt wird, befindet sich in den technischen Dokumenten unter «PLC_Next_Engineer\Projects».

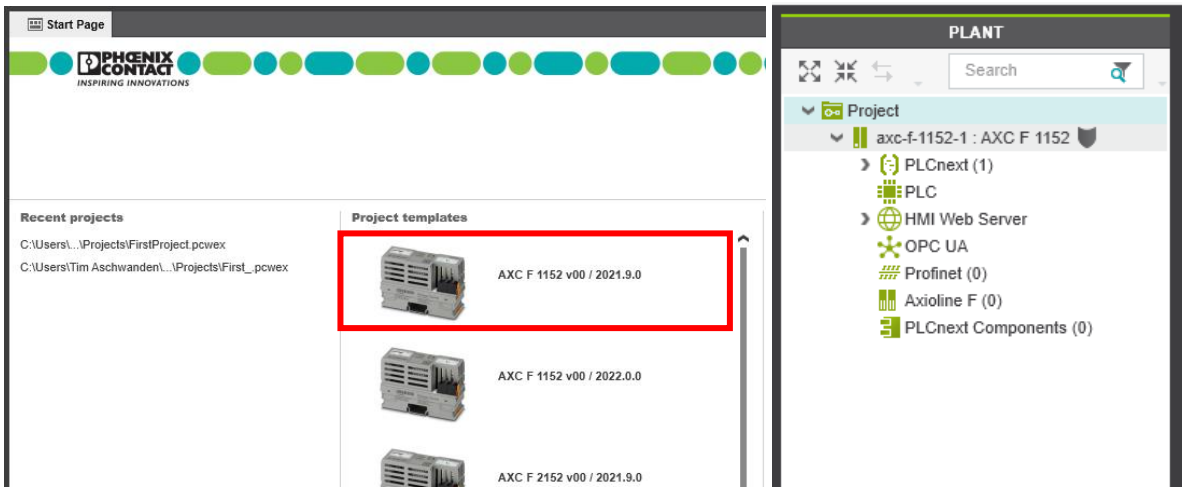



*Abbildung 12: PLCnext Engineer neues Projekt erstellen*

Nach dem Erstellen des Projekts sollte am linken Rand das PLANT-Menü erscheinen, gemäss dem rechten Bild in Abbildung 12. Falls dies der Fall ist, wurde das Projekt erfolgreich erstellt.

### 3.3.2. Verbindung zu SPS

Durch Doppelklicken des Project Ordners im PLANT-Menü erscheint eine Seite, um die Projekteinstellungen anzupassen. Auf dieser Seite kann unter dem Reiter «Online Controllers» in der linken oberen Ecke (Abbildung 13 in rot) das Network Device gewählt werden, welches benutzt wird, um nach vorhandenen Controllern (SPS) im Netz zu suchen. Durch Betätigen des Buttons rechts dieser Auswahlbox wird der Scan gestartet. In der linken Seitenhälfte (links von Status Spalte) ist das Projekt mit den wichtigsten Informationen aufgeführt. Nach dem Scan werden alle gefundenen SPS in der rechten Seitenhälfte (rechts von Status Spalte) aufgeführt. Nun kann im Feld «Select online device here» die gewünschte SPS ausgewählt werden. Daraufhin werden die lokalen Projekteinstellungen auf die SPS geschrieben und die beiden Zeilen (Project & Online) sollten sich zu einer Zeile, gemäss Abbildung 13 in der unteren Hälfte, vereinen. Um diesen Schritt zu überprüfen, kann durch Auswählen dieser vereinten Zeile und Betätigen der rechten Maustaste, die orange BF-C LED auf der SPS mittels «Flashing» zum Blinken gebracht werden.

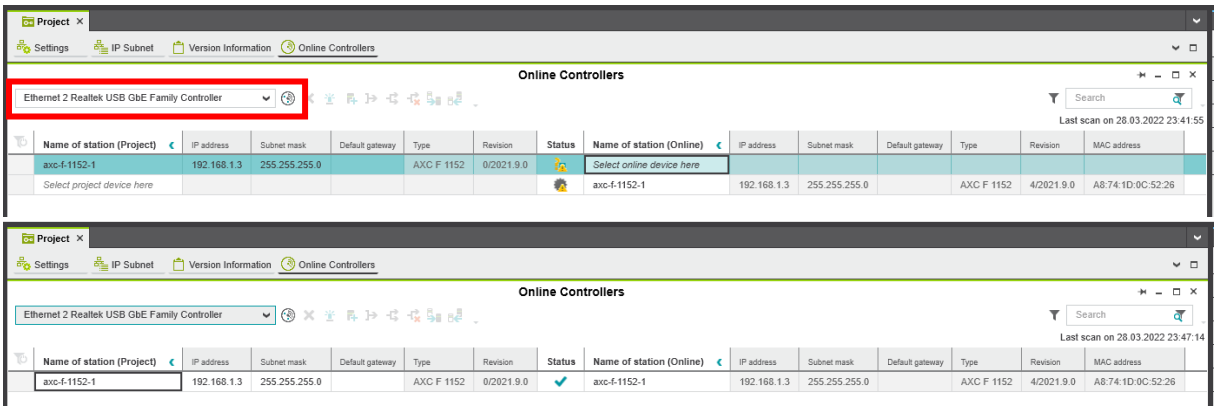



*Abbildung 13: PLC next Engineer SPS finden*

Falls die SPS beim Netzwerkscan nicht gefunden wurde, kann eine Anpassung der Scaneinstellungen unter «Settings» das Problem womöglich beheben. Abbildung 14 zeigt einen Ausschnitt dieser Einstellungen.

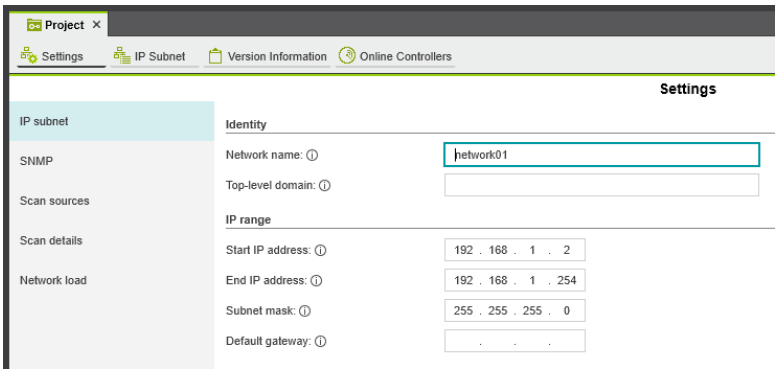



*Abbildung 14: PLCnext Engineer SPS Netzwerk Scaneinstellungen*

In einem nächsten Schritt soll eine Echtzeitverbindung zwischen dem Projekt und der SPS hergestellt werden. Dafür kann zum einen im PLANT-Menü durch Betätigen der rechten Maustaste mit «Connect/Disconnect» die Verbindung hergestellt oder getrennt werden. Zum andern kann durch Doppelklicken der SPS-Name im PLANT-Menü (axc-f-1152-1: AXC F 1152) die SPS Seite aufgerufen und unter dem Reiter «Cockpit» mit dem in Abbildung 15 rot markierten Button die Verbindung hergestellt oder getrennt werden. Beim Verbinden mit der SPS kann für das Login der Benutzername admin und das Admin-Passwort, welches auf der SPS abgedruckt ist, verwendet werden. Bei einem erfolgreichen Verbindungsaufbau erscheint im PLANT-Menü neben dem SPS-Namen ein blaues Schild und im Cockpit können die aktuellen LED Status, RAM- oder CPU-Auslastungen live eingesehen werden.

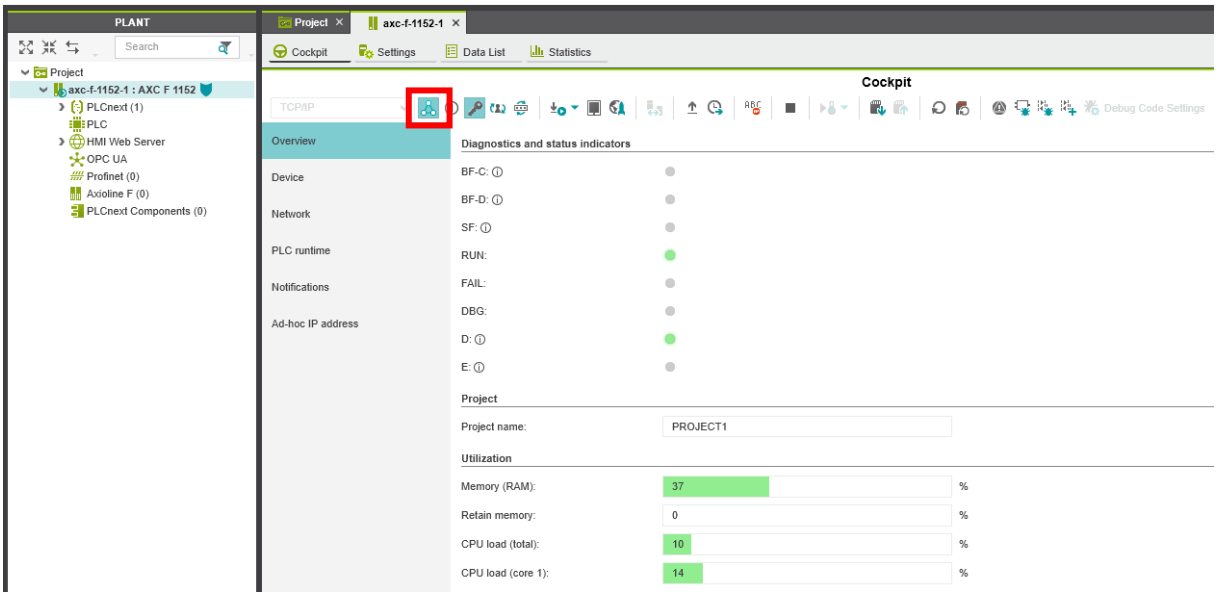



*Abbildung 15: PLCnext Engineer SPS Verbindung herstellen/trennen*

Nun ist die SPS bereit, konfiguriert und programmiert zu werden und nun können auch die Klemmen ins Projekt integriert werden. Auf der SPS Seite (Abbildung 15) im Reiter «Settings» können die wichtigsten Einstellungen und Konfigurationen der SPS, wie zum Beispiel Name, Sicherheit oder Netzwerk, vorgenommen werden. Im Reiter «Data List» werden alle globalen Variablen der SPS aufgelistet und im Reiter «Statistics» sind einige statistische Zahlen der SPS aufgeführt, wie zum Beispiel die Anzahl Programme, Variablen oder Tasks, oder die Grösse bestimmter Datentypen. In einem nächsten Schritt sollen die Ein- und Ausgangsklemmen konfiguriert werden, um die Sensoren und Aktoren ansteuern zu können.

### 3.3.3. Programmierung der Anwendung

Zur Erstellung eines Programms, welches dann auf der SPS ausgeführt wird, kann am rechten Bildschirmrand im COMPONENTS-Menü unter «Programming/Local/Programs» (Abbildung 16) das standartmässig vorhandene Main Programm geöffnet werden.

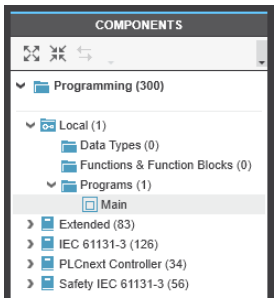



*Abbildung 16: PLCnext Main Program in COMPONENTS-Menü*

Immer, wenn ein Programm das erste Mal geöffnet wird, muss die gewünschte Programmiersprache gewählt werden. In PLCnext Engineering stehen Strukturierter Text (ST), Ladder (LD) oder Netzwerk orientierte Ladder (NOLD) als Programmiersprachen zur Verfügung. Jedoch besteht auch die Möglichkeit die PLCnext SPS mit Hochsprachen wie zum Beispiel C# oder Matlab/Simulink zu programmieren. Mehr dazu unter folgendem Link (https://www.plcnext.help/te/Programming/Introduction.htm). In diesem Projekt wird ausschliesslich der in Abbildung 17 rot markierte Strukturierte Text (ST)verwendet.

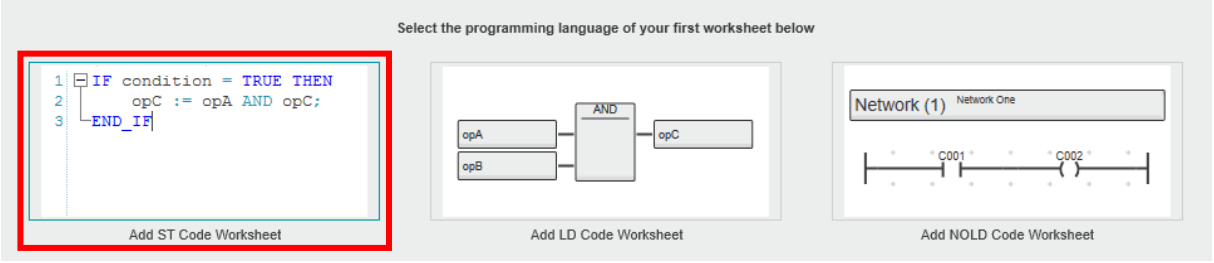
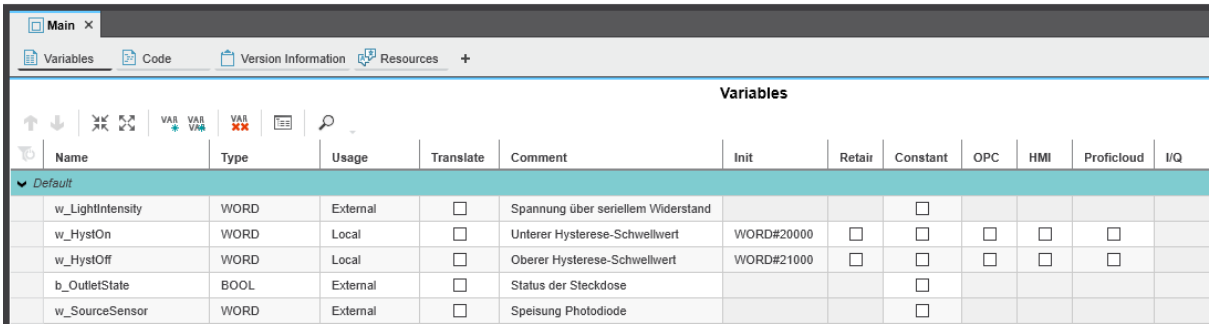



*Abbildung 17: PLCnext Engineer Wahl Programmiersprache*

Anschliessend können im Main-Fenster unter dem Reiter «Variables» die benötigten Variablen eingefügt und konfiguriert werden. Abbildung 18 zeigt die für die Beispielanwendung benötigten Variablen.




| Name | Type | Usage | Translate | Comment | Init | Retain | Constant | OPC | HMI | Proficloud | I/Q |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Default* | | | | | | | | | | | |
| w_LightIntensity | WORD | External | ☐ | Spannung über seriellem Widerstand | | | ☐ | | | | |
| w_HystOn | WORD | Local | ☐ | Unterer Hysterese-Schwellwert | WORD#20000 | ☐ | ☐ | ☐ | ☐ | ☐ | |
| w_HystOff | WORD | Local | ☐ | Oberer Hysterese-Schwellwert | WORD#21000 | ☐ | ☐ | ☐ | ☐ | ☐ | |
| b_OutletState | BOOL | External | ☐ | Status der Steckdose | | | ☐ | | | | |
| w_SourceSensor | WORD | External | ☐ | Speisung Photodiode | | | ☐ | | | | |

*Abbildung 18: PLCnext Engineer Variablen Beispielanwendung*

In der Spalte «Name» ist der Name der Variable für die Verwendung im Code, wobei der erste Buchstabe jeweils den Variablentyp signalisiert. In der Spalte «Type» ist der Variablentyp und in der Spalte «Usage» die Sichtbarkeit, zum Beispiel global, lokal oder nur in jenem Programm, definiert. Bei Variablen, die für die Klemmen benötigt werden, ist wichtig, dass der Typ WORD und die Sichtbarkeit

External gewählt werden. In der Spalte «Comment» ist die Variable beschrieben und in der Spalte «Init» kann der initiale Wert der Variable festgelegt werden. Die Variablen w_LightIntensity, w_HystOn, w_HystOff und w_SourceSensor werden keine Angaben in Volt sein, sondern in einem Wert zwischen 0 und 32'767. Wird gemäss Kapitel 3.3.5 beispielsweise eine Spannungsbereich für von 0 V bis 5 V für die analogen Ein- und Ausgänge gewählt, ist ein Wert von 32'767 eine Spannung von 5V.

Als nächster Schritt, kann im Main-Fenster unter dem Reiter «Code» folgender Code implementiert werden. Um die Photodiode und den Widerstand zu Speisen, wird auf der ersten Zeile jeweils ein analoger Ausgang auf 5 V gesetzt. Die darauffolgende IF-ELSEIF-ELSE-Schleife repräsentiert die Hysterese, wobei die Steckdose eingeschaltet wird, wenn die Lichtintensität (w_LightIntensity) kleiner als die untere Grenze der Hysterese (w_HystOn) ist und ausgeschaltet wird, wenn die Lichtintensität grösser als die obere Grenze der Hysterese (w_HystOff) ist.

```
w_SourceSensor := 32767;  // set source to 5 V

// Toggle LED with hysteresis considering voltage of resistor
IF w_LightIntensity < w_HystOn THEN
    b_OutletState := TRUE;   // enable outlet
ELSIF w_LightIntensity > w_HystOff THEN
    b_OutletState := FALSE;  // disable outlet
ELSE
    // Do nothing
END_IF
```

### 3.3.4. Festlegung Task-Zeit

Eine SPS führt ihre Aufgaben beziehungsweise die Programme stets genau zu zyklischen Zeitpunkten aus. Verglichen mit einem Linux oder Windows Betriebssystem, bei welchem Befehle erst ausgeführt werden, wenn der Prozessor Rechenzeit zur Verfügung hat, kann dadurch auf der SPS eine zuverlässige und genaue Abarbeitung der Befehle gewährleistet werden. Der in Kapitel 3.3.3 aufgeführte Main-Code wird somit standartmässig alle 100 ms ausgeführt. Dabei werden zuerst alle Ein- und Ausgänge gelesen, dann wird der Code einmal durchlaufen und zum Schluss werden die resultierenden Werte wieder an die Ein- und Ausgänge geschrieben. Durch Doppelklicken von «PLCnext» im PLANT-Menü kann unter dem Reiter «Tasks and Events» zyklische Abarbeitungszeit, auch Task-Zeit genannt, eingestellt werden. Abbildung 19 zeigt das Menü zur Anpassung der Task- und Event-Zeiten.

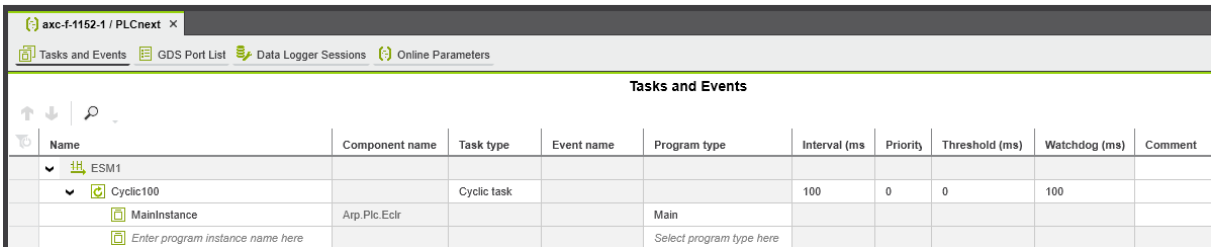



*Abbildung 19: PLCnext Engineer Festlegung Task-Zeit*

In Abbildung 19 in der Tabelle befindet sich auf der ersten Zeile der Execution and Synchronization Manager ESM1. Die SPS PLCnext AXC F 1152 hat nur einen Prozessor und deshalb auch nur einen ESM. Bei zwei Prozessoren, hätte die SPS zwei ESM. Im ESM können mehrere Tasks mit unterschiedlichen Zykluszeiten definiert werden. Das obige Beispiel (Abbildung 19) enthält einen zyklischen Task «Cyclic100» mit einer Zykluszeit (Interval) von 100 ms und der Priorität 0. Nebst dem zyklischen Task gibt es noch Event Tasks, Idle Tasks oder User Event Tasks. Mit der Priorität wird die Wichtigkeit des Tasks festgelegt. Eine Priorität von 0 hat die höchste und eine von 15 die niedrigste Wichtigkeit. Das

heisst, wenn ein Task mit der Priorität 0 und ein Task mit der Priorität 5 zur selben Zeit abgearbeitet werden sollten, wird der Task mit der Priorität 0 zuerst ausgeführt.

Den oben beschriebenen Tasks können jeweils mehrere Programme zugewiesen werden, welche dann zur Task-Zeit ausgeführt werden. Abbildung 19 zeigt beispielsweise die «MainInstance» zur Ausführung des Main-Codes. Wird ein weiteres Programm im COMPONENTS-Menü unter «Programming/Local/Programs» erstellt, kann dieses Programm auf der letzten Zeile mit «Select program type here» dem Task hinzugefügt werden. Wichtig zu beachten ist, dass ein Programm nur ausgeführt wird, wenn es einem Task angehängt ist. (PLCnext, 2022)

### 3.3.5. Konfiguration Klemmen

Damit die Ein- und Ausgangsklemmen verwendet werden können, werden bei diesem Prototyp die verwendeten Klemmen mit dem simplen, aber vielseitigen und robusten Klemmensystem Axioline F eingebunden. Durch Drücken von «Axioline F» mit der rechten Maustaste im PLANT-Menü und durch Auswählen von «Read Axioline F devices» (Abbildung 20 oben), können die verbundenen Klemmen gesucht werden. Nach einem erfolgreichen einlesen, erscheinen die Klemmen im PLANT-Menü unter «Axioline F» und auf der Axioline F Seite im «Device List» Reiter gemäss dem unteren Bild in Abbildung 20. Die Klemme «AXL F AI2 AO2 1H» ist die analoge Ein- und Ausgangsklemme und «AXL F DOR4/2 AC/220DC 1F» ist die Relais-Klemme.

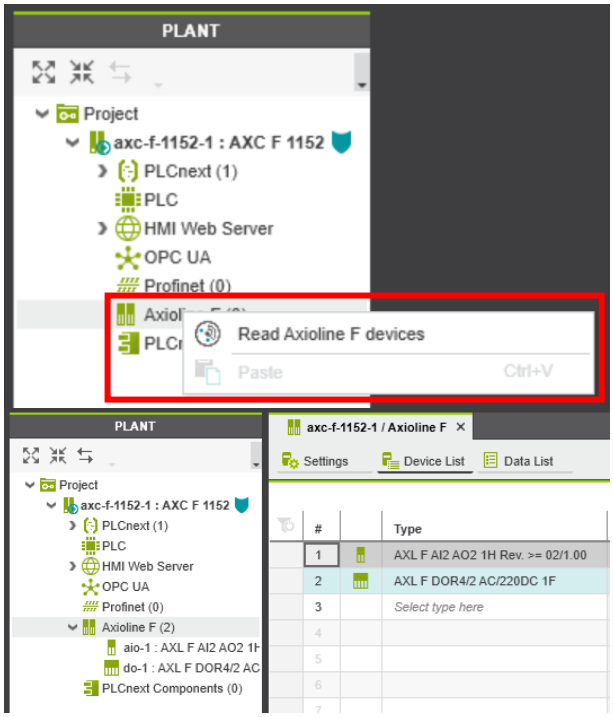



*Abbildung 20: PLCnext Engineer Axioline F Klemmen einlesen*

Durch Doppelklicken der Analogklemme (aio-1) im PLANT-Menü kann unter dem Reiter «Parameters» eingestellt werden, an welchen Ein- und Ausgängen welche Spannungen oder Ströme ausgegeben, beziehungsweise eingelesen werden. Abbildung 21 zeigt die benötigten Einstellungen für die Beispielanwendung. Der Input channel 1 wird verwendet, um eine Spannung von 0 V bis 5 V über dem Widerstand zu messen und der Output channel 2 wird verwendet, um den Widerstand und die Photodiode mit einer Spannung von 0 V bis 5 V zu speisen. Die restlichen beiden Ein- und Ausgangskanäle werden deaktiviert.

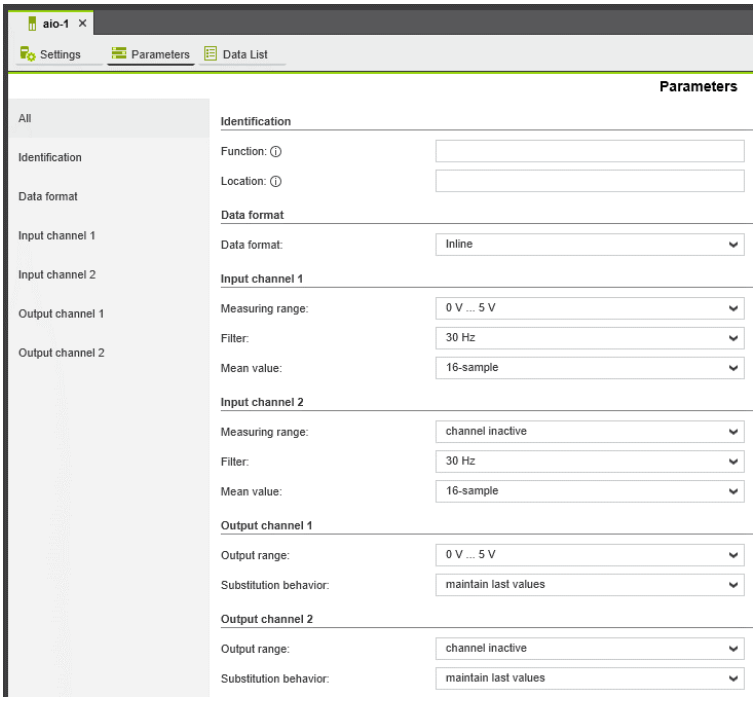



*Abbildung 21: PLCnext Engineer Analog-Klemme Konfiguration Parameter*

Anschliessend kann unter dem Reiter «Data List» gemäss Abbildung 22 die Variable w_LightIntensity als Input auf der Zeile «aio-1 / IN01» ausgewählt werden und die Variable w_SourceSensor auf der Zeile «aio-1 / OUT01» als Output. Falls die Variablen im Main Programm nicht als WORD und External deklariert wurden, können diese hier nicht ausgewählt werden.

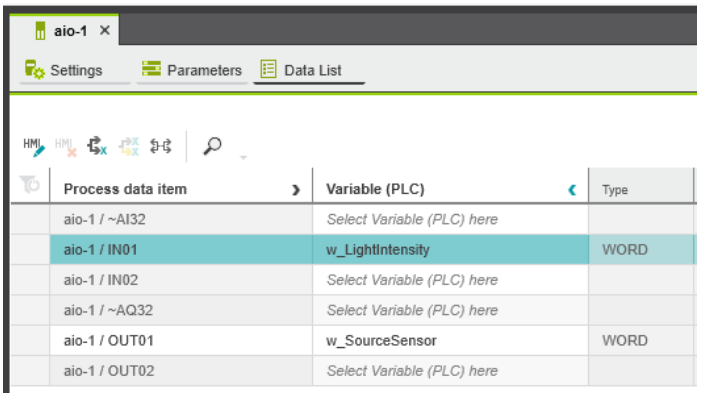



*Abbildung 22: PLCnext Engineer Analogklemme Variablen Zuweisung.*

Durch Doppelklicken der Relaisklemme (do-1) im PLANT-Menü kann unter dem Reiter «Data List» eingestellt werden, welche BOOL-Variable welchem der vier Relais zugewiesen werden soll. Dort kann die Variable «b_OutletState» gemäss Abbildung 23 dem ersten Relais der Klemme zugewiesen werden.

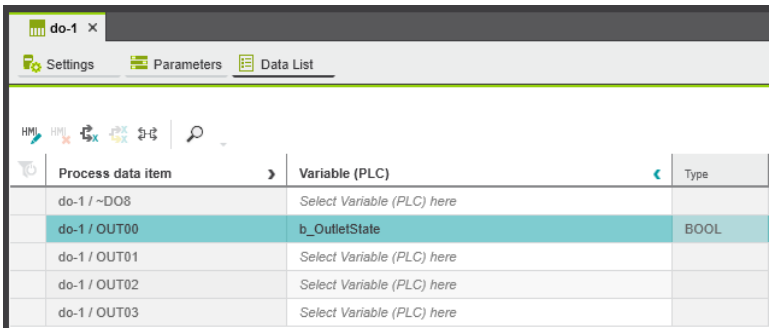



*Abbildung 23: PLCnext Engineer Relaisklemme Variablen Zuweisung*

Damit wurden die Variablen aus dem Main Programm mit den Ein- und Ausgängen der Klemmen verbunden und der Prototyp ist somit betriebsbereit. Im folgenden Kapitel wird beschrieben, wie das Programm gestartet wird und wie es gedebuggt werden kann.

### 3.3.6. Inbetriebnahme und Debugging Beispielanwendung

In einem letzten Schritt muss das zuvor beschriebene Programm noch auf die SPS geladen und darauf ausgeführt werden. Wie dies gemacht werden kann und wie durch das Programm durchgesteppt werden kann, wird in diesem Kapitel beschrieben.

Gemäss Abbildung 24 kann im PLANT-Menü durch Betätigen der rechten Maustaste auf der SPS (axc-f-1152-1) das Projekt mit «Write and Start Project» auf die SPS geladen und gestartet werden. Danach sollte nach der Eingabe der Benutzerdaten das Projekt direkt ausgeführt werden und die Steckdose einschalten, wenn die Photodiode abgedeckt ist. Falls die LED 00 auf der Relais-Klemme orange leuchtet, bedeutet dies, dass das verwendete Relais geschaltet ist und somit die Steckdose eingeschaltet ist.

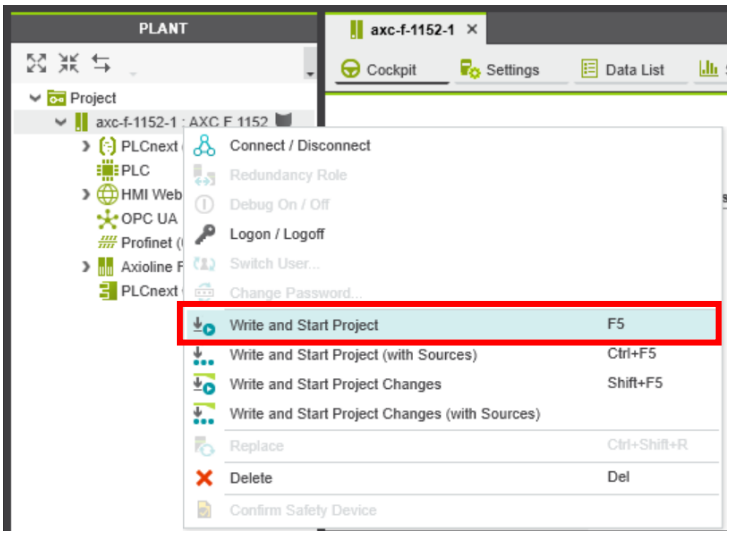



*Abbildung 24: PLCnext Engineer Ausführen des Projekts*

Ob das Projekt ausgeführt wird und ob es auch wunschgemäss funktioniert, kann in SPS-Programmierumgebungen oft mit einer Funktion gemacht werden, die es ermöglicht, sich an die Runtime der SPS anzuhängen und so die Werte der einzelnen Variablen in Echtzeit einzusehen. Dies ist auch in PLCnext Engineering möglich. Dafür muss wie in Abbildung 25 gezeigt, im PLANT-Menü unter

«PLCnext/ESM1/Cyclic100» das gewünschte Programm geöffnet werden. Dann kann im Reiter «Variables» eine Liste aller Variablen mit den aktuellen Werten dieses Programmes angezeigt werden. Im Reiter «Code» kann gemäss Abbildung 25 auch der gesamte Code mit den Werten der einzelnen Variablen eingesehen werden. Diese Echtzeit-Ansichten sind nur möglich, wenn Projekt gerade neu geladen und gestartet wurde. Wie die Werte eingesehen werden können, ohne das Projekt neu auf die SPS zu laden, wird im Folgenden erläutert.

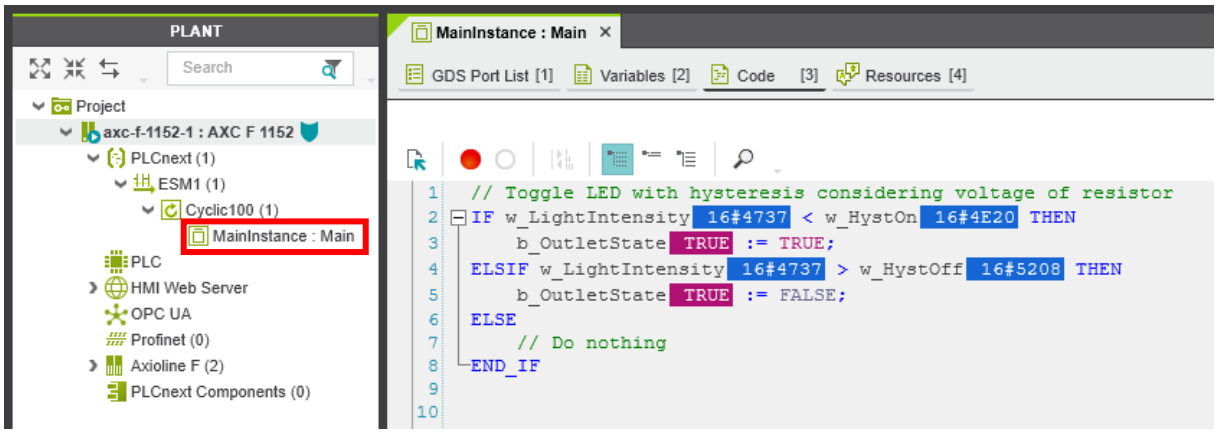



*Abbildung 25: PLCnext Engineer Realtime Einsicht Variablen*

Im PLANT-Menü in der SPS (axc-f-1152-1) kann unter dem Reiter «Cockpit» mit dem in Abbildung 26 links markierten Button die Programmierumgebung an die Runtime der SPS angehängt werden. Dafür muss die SPS verbunden sein. Dann können die Variablen gemäss Abbildung 26 live eingesehen werden. Die grünen Ecken an den Fenster-Tabs in Abbildung 25 und Abbildung 26 weisen darauf hin, dass der Computer im Moment an die Runtime angehängt ist.

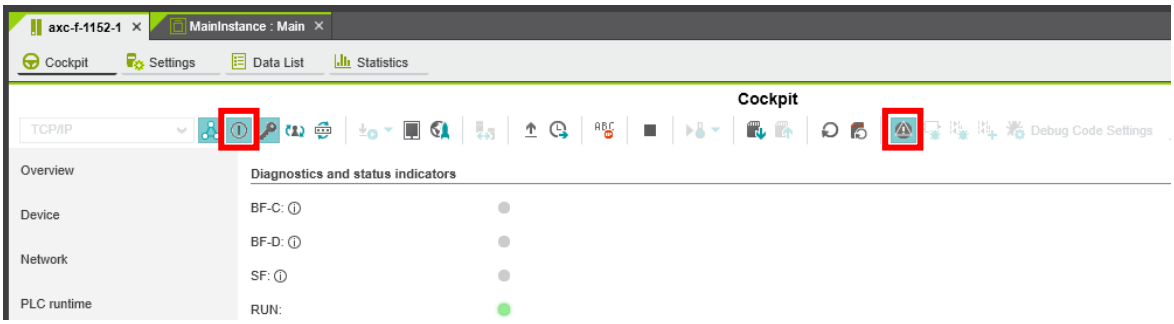



*Abbildung 26: PLCnext Engineer Anhängen an Prozess und Aktivierung Debugging*

Als nächstes wird beschrieben, wie zur Fehlerfindung und -behebung das Debugging-Tool verwendet werden kann. Dafür muss der in Abbildung 26 rechts mit dem rot markierten Button, während die Runtime angehängt ist, die Breakpoints aktiviert werden. Achtung, das Einschalten der Breakpoints kann die Ausführung des Programms unterbrechen, was zu Schäden an Maschinen oder Menschen führen könnte. Diese Meldung erscheint auch nach Betätigen des Buttons und kann mit «Yes» beantwortet werden. Anschliessend ist es möglich, im Programm, welches die Echtzeitwerte zeigt, mit der rechten Maustaste oder dem roten Punkt oberhalb des Codes (Abbildung 27), auf der gewünschten Zeile ein Breakpoint zu setzen. Abbildung 27 zeigt ein Beispiel, in welchem zwei Breakpoints gesetzt sind und sich der Debugger gerade am Zweiten befindet. Im unteren Teil der Abbildung ist das Breakpoint-Tool gezeigt. Dieses listet alle Breakpoint mit dem aktuellen Status auf. Dort kann das Programm auch fortgesetzt oder Zeile für Zeile durchgesteppt werden.

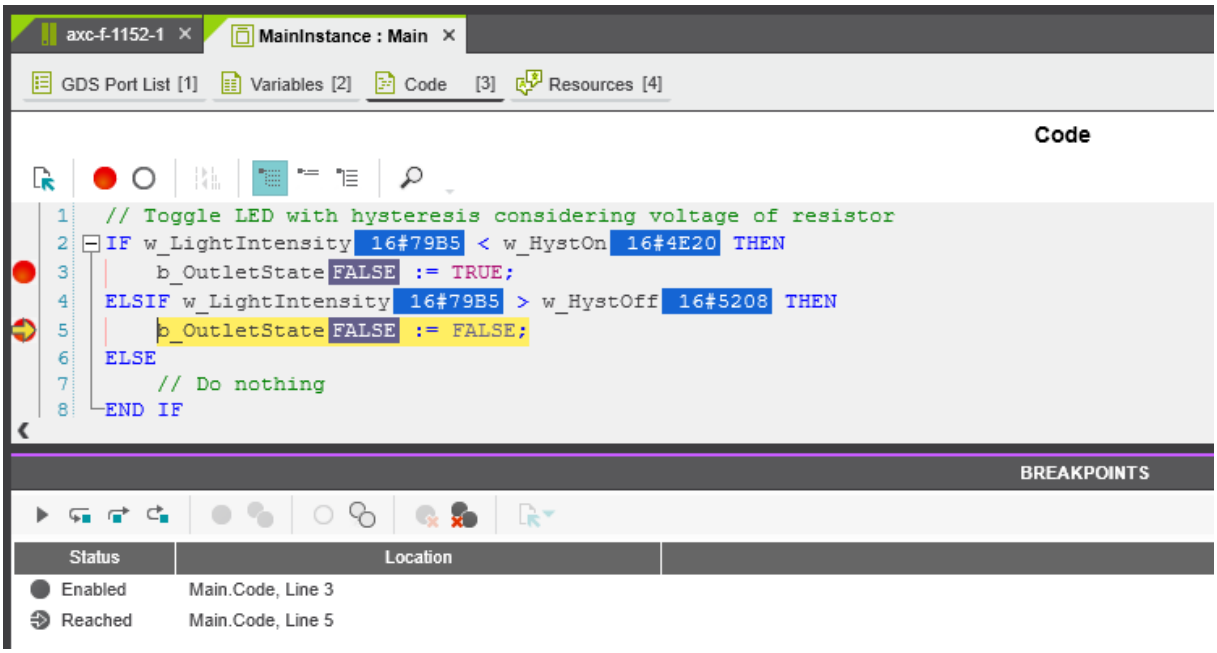



*Abbildung 27: PLCnext Engineer Breakpoint Management*

# 4. OPC UA Client Klasse und Anwendung

Dieses Kapitel enthält die Dokumentation des Aufbaus und der Anwendung der selbst erstellten OPC UA C#-Konsolenapplikation, mit welcher die Kommunikation zwischen einem Client Computer und der SPS umgesetzt werden kann. Als erstes wird dabei kurz beschrieben was OPC UA ist und wofür es verwendet werden kann. Dann wird beschrieben, welche Anpassungen in PLCnext Engineer an der SPS vorgenommen werden müssen, um den OPC UA Server zu aktivieren. Anschliessend folgt der Aufbau der Applikation mit den benötigten Einstellungen und Konfigurationen und einem Beschrieb der OPC UA Client Klasse mit potenziellen Erweiterungsmöglichkeiten. Zum Schluss wird noch die Benutzung der Applikation anhand eines Beispielprogramms erwähnt.

## 4.1. OPC UA Grundlagen

Open Platform Communications Unified Architectures (OPC UA) ist ein Kommunikationsprotokoll, welches 2008 von der OPC Foundation veröffentlicht wurde. Verglichen mit dem Vorgänger OPC Classic, welches nur auf Windows Betriebssystemen lief, ist OPC UA plattformunabhängig und ermöglicht somit die Kommunikation zwischen beliebiger Hardware und Software. Bereits heute ist das serviceorientierte, standardisierte Protokoll vor allem in der Industrie weit verbreitet und wird womöglich in Zukunft noch mehr an Wichtigkeit gewinnen. (OPC Foundation, 2022)

Aufgrund der Plattformunabhängigkeit ist OPC UA perfekt geeignet für die Vernetzung aller Geräte und Maschinen innerhalb einer Fabrik, aber durchaus auch für die Vernetzung Fabriken übergreifend. Damit könnten die Bestellung, die Planung, die Konstruktion, die Herstellung, die Logistik und die Auslieferung vernetzt und automatisiert werden. Oder wenn beispielsweise ein automatisiertes Lager feststellt, dass es keine Schrauben mehr hat, könnte es direkt über OPC UA eine Schraubenbestellung an den Schraubenlieferanten senden. Deshalb ist OPC UA in der Industrie 4.0, im IIOT und in der automatisierten Industrie ein geläufiger Begriff.

OPC UA ist ein Protokoll mit vielen unterschiedlichen Möglichkeiten, wie zum Beispiel das Schreiben und Lesen von Daten, das Server- und Clientseitige Überwachen von Werten, wobei Notifikationen gesendet werden können, falls der Wert eine bestimmte Abweichung erfährt. Zudem gibt es auch diverse Möglichkeiten Authentifikation, Zertifizierung und Verschlüsselung zu gewährleisten. Vom Client können auch vordefinierte Programme auf dem Server ausgeführt werden. Ausserdem bietet OPC UA neuerdings eine Kommunikation mit dem Publisher-Subscriber-Verfahren. Diese sogenannte PubSub Variante ist grundsätzlich sehr ähnlich wie MQTT. In dieser Arbeit wird jedoch lediglich auf das Schreiben, Lesen und auf die Clientseitige Wertüberwachung «monitored Items» eingegangen. (OPC Foundation, 2022)

Ein weiterer sehr wichtiger Aspekt von OPC UA ist der Aufbau und die Anpassbarkeit des Informationsmodels. Ein Informationsmodel beschreibt, wie Informationen zwischen Sensoren, Aktoren, SPS, Computern, Maschinen oder Fabriken ausgetauscht werden können bzw. müssen. Mit einem solchen Model kann die Vernetzung aller Geräte deutlich vereinfacht werden. Aufgrund der Objekt-Orientiertheit ist es möglich komplexe Modelstrukturen auf dem Server zu implementieren, welche gesamte Fabriken inklusive aller Geräte beinhaltet. Bei dieser Projektarbeit liegt der Fokus jedoch nur auf einem einfachen Informationsmodel mit einer SPS und einigen Variablen für die Sensor- und Aktor-Werte. (Industry40tv, Youtube, 2022)

In OPC UA sind alle Daten und Objekte in eindeutig identifizierbare Knoten aufgeteilt. Diese Knoten können zum Beispiel Variablen, Programme oder Methoden darstellen in denen Identifier, Daten, und Referenzen abgespeichert sind. Die Referenzen zeigen auf alle verbunden Knoten im Model und bilden somit eine nach Belieben gestaltbare Mesh Struktur. Abbildung 28 zeigt ein Beispiel Mesh, wie es auf

einem OPC UA Server implementiert werden könnte. Dabei könnte der Knoten 1 beispielsweise als SPS fungieren, welche Referenzen auf die Knoten 2, 3, 5 und 6 hat. Der Knoten 6 könnte dann die Variable Lichtintensität sein, welche die Werte der Lichtintensität speichert und Referenzen auf die Knoten 1 und 5 hat. Alle Knoten wissen welche Nachbarknoten sie besitzen, beziehungsweise auf welche Knoten sie referenzieren. Mehr zu Knoten, Identifier und dem in diesem Projekt verwendeten Informationsmodel folgt in den Kapiteln 4.3.2 bis 4.3.4. (Industry40tv, Youtube, 2022)

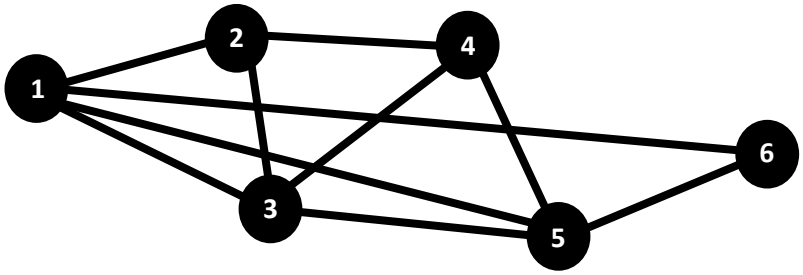



*Abbildung 28: Beispiel Mesh Informationsmodel OPC UA*

## 4.2. OPC UA Server Setup

Damit die verwendete PLCnext SPS als OPC UA Server zur Verfügung steht, müssen zuerst einige Einstellungen vorgenommen werden, welche in diesem Kapitel dokumentiert werden. In diesem Projekt wird eine SPS benutzt, welche auch gleichzeitig der OPC UA Server ist. Bei einer Firma oder einem grösseren Projekt hätte es beispielsweise einen Server und mehrere verteilte SPS welche jeweils als OPC UA Clients fungieren. Das komplette PLCnext Engineer Projekt befindet sich in den technischen Dokumenten unter «PLC_Next\PLC_Next_Engineer\Projects»

Als erste Einstellung muss im PLCnext Engineer Projekt am linken Rand im PLANT-Menü unter OPC UA unter «Basic settings» die «Visibility of variables» gemäss Abbildung 29 auf «Marked» gesetzt werden. Damit werden nur die markierten Variablen auf dem Server sichtbar sein.

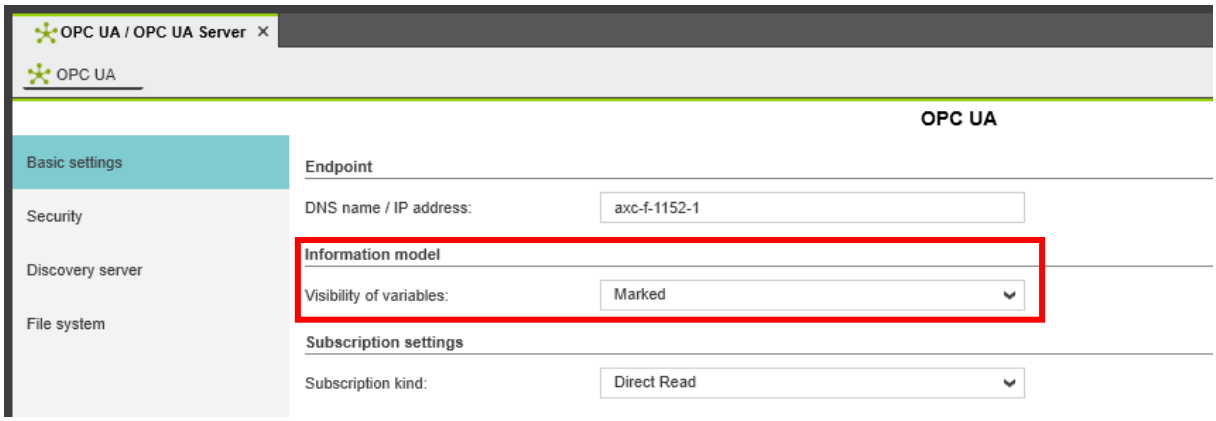



*Abbildung 29: PLCnext Engineer OPC UA Server Basic settings*

Zur Erstellung der vereinfachten OPC UA C#-Konsolenanwendung wird gemäss Abbildung 30 die Überprüfung des Zertifikats ausgeschaltet, damit die Zertifizierung vereinfacht beziehungsweise ganz umgangen werden kann. Denn in Kapitel 4.4.1.1 wird beim Verbindungsaufbau in der Konsolenanwendung angegeben, dass auf eine Zertifizierung verzichtet werden soll.

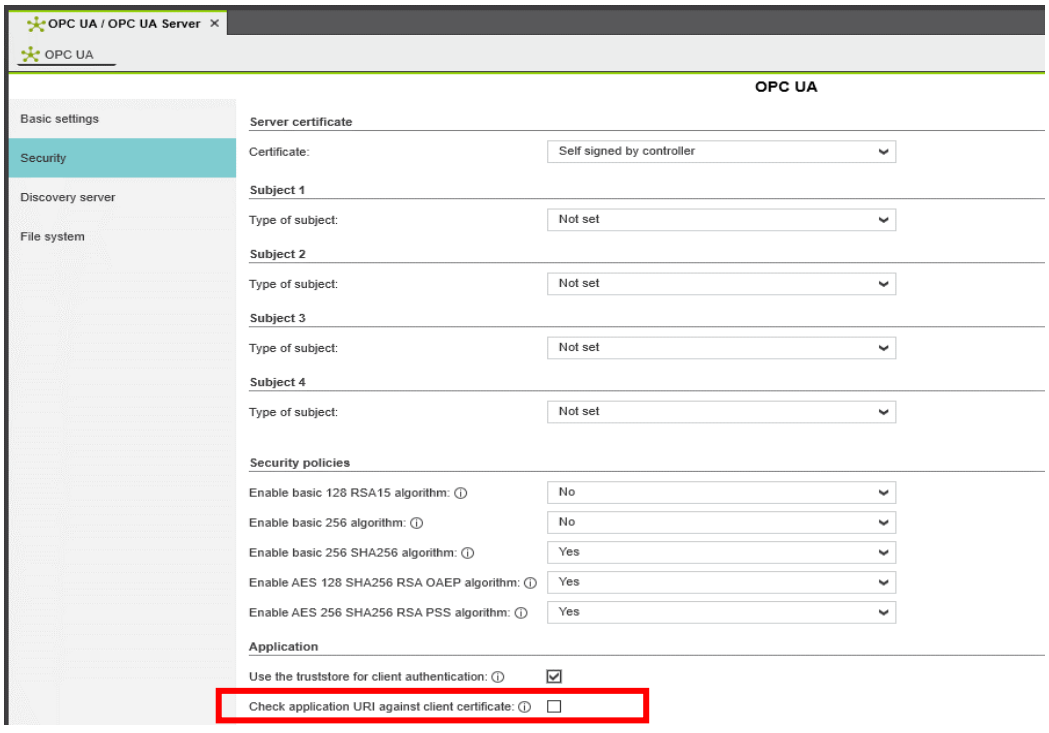



*Abbildung 30: PLCnext Engineer OPC UA Server Security*

Wie bereits erwähnt, müssen die Variablen, welche auf dem OPC UA Server sichtbar sein sollen, markiert werden. Für externe respektive globale Variablen kann dies im PLANT-Menü unter dem SPS-Namen im Reiter «Data List» erledigt werden. Dort müssen gemäss Abbildung 31 in der Spalte «OPC» bei den gewünschten Variablen die Häkchen gesetzt werden. In Falle dieses Projekts sind es die Variablen w_LightIntensity, w_SourceSensor und b_OutletState.

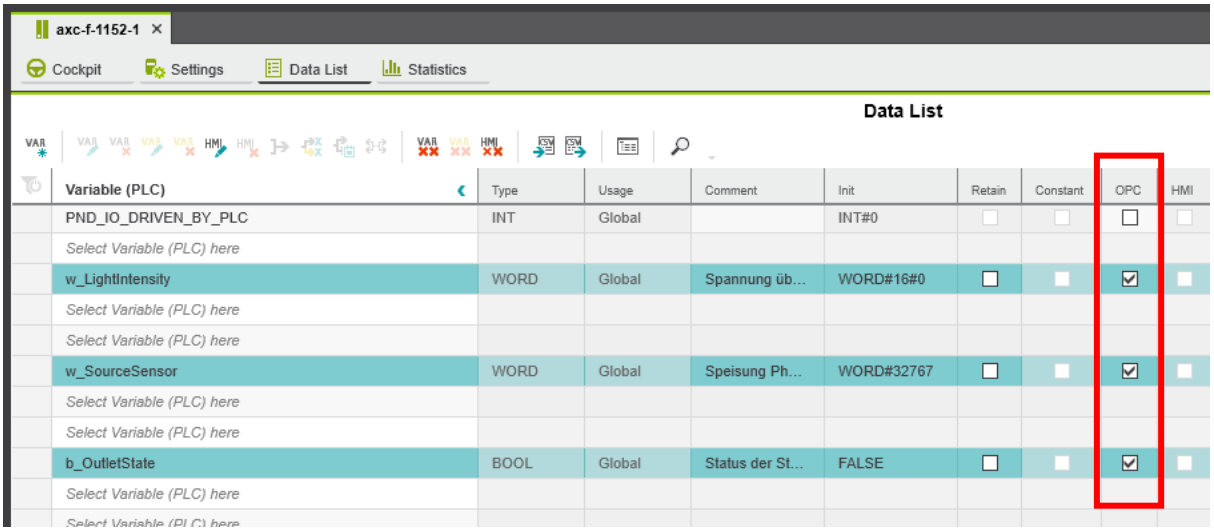



*Abbildung 31: PLCnext Engineer OPC UA Markierung externer/globaler Variablen*

Für die beiden lokalen Variablen w_HystOn und w_HystOff muss im COMPONENTS-Menü am rechten Bildschirmrand unter «Programming/Local/Programs/Main» im Reiter «Variables» gemäss Abbildung 32 in der OPC-Spalte bei jenen Variablen das Häkchen gesetzt werden. (PLCnext, 2022)

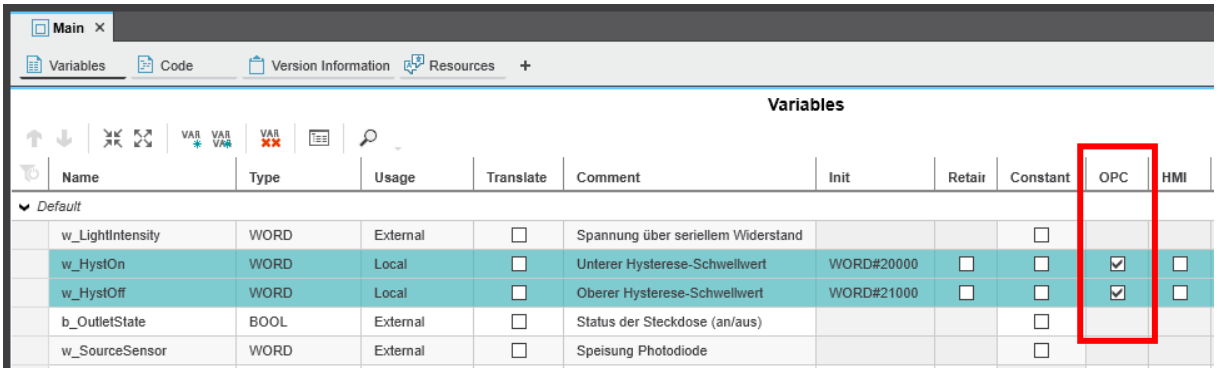



*Abbildung 32: PLCnext Engineer OPC UA Markierung lokaler Variablen*

Der Grund warum die globalen respektive externen Variablen w_LightIntensity, w_SourceSensor und b_OutletState nicht im Main-Programm markiert werden können ist, dass diese Variablen auf den Klemmen benötigt werden und somit als extern markiert werden mussten im Main-Programm. Alle externen Variablen werden anschliessend automatisch in die «Data List» der SPS (Abbildung 31) als globale Variablen hinzugefügt.

Nach dem Schreiben des Projekts auf die SPS treten die Änderungen in Kraft und der OPC UA Server mit den ausgewählten Variablen sollte laufen. Ob es auch wirklich funktioniert hat, kann wie im folgenden Kapitel (4.3) beschrieben, überprüft werden.

## 4.3. OPC UA Server UA Expert

Zur Überprüfung, ob das Setup des OPC UA Servers aus Kapitel 4.2 funktioniert hat, soll in diesem Abschnitt mit dem OPC UA Client UaExpert von der Firma Unified Automation eine erste Verbindung zwischen dem Client und dem Server hergestellt und dann die Daten ausgelesen werden. UaExpert ist ein mit allen Funktionen ausgestatteter OPC UA Client, mit welchem unter anderem Daten gelesen und geschrieben werden können. Hierbei wurde die UaExpert Version 1.6.1 424 verwendet. Des weiteren soll der Aufbau eines Knoten mit den Referenzen zu anderen Knoten in diesem Kapitel verdeutlicht werden. (Unified Automation, 2022)

### 4.3.1. OPC UA Server UA Expert Verbindungsaufbau

In einem ersten Schritt kann unter dem Reiter «Server» unter «Add…» das «Add Server» Fenster, welches in Abbildung 33 links zu sehen ist, geöffnet werden. Dort kann unter «Custom Discovery» durch Doppelklicken der Server mit dem URL des Servers gemäss dem rechten Fenster in Abbildung 33 hinzugefügt werden. Der URL (Uniform Resource Locator) setzt sich zusammen aus «opc.tcp://», der IP des Servers und dem OPC UA Port 4840. Für dieses Projekt ergibt sich dadurch den URL opc.tcp://192.168.1.3:4840.

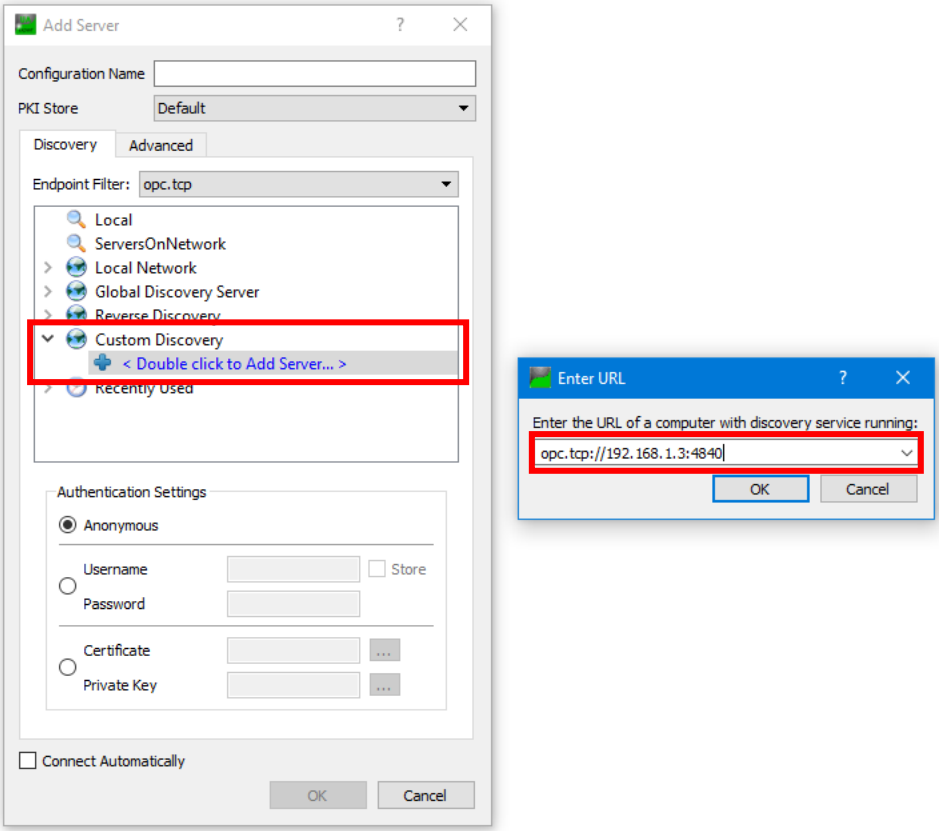



*Abbildung 33: UaExpert Add Server*

Daraufhin sollten, wie in Abbildung 34 gezeigt, im «Add Server» Fenster die unterschiedlichen Verschlüsselungsmethoden angezeigt werden können. Die dort sichtbaren Methoden, sind jene, die auch in Kapitel 4.2 (OPC UA Server Setup) in Abbildung 30 unter «Security policies» ausgewählt wurden. Es wird empfohlen, die Methode «Basic256Sha256 – Sign & Encrypt» zu verwenden. Dabei werden die Daten signiert und mit dem SHA-256 verschlüsselt. Somit wird sichergestellt, dass die Daten nicht eingesehen oder verändert werden können und dass sie vom gewünschten Server/Client kommen.

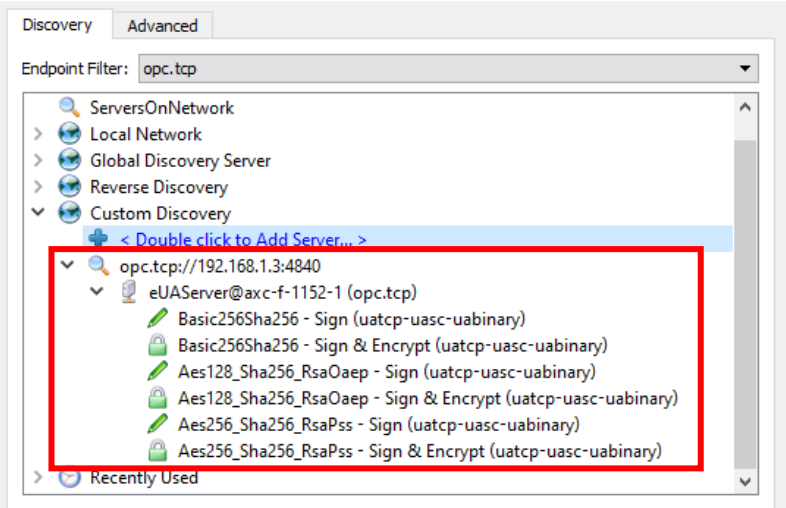



*Abbildung 34: UaExpert Verschlüsselungsarten*

Nach Doppelklicken der gewünschten Verschlüsselungsart erscheint der Server, also die SPS, im Projektbaum am linken Fensterrand unter «Servers» (Abbildung 35). Falls bim Hinzufügen die «Replaced Hostname» Meldung erscheint, kann diese bejaht werden. Zum Schluss kann dann durch Auswählen des Servers mit der rechten Maustaste, mit «Connect» die Verbindung zum Server hergestellt werden. Beim erfolgreichen Herstellen einer Verbindung wird dies im «Log» Fenster ausgegeben.

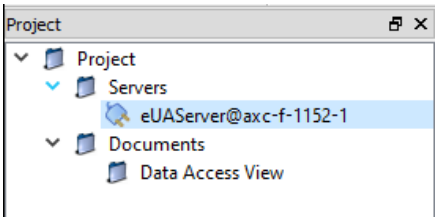



*Abbildung 35: UaExpert Projektbaum mit Server*

### 4.3.2. OPC UA Server UA Expert Daten Lesen & Schreiben

Um Werte zu lesen kann im «Address Space» Fenster am linken Rand die gesamte Struktur der SPS eingesehen werden. Wie in Abbildung 36 gezeigt, befinden sich unter «Root/Objects/ PLCnext/Arp.Plc.Eclr» die eigens definierten globalen, respektive externen Variablen und unter «MainInstance» die Lokalen. Somit kann bestätigt werden, dass das Aufsetzen des Servers erfolgreich war.

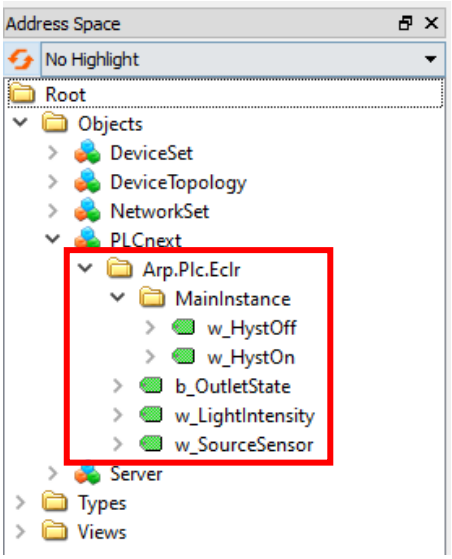



*Abbildung 36: UaExpert Address Space OPC UA Server UA Expert Verbindungsaufbau*

Durch Ziehen einer Variable aus dem «Address Space» Fenster ins Hauptfenster, dem «Data Access View» Fenster, kann wie in Abbildung 37 gezeigt, eine Variable in Echtzeit eingesehen werden. Zudem kann durch Doppelklicken eines Wertes in der Spalte «Value» ein Wert in eine Variable geschrieben werden.

Data Access View

| # | Server | Node Id | Display Name | Value | Datatype | Source Timestamp | Server Timestamp | Statuscode |
|---|---|---|---|---|---|---|---|---|
| 1 | eUAServer@axc-f-1152-1 | NS5\|String\|Arp.Plc.Eclr/w_LightIntensity | w_LightIntensity | 14070 | UInt16 | 21:12:40.617 | 21:12:40.618 | Good |
| 2 | eUAServer@axc-f-1152-1 | NS5\|String\|Arp.Plc.Eclr/w_SourceSensor | w_SourceSensor | 32767 | UInt16 | 21:11:48.195 | 21:11:48.195 | Good |
| 3 | eUAServer@axc-f-1152-1 | NS5\|String\|Arp.Plc.Eclr/b_OutletState | b_OutletState | true | Boolean | 21:11:49.192 | 21:11:49.192 | Good |
| 4 | eUAServer@axc-f-1152-1 | NS5\|String\|Arp.Plc.Eclr/MainInstance.w_HystOn | w_HystOn | 20000 | UInt16 | 21:11:50.360 | 21:11:50.360 | Good |
| 5 | eUAServer@axc-f-1152-1 | NS5\|String\|Arp.Plc.Eclr/MainInstance.w_HystOff | w_HystOff | 21000 | UInt16 | 21:11:52.026 | 21:11:52.026 | Good |

*Abbildung 37: UaExpert Data Access View*

Die in Abbildung 37 aufgelisteten Objekte werden zyklisch aktualisiert. Dies geschieht mit sogenannten Monitored Items, welche eine Möglichkeit sind, vom Client aus Werte auf dem Server zu überwachen und informiert zu werden, falls diese Werte eine gewisse Veränderung erfahren. Mehr Informationen zu Monitored Items und ihrer Implementation in C# befindet sich im Kapitel 4.4.1.5.

### 4.3.3. OPC UA Server UA Expert Knoten Attribute

Wie bereits in Kapitel 4.1 erwähnt, werden in OPC UA alle Variablen, Methoden, Programme und Maschinen anhand von Knoten (nodes) repräsentiert. Diese Knoten enthalten unterschiedlichste Informationen wie NodeId, NodeClass, DisplayName, Value oder DataType, welche Attribute genannt werden. Mit UaExpert besteht die Möglichkeit durch Auswählen eines Elements im «Address Space» Fenster, alle Attribute dieses Knotens, gemäss Abbildung 38, im «Attributes» Fenster anzuzeigen.

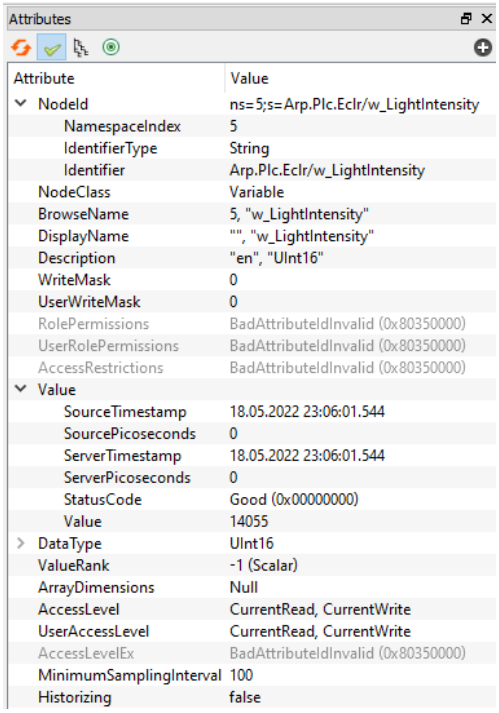



*Abbildung 38: UaExpert w_LightIntensity Attribute*

Von zentraler Bedeutung in diesem Projekt ist die einmalige NodeId, denn diese wird im Kapitel 4.4 oft verwendet, um Knoten zu erstellen, beziehungweise nach ihnen zu suchen. Eine NodeId setzt sich immer aus einem Namespace Index, einem Identifier Typ und einem Identifier zusammen. Auf den folgenden Zeilen sind die NodeId der Variablen «w_LightIntensity» und «w_HystOff»gezeigt. Dabei ist der Namespace Index ns fünf (gelb), der Identifier Typ string (grün) und der Identifier «Arp.Plc.Eclr/w_LightIntensity» beziehungsweise «Arp.Plc.Eclr/MainInstance.w_HystOff» (blau). Die Variable «w_HystOff» ist eine lokale Variable der «MainInstance», weshalb dies auch so im Identifier vorkommt. Der grün markierte Buchstabe s weisst auf den Identifier Typ string hin. Nebst string gibt es beispielsweise noch GUID (global unique identifier) oder ByteString als Identifier Typen. (Singh, 2022)

ns=5;s=Arp.Plc.Eclr/w_LightIntensity

ns=5;s=Arp.Plc.Eclr/MainInstance.w_HystOff

Der Namespace repräsentiert einen Abschnitt bzw. einen Raum, in welchem die jeweiligen Knoten nur einmal vorkommen und auf welche nicht von anderen Namespaces zugegriffen werden kann. Sie können vom Prinzip her gut mit Namespaces in C# verglichen werden. Jeder der Namespaces, welcher auf dem Server vorhanden ist, wird mit dem einzigartigen Namespace Index unterschieden. Durch Anzeigen des Werts der Variable «Root/Objects/Server/NamespaceArray» im UaExpert, können alle vorhandenen Namespaces eingesehen werden. Die Namespaces null bis zwei sind standardmässig für die OPC Foundation reserviert. In diesem Projekt ist der Namespace mit dem Index fünf der «GlobalDataSpace» von Phoenix Contact.

### 4.3.4. OPC UA Server UA Expert Knoten Referenzen

In OPC UA besteht für jeden Knoten die Möglichkeit auf einen anderen Knoten zu referenzieren oder von einem anderen Knoten referenziert zu werden. Dieser Aspekt wurde auch schon im Kapitel 4.1 angesprochen und ermöglicht es, beliebige Strukturen, zur Organisation gesamter Firmen, mit unterschiedlichen Knoten, herzustellen. Mit UaExpert können durch Auswählen des Knotens im «Address Space» Fenster die jeweiligen Referenzen im «References» Fenster angezeigt werden. Abbildung 39 zeigt die Referenzen des Object Knotens «MainInstance». Dabei wurde die «Browse Direction» in der roten Box auf «Both» gesetzt, dass die Referenzen auf «MainInstance», in diesem Fall «Arp.Plc.Eclr», und jene von «MainInstance», in diesem Fall «w_HystOn» und «w_HystOff» angezeigt werden. Zudem kann der Knotentyp an der Referenz «HasTypeDefinition» abgelesen werden.

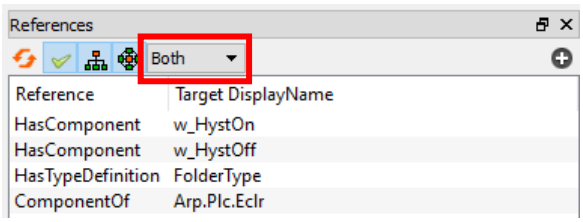



*Abbildung 39: UaExpert MainInstance Referenzen*

Abbildung 40 zeigt eine detaillierte Darstellung der wichtigsten Knoten dieses Projekts mit den jeweiligen Referenzen auf folgende Knoten (HasComponent, in schwarz) und auf vorherige Knoten (ComponentOf, in blau), signalisiert durch Pfeile auf oder von weiteren Quadraten.

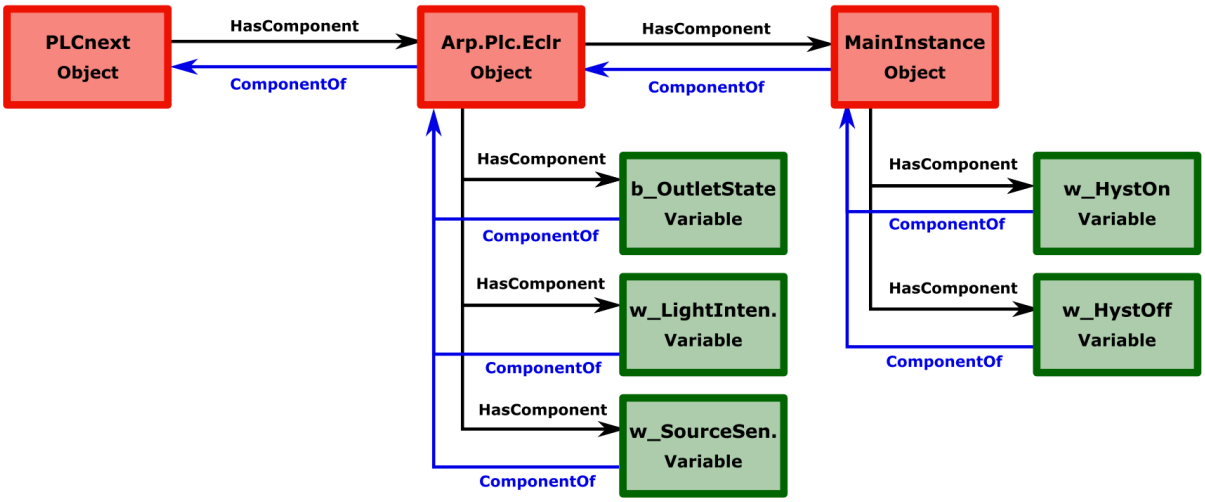



*Abbildung 40: OPC UA Server Project Mesh*

## 4.4. OPC UA Konsolenanwendung

In diesem Abschnitt wird der Aufbau, die Implementation und die Benutzung der OPC UA Client C#-Konsolenanwendung dokumentiert. Dabei liegt der Fokus auf der eigenen Klasse, mit welcher ein Client erstellt, die Verbindung zum Server hergestellt und die in Kapitel 4.1 definierten Punkte (Lesen, Schreiben & Monitored Items) umgesetzt werden können.

Zur Entwicklung der Anwendung wurde Microsoft Visual Studio Enterprise 2019 Version 16.8.3 verwendet. Nach dem Erstellen der Konsolenanwendung-Projekts (Console App) muss sichergestellt werden, dass das File «Opc.Ua.Client.Config.xml» dem Projekt hinzugefügt werden. Dieses File enthält Informationen bezüglich des Clients und der Client-Server-Kommunikation wie zum Beispiel Namen, Sicherheitskonfigurationen oder Paketgrössen. Es muss dringend vorhanden sein, ansonsten kann keine Verbindung aufgebaut werden (Kapitel 4.4.1.1). Es befindet sich bei den technischen Dokumenten im Ordner «_UA\Projects\BachelorThesis_OPC_UA» und im Anhang 9.1. Anschliessend muss das NuGet Package «OPCFoundation.NetStandart.Opc.Ua» der OPC Foundation dem Projekt hinzugefügt werden, damit die grundlegenden OPC UA Funktionen verwendet werden können. Bei dieser Arbeit wurde die Version 1.4.368.33 benutzt. Zur Erstellung der OPC UA Client Klasse wurde vom Projekt unter folgendem Link ausgegangen:

https://github.com/OPCFoundation/UA-.NETStandard-Samples/tree/master/Samples/NetCore ConsoleClient

In den technischen Dokumenten im Ordner «OPC_UA\Projects» befindet sich das Projekt mit der eigens erstellten OPC UA Client Klasse. Zudem befinden sich in diesem Projekt auch ein Programm als Beispiel zur Verwendung der OPC UA Client Klasse. Dieses Programm entspricht dem in Abbildung 41 gezeigten Ablauf. Die detailliertere Funktionsweise, der Aufbau und die Implementation der Klasse werden in den folgenden Kapiteln beschrieben.

Abbildung 41 zeigt ein Beispiel, wie die Klasse angewendet werden könnte. Dabei wird mit der Connect-Methode als erstes die Verbindung zwischen dem Client und dem Server hergestellt. Diese Verbindung, auch Session genannt, bleibt stets erhalten und alle Befehle und Informationen werden jeweils über diese gesicherte Verbindung übermittelt. Um sich einen Überblick zu verschaffen, welche Variablen auf dem Server zur Verfügung stehen, werden mit der Methode «Show_All_Identifiers()» alle vorhandenen Variablen, beziehungsweise Identifier, angezeigt. Dann wird der Wert der Variable «w_LightIntensity» mit der Methode «Get_Value(“w_LightIntensity”)» gelesen und der Wert 21000 mit der Methode «Set_Value(“w_HystOff”, 21000)» in die Variable «w_HystOff» geschrieben. Zuletzt wird auf dem Server mit der Methode «Create_Monitored_Item(…)» ein Monitored Item erstellt, welches die Variable «w_LightIntensity» überwacht und bei einer bestimmten Veränderung des Werts eine Notifikation an den Client sendet. Das Eintreten einer solchen Veränderung ist in Abbildung 41 mit dem Blitz gekennzeichnet. Zu diesem Zeitpunkt fehlt noch eine Methode, um die Verbindung, also die Session, zwischen dem OPC UA Client (Computer) und dem OPC UA Server (SPS) zu trennen. Mit der Methode CloseSession(…) der Klasse Session wäre dies womöglich ohne grossen Aufwand umzusetzen.

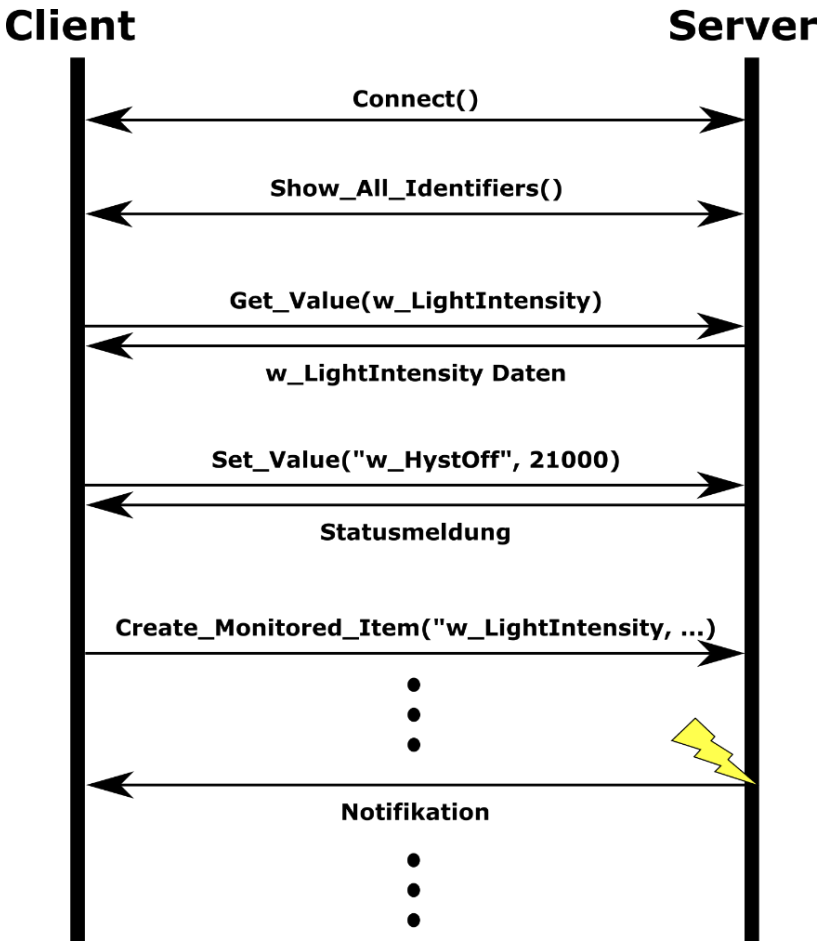



*Abbildung 41: OPC UA Client Klasse Beispiel*

### 4.4.1. OPC_UA_Client Klasse

Die eigens erstellte C# OPC UA Client Klasse beinhaltet die wichtigsten Methoden, um mit einem OPC UA Server zu kommunizieren. In diesem und den folgenden Abschnitten der Klassenbeschreibung wird nur auf die wesentlichen Punkte eingegangen. Jedes der folgenden Kapitel enthält jeweils die Beschreibung einer Methode, welche zur Beschreibung in Teil-Codeabschnitte aufgeteilt sind. Der gesamte Code, welcher auch Konsolenausgaben beinhaltet, befindet sich bei den technischen Dokumenten unter «OPC_UA\Projects\BachelorThesis_OPC_UA». Der folgende Codeabschnitt zeigt die Membervariablen und den Konstruktor. Die Variablen «userName» und «password» sind der Benutzername und das Passwort um sich mit dem Server, also der SPS, zu verbinden. Das Passwort zum Benutzernamen «admin» steht jeweils auf der SPS. Beim Verbindungsaufbau (Connect()) wird dann aus der Variable «serverIP», welche beim Erstellen eines OPC_UA_Client Objekts übergeben wird, der gesamte Server URL zusammengesetzt und in die Variable «serverURL» gespeichert. Dieser Server URL könnte wie folgt aussehen: «opc.tcp://192.168.1.3:4840».

```
private string serverURL; // opc.tcp://192.168.1.3:4840
public string serverIP;
private bool autoAccept; // true = disable certification
public string userName;
private string password;
Session session; // Session for the client-server communication

public OPC_UA_Client(string serverIP, bool autoAccept,
   string userName, string password)  // Constructor
{
    this.serverIP = serverIP;
    this.autoAccept = autoAccept;
    this.userName = userName;
    this.password = password;
}
```

Damit beim Verbindungsaufbau in der Connect() Methode die Zertifizierung umgangen, beziehungsweise ausgeschaltet werden kann, besitzt die Klasse die Membervariable «autoAccept». Falls diese true ist, wird eine Verbindung zwischen dem Client und dem Server hergestellt, auch bei nicht Vorhandensein eines vertrauenswürdigen Zertifikats. Das Session Objekt «session» beinhaltet nach dem Verbindungsaufbau den eigentlichen sicheren und verschlüsselten Kommunikationskanal zwischen dem Client und dem Server. Dieser Kanal bleibt stets erhalten und für jegliche Kommunikationen wird dieser, respektive das Session Objekt verwendet. Mit der folgenden Zeile Code wird gezeigt, wie ein OPC_UA_Client Objekt mit dem Namen «client» erstellt werden könnte. (Aro, 2022)

```
OPC_UA_Client client = new OPC_UA_Client("192.168.1.3", true, "admin", "1234");
```

#### 4.4.1.1. OPC UA Client Connect()

Dieser Abschnitt beinhaltet die Beschreibung der Connect()-Methode, mit welcher eine Verbindung, eine sogenannte Session, zwischen dem Client und dem gewünschten Server hergestellt wird. Dabei resultiert ein Session-Objekt, welches in die Membervariable «session» der Klasse geschrieben wird. Diese Session wird dann für alle Kommunikationen zwischen dem Server und dem Client benutzt. Die folgende Zeile Code zeigt die Signatur der Connect() Methode und dass diese weder Rückgabe- noch Übergabeparameter hat.

```
public async void Connect();
```

Um erfolgreich eine Session zu erstellen, werden die in Tabelle 1 aufgeführten Objekte benötigt, damit am Ende der Connect() Methode die Methode Session.Create(…) ausgeführt werden kann. In der ersten Spalte befindet sich die Position des Parameters beim Methodenaufruf. In der Zweiten und Dritten der Name und Typ des Parameters und in der vierten und letzten Spalte noch eine kurze Beschreibung.

*Tabelle 1: Übergabeparameter Session.Create(...)*

| Nr. | Name | Typ | Beschreibung |
|---|---|---|---|
| 1 | config | ApplicationConfiguration | Applikationskonfiguration der OPC UA Anwendung. |
| 2 | endpoint | ConfiguredEndpoint | Konfiguration des Endpoints/Servers. |
| 3 | updateBeforeConnect | bool | Ob die EndpointDescription vor dem Verbinden aktualisiert wird. |
| 4 | sessionName | string | Name der Session |
| 5 | sessionTimeout | uint | Zeit in ms nach welcher die Session geschlossen wird bei Inaktivität. |
| 6 | identity | IUserIdentity | Benutzer-Login der SPS, welches OPC UA Berechtigung hat |
| 7 | preferredLocales | IList<string> | Bevorzugte «Locales», welche Angaben über Sprache, Standort oder Firma enthalten |

Im Folgenden wird die gesamte Connect() Methode in Teilabschnitte aufgeteilt und dabei beschrieben, wie die Objekte «config» und «endpoint» erstellt werden. Bei den restlichen Objekten handelt es sich um Werte, die direkt in den Methodenaufruf von Session.Create(…) eingetragen werden können.

Als erstes wird wie folgt ein Objekt des Typs ApplicationInstance erstellt. Dabei wird der Typ der Applikations-Instanz festgelegt und der Name des Konfigurationsfiles ohne die Endung «.Config.xml» übergeben und abschliessend noch geladen. Anstelle des Client Applikationstypen könnte unter anderem mit einem Server Applikationstyp auch ein OPC UA Server erstellt werden. Beim Laden der Applikations-Konfiguration kann mittels dem Boolean Übergabeparameter bestimmt werden, ob die Methode «silent» ausgeführt werden soll. Beim Konfigurations-File handelt es sich um ein xml-File, welches die grundlegenden Parameter, wie zum Beispiel maximale Paketgrösse, bezüglich der Client-Server-Kommunikation beinhaltet. Es befindet sich bei den technischen Dokumenten unter «OPC_UA\Projects\BachelorThesis_OPC_UA» oder im Anhang 9.1.

```
ApplicationInstance application = new ApplicationInstance
{
    ApplicationType = ApplicationType.Client,
    ConfigSectionName = "Opc.Ua.Client" // name of config file without .Config.xml
};

// load the application configuration.
ApplicationConfiguration config = await application.LoadApplicationConfiguration(false);
```

In einem nächsten Schritt wird mit der folgenden Zeile die Zertifizierung ausgeschaltet, da beim Erstellen des Opc_Ua_Client-Objekts der Übergabeparameter «autoAccept» auf «true» gesetzt wurde. Das heisst, der Zertifikatsvalidator der Applikations-Konfiguration akzeptiert auch nicht vertrauenswürdige Zertifikate. Grundsätzlich könnten Einstellungen wie diese oder die Festlegung des Applikationstyps auch alle im xml-Konfigurationsfile gemacht werden. Somit ist der erste Übergabeparameter der Session.Create(…) Methode erstellt.

```
config.CertificateValidator.AutoAcceptUntrustedCertificates = autoAccept;
```

Anschliessend wird nach dem gewünschten Endpoint, also dem Server, gesucht und dann wird dieser konfiguriert. Dafür wird als erstes der gesamte URL des Servers «serverURL» zusammengestellt. Dieser setzt sich zusammen aus «opc.tcp://», der IP des Servers und dem OPC UA Port 4840. Für dieses Projekt ergibt sich demnach den URL opc.tcp://192.168.1.3:4840. Falls ein passender Server gefunden wird, wird dieser in dem EndpointDescription-Objekt «endpointDescription» abgespeichert. Die Methode «SelectEndpoint» der Klasse «CoreClientUtils» benötigt zum einen den Server URL und einen Boolean zur Festlegung, ob der gewünschte Endpoint gesichtert ist, als Übergabeparameter. Um die Konfiguration des Endpoints festzulegen wird dann das EndpointConfiguration-Objekt «endpointConfiguration» mit dem ApplicationConfiguration-Objket «config» erstellt. Dieses Objekt enthält auch die im oben genannten xml-File angegebenen Parameter zur Konfiguration der maximalen Paketgrösse etc. Schlussendlich wird der zweite Übergabeparameter der Session.Create(…) Methode, ConfiguredEndpoint «endpoint», mit der «endpointDescription» und der «endpointConfiguration» erstellt.

```
serverURL = "opc.tcp://" + serverIP + ":4840";

EndpointDescription endpointDescription = CoreClientUtils.SelectEndpoint(serverURL,
    false);

EndpointConfiguration endpointConfiguration = EndpointConfiguration.Create(config);

ConfiguredEndpoint endpoint = new ConfiguredEndpoint(null, endpointDescription,
    endpointConfiguration);
```

Zu guter Letzt wird die bereits erwähnte Kommunikations-Session zwischen dem Client und dem Server aufgebaut. Diese Session wird von nun an für alle Kommunikationen zwischen dem Server und dem Client benutzt. Gemäss der folgenden Zeile wird in diesem Projekt die Session erstellt. Dabei wird zur Authentifizierung ein neues Objekt des Typs «UserIdentity» mit den Membervariablen «userName» und «password» benutzt. Auf die benutzte SPS von Phoenix Contact kann standardmässig nicht ohne Login, beziehungsweise Authentifikation zugegriffen werden.

```
session = await Session.Create(config, endpoint, false, "OPC UA Console Client",
    60000, new UserIdentity(userName, password), null);
```

Abbildung 42 zeigt den Konsolenoutput nach dem ausführen der Methode Connect(). Die folgende Codezeile zeigt den Aufruf der Connect() Methode, welcher so auch im Beispielprogramm in Kapitel 4.4.2 angewendet wird.

```
client.Connect();
```

```
Connecting to Client opc.tcp://192.168.1.3:4840
Discovering endpoints of opc.tcp://192.168.1.3:4840
Selected server uses: Basic256Sha256
Creating a session with server
Connection successfully established
```

*Abbildung 42: OPC UA Client Output Connect()*

Da die Methoden Create(…) und LoadApplicationConfiguration(…) von der OPC Foundation her mit dem Ausdruck «await» aufgerufen werden müssen, muss die umschliessende Connect() Methode den Modifier async besitzen. Das heisst, die Connect Methode wird synchron ausgeführt, bis der erste «await» Ausdruck auftritt (LoadApplicationConfiguration(…)). Ab diesem Aufruf wird die umschliessende Methode, respektive der Task in der diese läuft, suspendiert, bis die mit «await» aufgerufene Methode vollständig ausgeführt wurde. Daraufhin läuft der Task dort weiter, wo er suspendiert wurde. In diesem Beispiel hat dieser async-await-Aufruf jedoch keinen Einfluss, da nur ein Task läuft. (Microsoft, 2022)

#### 4.4.1.2. OPC UA Client Show_All_Identifiers()

Mit der Show_All_Identifiers() Methode können alle Variablen des Servers auf der Konsole angezeigt werden, um sich einen Überblick zu verschaffen, welche zur Verfügung stehen. Auf der folgenden Zeile befindet sich die Signatur der Methode, welche keine Rückgabe- oder Übergabeparameter besitzt.

```
public void Show_All_Identifiers();
```

In dieser Methode wird nebst dem Konsolenoutput für die Überschrift jedoch lediglich die Methode Show_Identifiers(…) mit den Übergabeparametern «Arp.Plc.Eclr/» und 0 aufgerufen. Mit jener Methode werden alle Referenzen, welche Variablen oder Knoten zu weiteren Knoten sind, ausgegeben. Falls es sich um einen Knoten zu weiteren Variablen handelt, wird die Methode rekursiv aufgerufen, bis alle vorhanden Variablen angezeigt wurden. Grundsätzlich ist es dadurch möglich, dass bei gewissen OPC UA Server Strukturen endlose Rekursionen entstehen, wenn beispielsweise eine Maschine A auf eine Maschine B referenziert und andersrum auch noch. Bei diesem Projekt wird dies aber nicht der Fall sein.

Der folgende Codeabschnitt zeigt jene Methode. Sie hat keine Rückgabeparameter, jedoch den string «nodeId» und den int «indent» als Übergabeparameter. Wobei die «nodeId» den Identifier des gewünschten Knotens beinhaltet und der «indent» den Einzug in Anzahl Leerschlägen, um die unterschiedlichen Knotenebenen grafisch zu separieren (Abbildung 43). In der Methode wird als erstes der Knoten anhand der übergebenen «nodeId» erstellt. Beim Aufruf von Show_All_Identifiers() ist dieser Identifier zuerst «Arp.Plc.Eclr/», was gemäss Kapitel 4.3.4 dem Grundknoten einer SPS von Phoenix Contact entspricht. Die Zahl fünf repräsentiert in diesem Projekt den Namespace Index von Phoenix Contact. Deshalb wird dieser auch «hard coded». Weitere Informationen dazu befinden sich ebenfalls in Kapitel 4.3.3. (Mercado, 2022)

```
public void Show_Identifiers(string nodeId, int indent) {
    Node node = session.ReadNode(new NodeId(nodeId, 5));
    ReferenceDescriptionCollection refDescr = session.FetchReferences(node.NodeId);

    foreach (var rd in refDescr)
    {
        // check if rd is a following node and if it is type object or variable
        if (rd.IsForward == true && rd.DisplayName != "FolderType")
        {
            Console.WriteLine(…); // write Name and Node Type of rd

            if (rd.NodeClass == NodeClass.Object)
            {
                Show_Identifiers(nodeId + rd.DisplayName.ToString(), indent + 2);
            }
        }
    }
}
```

Anschliessend wird mit dem soeben erstellten Knoten «node» die ReferenceDescriptionCollection «refDescr» erstellt. Dabei handelt es sich grundsätzlich um eine Kollektion, bzw. Liste, welche alle Referenzen von und auf den Knoten «node» haben. Diese sogenannten Vorwärts- und Rückwärtsreferenzen sind sehr schön in Abbildung 40 zu sehen. Daraufhin wird durch diese Kollektion «refDescr» durchiteriert und wenn es sich bei einer dieser Referenzen um eine Vorwärtsreferenz handelt und zudem der «DisplayName» der Referenz nicht «FolderType» ist, wird der Name und der Knotentyp auf der Konsole mit dem übergebenen Einzug «indent» ausgegeben. Zum Schluss wird noch überprüft, ob es sich bei jener Referenz um den Knotentyp Objekt handelt. Falls ja, könnte es hinter diesem Knoten weitere Variablen haben und deswegen wird dann die Methode Show_Identifiers(…) mit jenem Knoten und einem «indent» von zwei zusätzlichen Leerschlägen ausgeführt.

Beim Aufruf der Methode gemäss der folgenden Zeile mit dem Server dieser Arbeit, sollte der Output der Abbildung 43 entsprechen. Dabei sind die ersten vier Elemente Referenzen des Knotens «Arp.Plc.Eclr/» und die letzten beiden Elemente Referenzen des Objekt-Knotens «MainInstance», was durch die Einschübe verdeutlicht ist. Zudem ist in Abbildung 43 auch gut das in Kapitel 4.3.4 angesprochene Information Model dieses Projekts zu sehen.

```
client.Show_All_Identifiers();
```

```
Name                 Type
--------             --------
b_OutletState        Variable
w_LightIntensity     Variable
w_SourceSensor       Variable
MainInstance         Object
  w_HystOn           Variable
  w_HystOff          Variable
```

*Abbildung 43: OPC UA Client Klasse Beispiel Output Show_All_Identifiers()*

#### 4.4.1.3. OPC UA Client Get_Value(…)

Um den aktuellen Wert einer Variable vom Server zu lesen, kann die Methode Get_Value(…) aus der OPC UA Client Klasse verwendet werden. Die Implementation der Methode ist im folgenden Codeabschnitt zu sehen. Als Rückgabeparameter besitzt sie ein Objekt des Typs «DataValue», welches den Wert und die Zeitstempel des Servers und des Clients enthalten. Der Übergabeparameter ist der Knoten Identifier «valueIdentifier» als String. In der Funktion wird mit dem gesamten Identifier und dem Namespace Index von fünf ein NodeId-Objekt erstellt, welches dann in der session.ReadValue(…) Methode verwendet wird, um den Wert über die Session vom Server zu holen.

```
public DataValue Get_Value(string valueIdentifier){
    return session.ReadValue(new NodeId("Arp.Plc.Eclr/" + valueIdentifier, 5));
}
```

Zum Lesen und Ausgeben der aktuellen Lichtintensität «w_LightIntensity» auf der Konsole kann die Get_Value(…) Methode wie folgt verwendet werden.

```
Console.WriteLine("Intensity:\t" + client.Get_Value("w_LightIntensity"));
```

Dabei ist zu beachten, dass der Knoten «w_LightIntensity», gemäss Abbildung 40, ein Folgeknoten des Objektknotens «Arp.Plc.Eclr» ist und deshalb lediglich der Variablenname beim Aufruf der Get_Value(…) benötigt wird. Damit zum Beispiel die Variable «w_HystOff», welche sich unter dem Objektknoten «MainInstance» befindet, gelesen werden kann, muss der String «MainInstance.w_HystOff» übergeben werden. Der folgende Code zeigt ein Beispiel Aufruf der Methode mit Ausgabe auf der Konsole.

```
Console.WriteLine("Upper Hyst.:\t" + client.Get_Value("MainInstance.w_HystOff"));
```

Abbildung 44 zeigt die Konsolenausgaben der Beispielanwendung aus Kapitel 4.4.2.

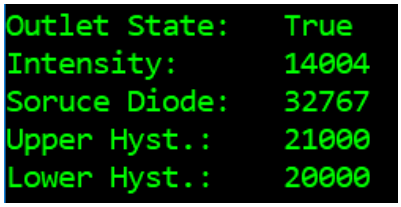



*Abbildung 44: OPC UA Client Klasse Beispiel Output Get_Value(...)*

#### 4.4.1.4. OPC UA Client Set_Value(…)

In diesem Kapitel wird die Methode Set_Value(…) der OPC UA Client Klasse beschrieben. Mit dieser Methode kann ein Wert einer Variable auf dem OPC UA Server gesetzt, beziehungsweise überschrieben werden. Die folgende Zeile Code zeigt die Signatur der Funktion, welche keinen Rückgabeparameter aber die zu überschreibende Variable als String «variable» und den zu schreibenden Wert als 16 Bit unsinged Integer (UInt16) «val» als Übergabeparameter besitzt. Der zu schreibende Wert ist vom Typ Uint16, da abgesehen von der Variable «b_OutletState» alle Variablen vom Typ Uint16 sind. Zudem kann die Variable «b_OutletState» nicht verändert werden, da sie im nächsten SPS-Zyklus wieder überschrieben werden würde. Um die Klasse vielseitig anwenden zu können, müsste diese Methode noch erweitert werden, damit sie auch mit anderen Variablentypen umgehen kann.

```
public void Set_Value(string variable, UInt16 val);
```

Zu Beginn der Methode wird das DataValue-Objekt «dataValue» erstellt. Es enthält lediglich den zu schreibenden Wert. Anschliessend wird das WriteValue-Objekt «writeValue» erstellt. Diesem Objekt wird dann mit einer NodeId mit dem eindeutigen Variablen Identifier und dem Namespace Index von fünf zugewiesen, welche Variable überschrieben werden soll. Daraufhin wird das «dataValue» Objekt dem «writeValue» Objekt zugewiesen und mit der Membervariable «AttributeId» auch definiert, dass es sich um einen Wert handelt, der geschrieben werden soll. Alternativ könnten auch eine NodeId oder ein EventNotifier gesetzt werden. Als letztes wird eine sogenannte WriteValueCollection «valuesToWrite» erstellt und das «writeValue» Objekt dort hinzugefügt. Dabei handelt es sich grundsätzlich um eine Liste, welche alle zu schreibenden Werte und ihre zugehörigen Variablen in Form von WriteValue-Objekten enthält. Mit dieser Liste besteht durch Hinzufügen von mehreren WriteValue-Objekten die Möglichkeit, mehrere Werte auf einmal zu schreiben. Dafür müsste die Set_Value(…) Methode aber erweitert werden, dass mehrere Variablennamen und Werte übergeben werden können. (HotExamples, 2022)

```
DataValue dataValue = new DataValue(new Variant(val));
WriteValue writeValue = new WriteValue();
writeValue.NodeId = new NodeId("Arp.Plc.Eclr/"+variable, 5);
writeValue.Value = dataValue;
writeValue.AttributeId = Attributes.Value;

WriteValueCollection valuesToWrite = new WriteValueCollection();
valuesToWrite.Add(writeValue);
```

In einem letzten Schritt wird mit der Write(…) Methode des «session» Objekts das eigentliche Schreiben auf dem OPC UA Server ausgeführt. Der erste Übergabeparameter der Write(…) Methode ist der Request Header, mit welchem zusätzliche Informationen wie zum Beispiel der Authentifizierungs-Token übermittelt werden könnte. In diesem Fall wird der Request Header aber nicht benötigt. Als zweiter Parameter wird die WriteValueCollection «valuesToWrite» übergeben. Beim dritten und vierten zu übergebenden Parameter handelt es sich um die vor der Ausführung der Write(…) Methode definierten Objekte «results» vom Typ StatusCodeCollection und «diagnosticInfos» des Typs DiagnosticInfoCollection. Mit diesen beiden Objekten kann nach dem Schreiben der Werte sichergestellt werden, dass die Werte korrekt geschrieben wurden und bei allfälligen Fehlern, die Fehlerursache herausgefunden werden. Der Einfachheit halber wird in diesem Projekt aber noch auf diese Fehlermeldungen verzichtet, weshalb beide mit «null» initialisiert werden. Es ist nicht möglich direkt «null» an der Methode zu übergeben, da es sich bei den Variablen, welche beim Aufruf mit «out» gekennzeichnet sind, um Referenz Typen handelt. Diese Referenz Typen sind eine Möglichkeit

mehrere Objekte als Output einer Methode zu erhalten, weswegen stets vorhandene Variablen übergeben werden müssen, auch wenn sie nur «null» sind. (Moldovean, 2022)

```
StatusCodeCollection results = null;
DiagnosticInfoCollection diagnosticInfos = null;

session.Write(null, valuesToWrite, out results, out diagnosticInfos);

ClientBase.ValidateResponse(results, valuesToWrite);
ClientBase.ValidateDiagnosticInfos(diagnosticInfos, valuesToWrite);
```

Der folgende Aufruf zeigt, wie die Set_Value(…) Methode angewendet wird, um die obere Hysterese-Grenze auf 21000 zu setzen. Hierbei muss darauf geachtet werden, dass der Methode der gesamte Identifier der Variable, abgesehen vom Startpunkt «Arp.Plc.Eclr/» übergeben wird. Das gesamte Beispielprogramm mit diesem und weiteren Aufrufen befindet sich im Kapitel 4.4.2. Zur Verdeutlichung der Funktionsweise der Set_Value(…) Methode ist in Abbildung 45 grafisch dargestellt, wie der Wert von «w_HystOff» von 20'000 auf 21'000 gesetzt wird. Mit dem UaExpert kann einfach kontrolliert werden, ob die Werte wie erwartet geschrieben wurden.

```
client.Set_Value("MainInstance.w_HystOff", 21000);
```

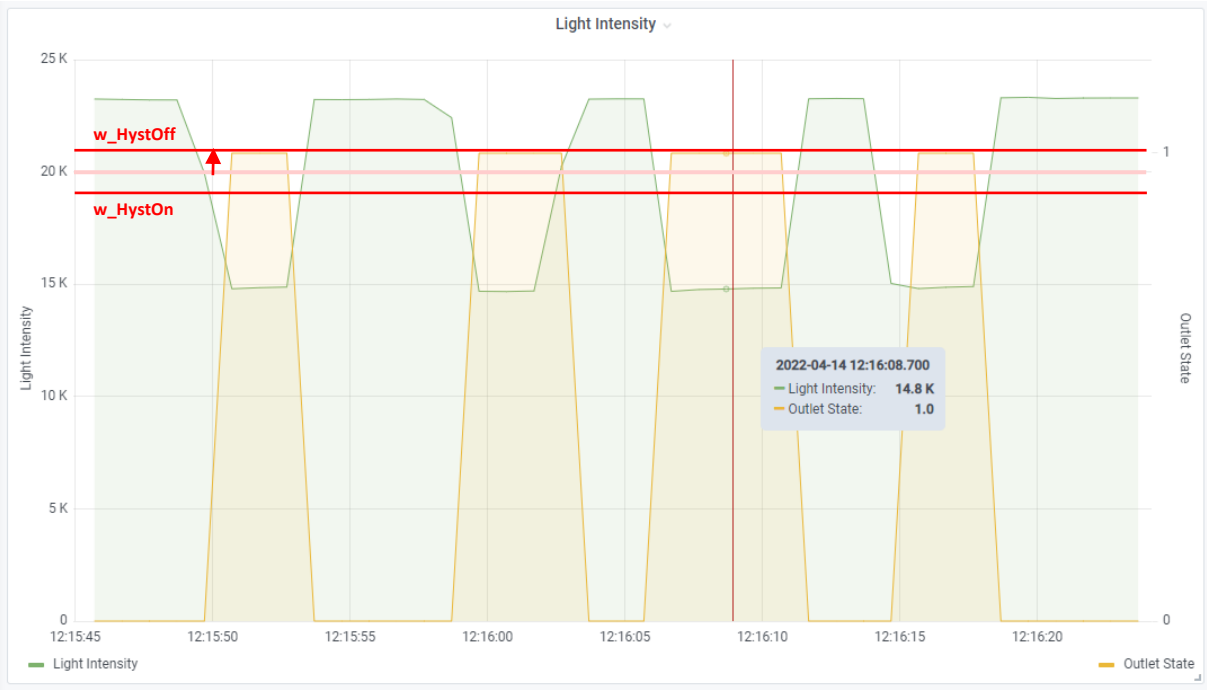



*Abbildung 45: OPC UA Client Klasse Beispiel Set_Value(...)*

#### 4.4.1.5. OPC UA Client Create_Monitored_Item(…)

In diesem Kapitel ist beschrieben, wie mit der Create_Monitored_Item(…) Methode ein Monitored Item für eine bestimmte Variable erstellt werden kann. Ein Monitored Item ist eine Möglichkeit als Client die Werte, welche auf dem OPC UA Server sind, zu überwachen und informiert zu werden, falls diese sich verändern. Dabei kann sich jeder beliebige Client mit einem Server verbinden und die von ihm gewünschten Variablen mit Monitored Items überwachen lassen. Das Pendant zu einem Monitored Item auf der Serverseite sind Alarms und Events, welche serverseitig erstellt werden müssen, um gewisse Veränderungen festzustellen (OPC Foundation, 2022) (TopView, 2022). Mit einem Monitored Item können Attribute, Werte oder Knoten überwacht werden. Dabei wird bei jeder Veränderung des ausgewählten Elements eine Notifikation über die Subscription der Session an den Client gesendet. Um nicht bei jeder kleinen Veränderung eine Notifikation zu erhalten, können, wie in diesem Kapitel erklärt wird, mit einem DataChangeFilter unwesentliche Notifikationen bewusst herausgefiltert werden. In diesem Projekt könnte zum Beispiel die Lichtintensität «w_LightIntensity» mit einem Monitored Item so überwacht werden, dass eine Notifikation bei einer Veränderung des Werts um 1000 Einheiten innerhalb eines bestimmten Sampling Intervalls gesendet wird. (OPC Foundation, 2022)

Die folgende Zeile Code zeigt die Signatur der Methode. Sie besitzt den Node Identifier (ohne «Arp.Plc.Eclr/») als String und den Grenzwert des Filters als Double Variable als Übergabeparameter. Auf die genauere Benutzung und Beschreibung des «deadbandValue» wird im Verlaufe dieses Kapitel eingegangen. Die Methode besitzt keinen Rückgabeparameter, am Ende der Methode wird jedoch eine Endlosschleife ausgeführt, welche die eintreffenden Notifikationen auflistet. In einem anderen Projekt würde diese Methode womöglich in einem separaten Task ausgeführt werden oder die Notifikationen nur zu einem bestimmten Zeitpunkt abgefragt werden.

```
public void Create_Monitored_Item(string identifier, double deadbandValue);
```

In einem ersten Schritt wird die Subscription «subscription» erstellt, über welche dann alle Notifikationen aller Monitored Items übermittelt werden. In dieser Subscription muss zum einen das Veröffentlichen von Notifikationen mit dem Setzen der Membervariable «PublishingEnabled» auf «true» eingeschaltet werden. Zum andern wird ein Publishing Interval von 1000 ms eingestellt. Mit diesem Intervall wird festgelegt, wie regelmässig Notifikationen versendet werden können.

```
Subscription subscription = new Subscription();
subscription.PublishingEnabled = true;
subscription.PublishingInterval = 1000;
```

Anschliessend wird das eigentliche Monitored Item «monitoredItem» erstellt und ihm direkt die NodeId mit dem eindeutigen Identifier und dem Namespace Index von 5 zugewiesen. Beim Monitored Item, kann der Sampling Intervall mit der Membervariable «SamplingInterval» gesetzt werden. Dieser Intervall bestimmt den zeitlichen Abstand in Millisekunden zwischen welchem die Werte einer Variable verglichen werden. Falls der «SamplingInterval» auf -1 gesetzt wird, wird der «PublishingInterval» der Subscription als «SamplingInterval» geerbt. In diesem Fall würde der «SamplingInterval» 1000 ms betragen. Das heisst, wenn beispielsweise die Lichtintensität innerhalb von einer Sekunde mehr als eine bestimmte Veränderung erfährt, wird eine Notifikation gesendet. Das Einstellen dieses bestimmten Wertes wird im folgenden Textabschnitt erklärt. Zum Schluss wird der «MonitoringMode» noch auf «Reporting» gesetzt, damit die Notifikationen auch gemeldet werden. Falls dieser Mode auf «Sampling» oder «Disabled» gesetzt ist, werden die Notifikationen im letzten Schritt der Methode nicht angezeigt. (HotExamples, 2022)

```
MonitoredItem monitoredItem = new MonitoredItem();
monitoredItem.StartNodeId = new NodeId(new NodeId("Arp.Plc.Eclr/" + identifier, 5));
monitoredItem.SamplingInterval = -1;
monitoredItem.MonitoringMode = MonitoringMode.Reporting;
```

Im nächsten Schritt wird der bereits erwähnte DataChangeFilter «filter» erstellt, damit Notifikationen bei unwesentlichen Veränderungen nicht gesendet beziehungsweise reported werden. Dabei wird als erstes der Filtertyp mit der Membervariable «DeadbandType» auf «Absolute» gesetzt. Damit filtert der Filter bis zu einer bestimmten absoluten Wert-Veränderungen. Eine weitere Filtertyp-Möglichkeit wäre der prozentuale Filter, welcher bis zu einer bestimmten prozentualen Veränderung filtert. Mit dem «DeadbandValue» kann dieser bestimmte absolute oder prozentuale Wert gesetzt werden, in welchem Bereich die Notifikationen gefiltert werden. Das heisst, wenn bei einem absoluten Filter, der «DeadbandValue» auf 1000 gesetzt wird, und der Wert des Monitored Items, sich innerhalb von 1000 ms nur um 900 Einheiten verändert, wird die Notifikation gefiltert und nicht an den Client gesendet. Falls sich dieser Wert aber innerhalb von 1000 ms um mehr als 1000 Einheiten verändert, wird die Notifikation zum Client gesendet. Mit der Membervariable «Trigger» kann festgelegt werden, ob beim Filtern, nur die Werte beachtet werden («StatusValue») oder ob auch die Zeitstempel der Werte beachtet werden («StatusValueTimestamp»). Falls auch die Zeitstempel beachtet werden, bemerkt der Filter anhand des Zeitstempels, ob ein frischer Wert vorhanden ist, auch wenn der eigentliche Wert genau derselbe ist. Ansonsten wird nur der Wert beachtet. Als letztes wird der «filter» dann dem «monitoredItem» zugewiesen. (OPC Labs, 2022) (QT, 2022)

```
DataChangeFilter filter = new DataChangeFilter();
filter.DeadbandType = (uint)DeadbandType.Absolute;
filter.DeadbandValue = deadbandValue;
filter.Trigger = DataChangeTrigger.StatusValue;
monitoredItem.Filter = filter;
```

Zu guter Letzt wird im folgenden Codeabschnitt das «monitoredItem» der «subscription» zugewiesen, die «subscription» der «session» zugewiesen und dann die «subscription» gestartet. Durch die beiden Add-Methoden wäre es möglich, mehrere Monitored Items einer Subscritption zuzuweisen und/oder mehrere Subscriptions einer Session zuzuweisen.

```
subscription.AddItem(monitoredItem);
session.AddSubscription(subscription);
subscription.Create();
```

Die folgende foreach-Schleife zeigt wie die einzelnen NotificationMessages «message» am Ende der Create_Monitored_Item(…) Methode aus der Membervariable «Notifications» des Objekts «subscription» geladen und dann auf der Konsole ausgegeben werden.

```
foreach (NotificationMessage message in subscription.Notifications)
{
    var DataChanges = message.GetDataChanges(true);

    Console.WriteLine("Notification at " + DataChanges[0].Message.PublishTime +
        ": Value changed to " + DataChanges[0].Value);
}
```

Wird die obige foreach-Schleife nun gemäss der Implementation in den technischen Dokumenten unter «xyz» in eine Endlosschleife gepackt, werden alle Notifikationen sauber aufgelistet. Nebst der PublishingTime und dem Wert der zu überwachenden Variable nach der Veränderung, besitzt das «DataChanges.Message» Objekt noch viele weiter interessante Membervariablen, wie zum Beispiel

die «SequenceNumber» welche benutzt werden kann, um festzustellen, ob Notifikationen verloren gegangen sind.

Die folgende Zeile Code zeigt, wie das in Kapitel 4.4.2 beschrieben Beispielprogramm ein Monitored Item für die Variable «w_LightIntensity» mit einem Deadband Value von 1000 Einheiten erstellt. Der Filtertyp und die Intervallzeiten sind bis anhin in der Methode standartmässig auf absolut beziehungsweise 1000 ms gesetzt. Dieses Monitored Item sendet also eine Notifikation an den Client, falls der Wert von «w_LightIntensity» innerhalb von 1000 ms eine absolute Veränderung von mehr als 1000 Einheiten erfährt. Zur Verdeutlichung der Funktionsweise eines Monitored Items ist in Abbildung 46 ein Beispiel eingezeichnet, bei welchem mit dem oben definierten Monitored Item eine Notifikation gesendet werden würde, da der Wert der Lichtintensität innerhalb einer Sekunde (gekennzeichnet durch die beiden roten senkrechten Striche) sich um mehr als 1000 Einheiten verändert.

```
client.Create_Monitored_Item("w_LightIntensity", 1000);
```

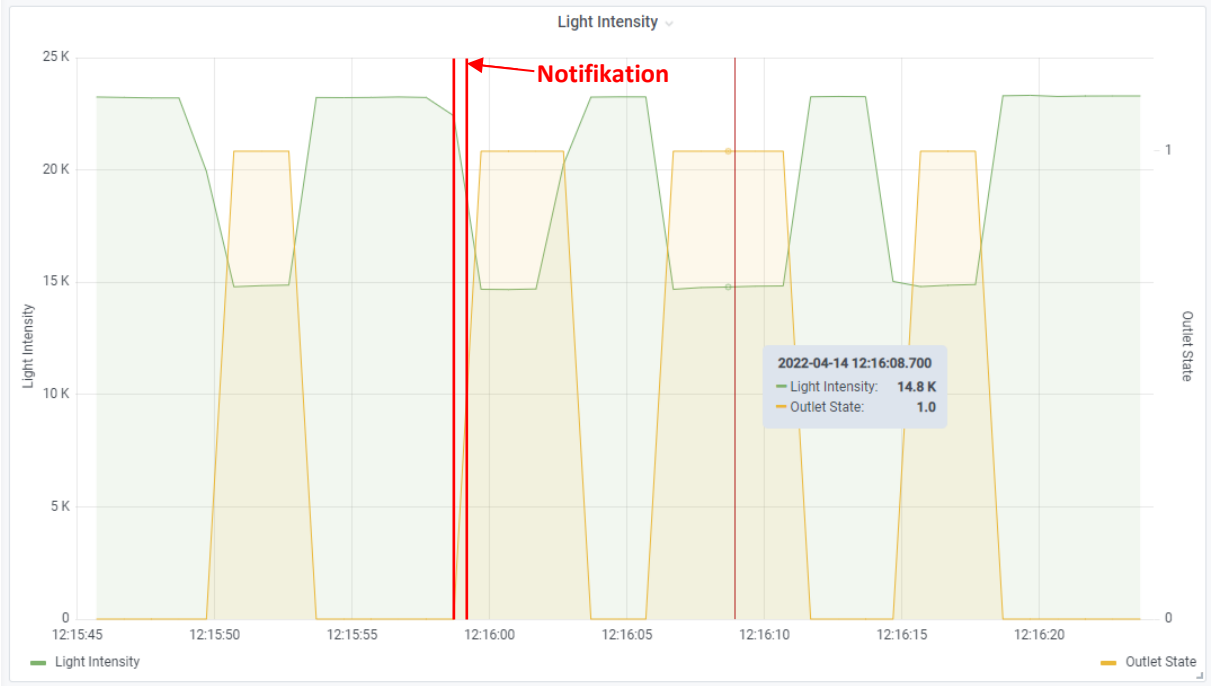



*Abbildung 46: OPC UA Client Klasse Beispiel Monitored Item Notifikation*

Wird die oben definierte Methode ausgeführt und die Photodiode ein paarmal zu- und wieder abgedeckt, wird in der Konsole der in Abbildung 47 gezeigte Output erscheinen.

```
Notification at 07.06.2022 14:02:24: Value changed to 31135
Notification at 07.06.2022 14:02:32: Value changed to 14129
Notification at 07.06.2022 14:02:33: Value changed to 24052
Notification at 07.06.2022 14:02:34: Value changed to 31127
Notification at 07.06.2022 14:02:41: Value changed to 14377
Notification at 07.06.2022 14:02:50: Value changed to 31064
Notification at 07.06.2022 14:02:53: Value changed to 16091
Notification at 07.06.2022 14:02:54: Value changed to 14227
Notification at 07.06.2022 14:02:55: Value changed to 31049
```

*Abbildung 47: OPC UA Client Klasse Beispiel Output Create_Monitored_Item(...)*

### 4.4.2. OPC UA Client Beispielanwendung

In den technischen Dokumenten unter «OPC_UA\Projects» befindet sich ein C#-Projekt mit dem Namen «BachelorThesis_OPC_UA», welches die OPC UA Client Klasse (OPC_UA_Client.cs), das XML-Konfigurationsfile (Opc.Ua.Client.Config.xml) und ein Beispielprogramm (Program.cs), gemäss dem Aufbau von Abbildung 41, beinhaltet. Dieses Beispielprogramm enthält Anwendungsbeispiele der Methoden, welche in den Kapiteln 4.4.1.1 bis 4.4.1.5 dokumentiert wurden. In Abbildung 48 ist der zu erwartende Output beim Ausführen dieses Programms gezeigt. Nachdem der letzte Output («Lower Hyst.: 20000») geschrieben wurde, wird die Konsole geleert und es erscheinen die Notifikationen des in Kapitel 4.4.1.5 erwähnten Monitored Item gemäss Abbildung 47.

```
Connecting to Client opc.tcp://192.168.1.3:4840
Discovering endpoints of opc.tcp://192.168.1.3:4840
Selected server uses: Basic256Sha256
Creating a session with server
Connection successfully established

Name                 Type
--------             --------
b_OutletState        Variable
w_LightIntensity     Variable
w_SourceSensor       Variable
MainInstance         Object
  w_HystOn           Variable
  w_HystOff          Variable



Outlet State:   True
Intensity:      14424
Soruce Diode:   32767
Upper Hyst.:    21000
Lower Hyst.:    20000
```

*Abbildung 48: OPC UA Client Klasse Beispielanwendung Konsolenoutput*

# 5. Proficloud.io

In diesem Kapitel wird die Inbetriebnahme der Cloud Proficloud.io der Firma Phoenix Contact beschrieben. Dabei sollen die in PLCnext Engineer ausgewählten Variablen auf der Cloud abgespeichert werden. Proficloud.io bietet insgesamt fünf verschiedene Smart Services auf der Cloud an. In Tabelle 2 sind diese fünf Service jeweils mit Namen und einer kurzen Beschreibung aufgelistet. (Proficloud, 2022)

*Tabelle 2: Proficloud.io Smart Services*

| Service-Name | Service-Beschreibung |
|---|---|
| Device Management | Mit dem Device Management Service können alle Cloud-Geräte von Phoenix Contact mit einer einfachen grafischen Benutzeroberfläche verwaltet und überwacht werden. |
| User Management | Im User Management können unterschiedliche Benutzer und Organisationen verwaltet und zur Cloud eingeladen werden Zudem können diesen auch bestimmte Berechtigungen zugewiesen werden. |
| EMMA Service | Der Energy Monitoring, Management, Analytics (EMMA) Service dient zur intelligenten Energiemessung-, Überwachung und -Analyse. |
| Time Series Data | Mit dem Time Series Data Service können aktuelle und historische Daten von der Cloud mit Leichtigkeit in einer selbst erstellbaren grafischen Oberfläche von Grafana dargestellt werden. |
| Impulse Analytics | Der Impulse Analytics Service ist ein Assistenzsystem für Überspannungsschutz. |

Wovon in diesem Projekt jedoch nur der Device Management und der Time Series Data Management Service benutzt wird. Als erstes muss die SPS mit dem Internet, respektive der Cloud verbunden werden. Anschliessend müssen im PLCnext Engineer die Variablen konfiguriert werden, dass diese auf der Cloud abgespeichert werden. Dann muss die benutzte SPS mit dem Device Management Service der Cloud hinzugefügt werden. Und zu guter Letzt können dann die Werte der Variablen mit dem Time Series Data Service eingesehen und grafisch dargestellt werden. Diese Schritte werden in den folgenden Abschnitten detailliert beschrieben.

## 5.1. Verbindung der SPS zum Internet

Damit die SPS eine Verbindung zum Internet und somit zur Cloud herstellen kann, muss die SPS nun direkt mit dem Ethernet-Kabel an einen Router gehängt werden. Eine weitere Möglichkeit, die SPS ins Internet zu bringen wäre, das Erstellen einer Netzwerkbrücke, wobei das Internet vom Laptop über das Ethernet-Kabel an die SPS weitergeleitet wird.

Anschliessend muss im Web Based Management (WBM) der SPS unter «Configuration/Network» gemäss Abbildung 49 das Default Gateway richtig gesetzt werden. Um auf das WBM zu gelangen, muss in einem Browser nach der IP der SPS mit anschliessendem «/wbm» gesucht werden. Bei diesem Projekt kann das WBM mit «192.168.1.3/wbm» erreicht werden. Im Falle eines Heimnetzwerks ist das Default Gateway meistens 192.168.1.1. Falls das Default Gateway nicht bekannt ist, kann dieses mit dem Befehl «ipconfig» in der Windows Eingabeaufforderung eingesehen werden. Nach dem Setzen des Default Gateways, müssen die Änderungen mit dem Button «Apply and Reboot» gespeichert werden.

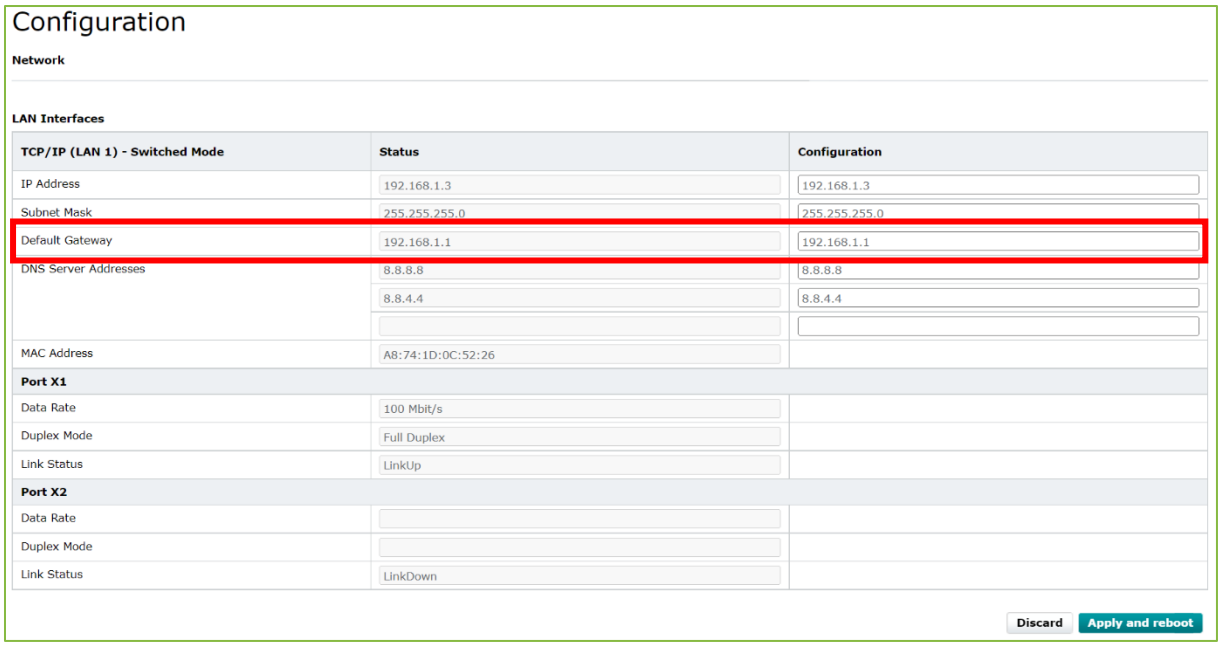



*Abbildung 49: PLCnext Web Based Management Gateway Konfiguration*

Um zu überprüfen, ob die Herstellung der Internetverbindung erfolgreich war, kann mit einer Secure Shell auf das Linux Betriebssystem, welches auf der SPS läuft, zugegriffen werden und dort ein ping auf eine Webseite ausgeführt werden.

Des Weiteren sollte im WBM unter «Configuration/Proficloud Services», wie in Abbildung 50 gezeigt, der Proficloud Status auf Online gewechselt haben. Dann ist die SPS mit der Cloud verbunden und es können nun, gemäss dem folgenden Kapitel (5.2), die Variablen in PLCnext Engineer konfiguriert werden. Es kann durchaus sein, dass die blauen Häckchen in Abbildung 50 noch gesetzt werden müssen und die SPS neu gestartet werden muss, da diese Verbindungsherstellung lange dauern kann.

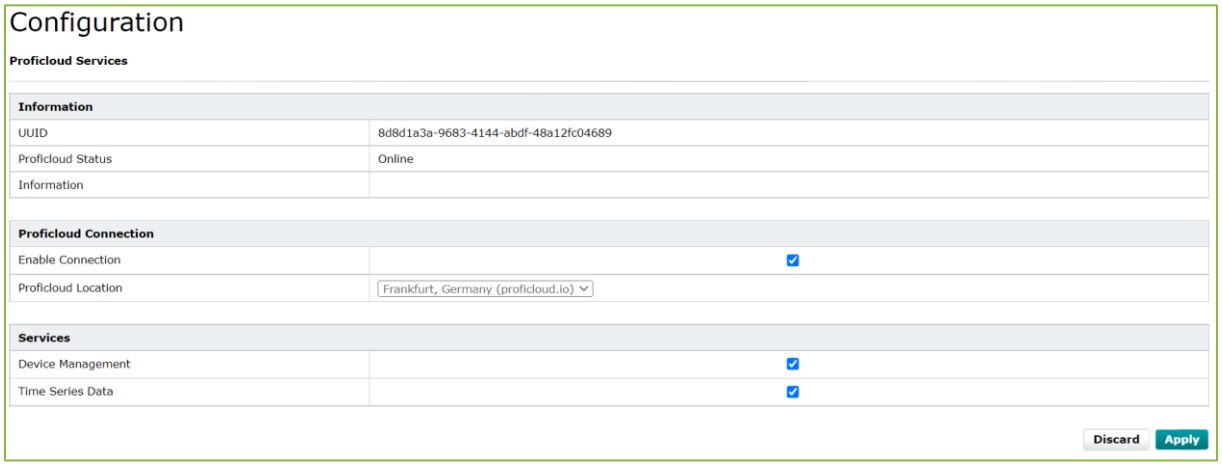
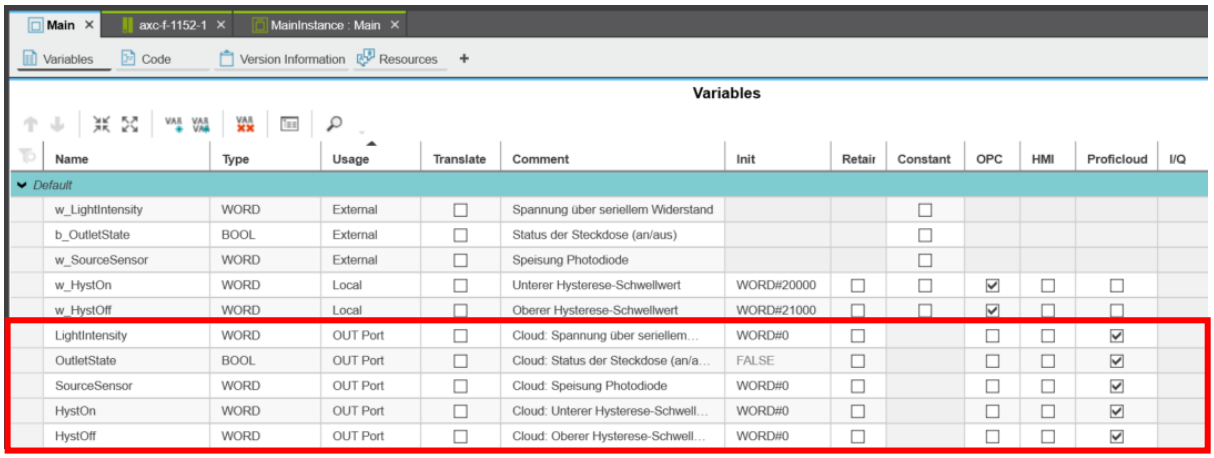



*Abbildung 50: PLCnext Web Based Management Proficloud Services*

## 5.2. PLCnext Engineering Variablen Konfiguration

Ähnlich wie in Kapitel 4.2, in welchem die Variablen auf dem OPC UA Server konfiguriert wurden, werden in diesem Kapitel die gewünschten Variablen so konfiguriert, dass diese dann auf der Proficloud sichtbar sind. Um eine Variable auf der Cloud sichtbar zu machen, muss diese in der Variablenliste eines Programmes in PLCnext Engineer zum einen die «Usage» «OUT Port» haben und zum andern muss in der «Proficloud» Spalte das Häkchen gesetzt sein. Aber da die Variablen w_LightIntensity, w_SourceSensor und b_Output die «Usage» External haben müssen, können diese nicht einfach auf «OUT Port» gesetzt werden. Deshalb wurde im PLCnext Engineer Projekt im COMPONENTS Menü unter «Programming/Local/Programs/Main» im Reiter «Variables» gemäss der roten Box in Abbildung 51 für jede Variable eine zusätzlich Variable erstellt, welche die «Usage» «OUT Port» und das Häkchen bei der «Proficloud» Spalte haben.




| Name | Type | Usage | Translate | Comment | Init | Retain | Constant | OPC | HMI | Proficloud | I/Q |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Default | | | | | | | | | | | |
| w_LightIntensity | WORD | External | ☐ | Spannung über seriellem Widerstand | | | ☐ | | | | |
| b_OutletState | BOOL | External | ☐ | Status der Steckdose (an/aus) | | | ☐ | | | | |
| w_SourceSensor | WORD | External | ☐ | Speisung Photodiode | | | ☐ | | | | |
| w_HystOn | WORD | Local | ☐ | Unterer Hysterese-Schwellwert | WORD#20000 | ☐ | ☐ | ☑ | ☐ | ☐ | |
| w_HystOff | WORD | Local | ☐ | Oberer Hysterese-Schwellwert | WORD#21000 | ☐ | ☐ | ☑ | ☐ | ☐ | |
| LightIntensity | WORD | OUT Port | ☐ | Cloud: Spannung über seriellem… | WORD#0 | ☐ | | ☐ | ☐ | ☑ | |
| OutletState | BOOL | OUT Port | ☐ | Cloud: Status der Steckdose (an/a… | FALSE | ☐ | | ☐ | ☐ | ☑ | |
| SourceSensor | WORD | OUT Port | ☐ | Cloud: Speisung Photodiode | WORD#0 | ☐ | | ☐ | ☐ | ☑ | |
| HystOn | WORD | OUT Port | ☐ | Cloud: Unterer Hysterese-Schwell… | WORD#0 | ☐ | | ☐ | ☐ | ☑ | |
| HystOff | WORD | OUT Port | ☐ | Cloud: Oberer Hysterese-Schwell… | WORD#0 | ☐ | | ☐ | ☐ | ☑ | |

*Abbildung 51: PLCnext Engineer Proficloud Variablen*

In einem letzten Schritt müssen die Werte der Klemmen im Code noch diesen neu erstellten Cloud-Variablen zugewiesen werden. Dies kann durch Hinzufügen des folgenden Codeabschnitts zum Code unter im Reiter «Code» gemacht werden. Das gesamte PLCnext Projekt befindet sich in den technischen Dokumenten unter «PLC_Next\PLC_Next_Engineer\Projects». Anschliessend sollte nach dem Laden des Projekts auf die SPS die Cloud vorbereitet sein. Im nächsten Kapitel (5.3) wird beschrieben, wie die SPS dem Device Management Service hinzugefügt wird, damit das Gerät anschliessend im Kapitel 5.4 mit dem Time Series Data Service benutzt werden kann.

```
// Update Proficloud
LightIntensity := w_LightIntensity;
OutletState    := b_OutletState;
SourceSensor   := w_SourceSensor;
HystOff        := w_HystOff;
HystOn         := w_HystOn;
```

## 5.3. Proficloud.io Device Management

Nachdem die SPS konfiguriert wurde und die Daten nun auf der Cloud sind, wird in diesem Kapitel die SPS dem Device Management Service hinzugefügt, damit das Gerät im nächsten Kapitel (5.4) mit dem Time Series Data Service verwendet werden kann, um die Werte grafisch darzustellen. Falls noch kein Proficloud.io Konto vorhanden ist, muss dieses als erstes unter folgendem Link erstellt werden https://app.proficloud.io/register. Anschliessend kann im Device Management Service mit dem «ADD DEVICE» Button gemäss Abbildung 52 ein Gerät hinzugefügt werden. Das Hinzufügen der Location ist optional.

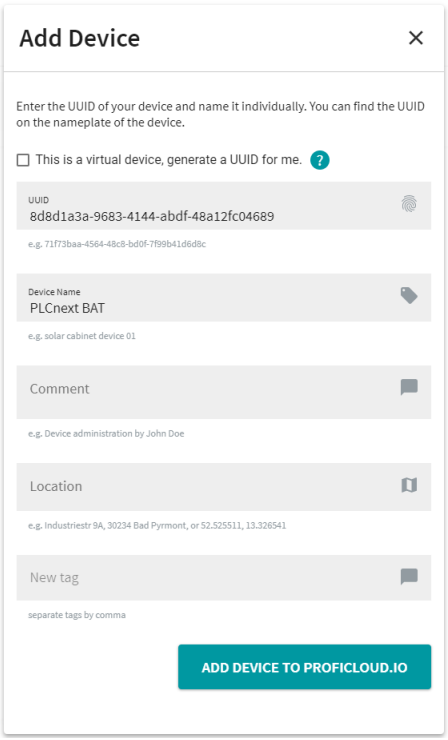



*Abbildung 52: Proficloud.io Device Management Service Add Device*

Der UUID (Universally Unique Identifier) ist eine Nummer zur eindeutigen Unterscheidung aller Geräte. Dieser kann, wie in Abbildung 53 in der roten Box gezeigt, im WBM unter «Configuration/Proficloud Services» bei den Informationen ausgelesen werden.

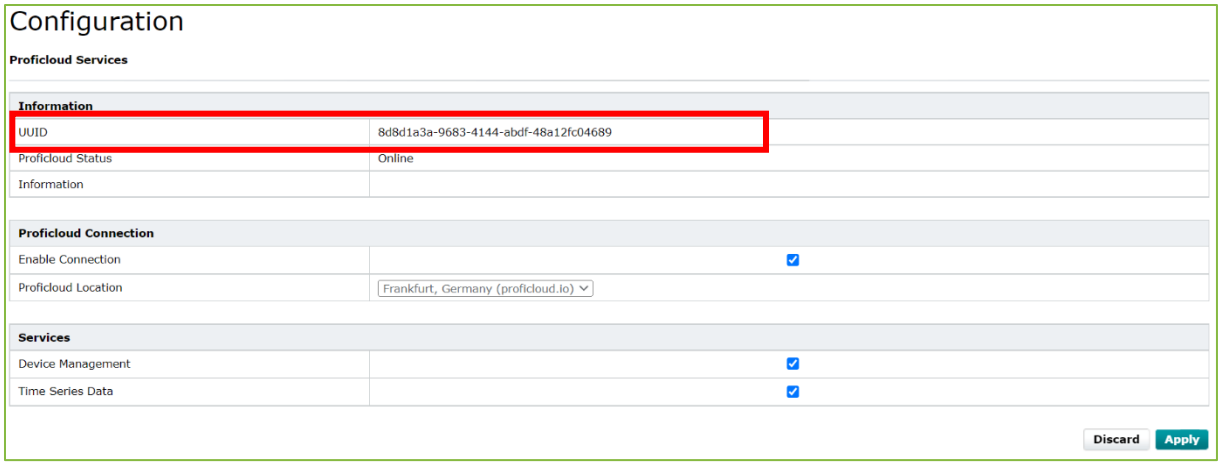



*Abbildung 53: PLCnext Web Based Management Proficloud Services UUID*

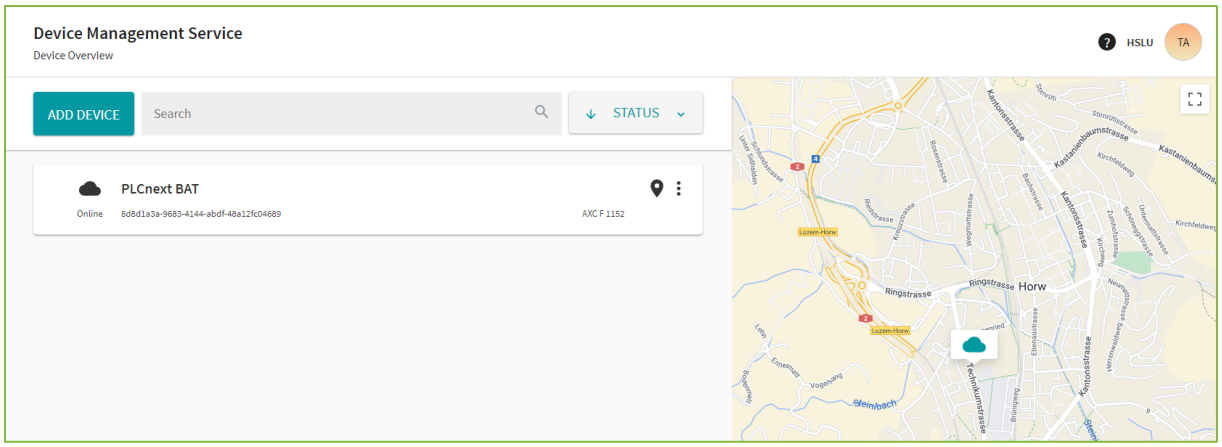



*Abbildung 54: Proficloud.io Device Management Service Device Overview*

Daraufhin sollte das hinzugefügte Gerät in der «Device Overview» gemäss Abbildung 54 angezeigt werden.

In einem letzten Schritt müssen die sogenannten Metrics noch zugewiesen werden. Dabei geht es um jene Variablen, welche anschliessend im Time Series Data Service benutzt werden sollen. Durch Betätigen des Geräts in der «Device Overview», werden detailliertere Informationen zur SPS angezeigt. Dort kann im Reiter «Service» mit dem «ASSIGN METRIC» Button das in Abbildung 55 gezeigte Fenster geöffnet werden. In diesem Fenster werden nebst zwei Mustervariablen (auto~con_LogLevel & auto~con_TrafficLightColor) die zur Verfügung stehenden Variablen der SPS angezeigt. Es werden auch jene Variablen angezeigt, welche im PLCnext Engineer nicht mit Proficloud gekennzeichnet wurden. Diese sind jedoch überflüssig, da dort keine Werte hinterlegt sind, welche mit dem Time Series Data Service eingesehen werden könnten. Deshalb müssen lediglich die in Abbildung 55 rot markierten Variablen ausgewählt und mit «ASSIGN METRICS» bestätigt werden. Somit ist die Proficloud vorbereitet, dass im nächsten Kapitel die Variablen mit dem Time Series Data Service eingesehen werden können.

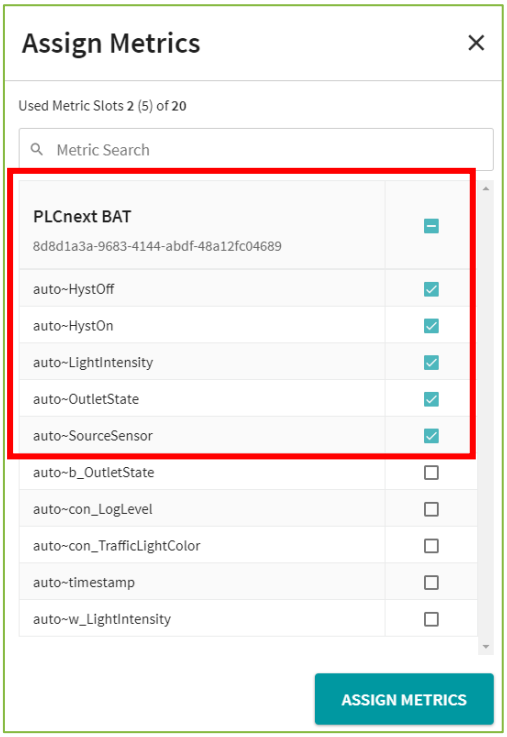



*Abbildung 55: Proficloud.io Device Management Service Assign Metrics*

## 5.4. Proficloud.io Time Series Data Service

Als letzter Schritt wird in diesem Abschnitt beschrieben, wie mit dem Time Series Data Service die Daten von der Proficloud geladen und in einem Dashboard nach belieben dargestellt werden können. Abbildung 56 zeigt ein Beispiel eines Dashboards dieses Projekts, in welchem die Lichtintensität, der Status der Steckdose und die Hysterese Grenzen unterschiedlich dargestellt werden.

Der Time Series Data Service ist auf Grafana aufgebaut, einer mächtigen Open-Source-Anwendung für grafische Darstellungen. In diesem Projekt werden lediglich auf einige Grundlagen der Dashboard Erstellung eingegangen, um zu verifizieren, dass die Daten korrekt verarbeitet und geladen werden konnten. Viele weitere Tipps und Tricks zu Grafana können unter dem folgenden Link gefunden werden: https://grafana.com/docs/.


*Abbildung 56: Proficloud.io Time Series Data Service Sample Dashboard*

Im Timer Series Data Service kann am linken Fensterrand mit dem Plus-Button ein neues Dashboard erstellt werden. Daraufhin erscheint ein leeres Dashboard, welches nun mit Panels besetzt werden kann. In solchen Panels können unterschiedlichste Anzeigen, wie zum Beispiel Grafen, Balkendiagramme oder Heatmaps, ausgewählt werden. Durch Betätigen des «Add new Panel» Buttons wird ein Graf-Panel hinzugefügt und das dazugehörige Bearbeitungsfensters geöffnet.

Als erstes muss im unteren Bereich im «Query» Reiter gemäss Abbildung 57 auf der «FROM» Zeile mit dem Plus, der «uuid» als Tag hinzugefügt werden. Dann kann mit «select tag value» die UUID der SPS ausgewählt werden. Zum Schluss kann die gewünschte Variable mit «select measurement» ausgewählt werden, worauf die Werte im Panel angezeigt werden sollten.


*Abbildung 57: Proficloud.io Time Series Data Service Panel Datenauswahl (Query)*

Auf der Zeile «ALIAS BY» kann diesem Datensatz einen Namen gegeben werden, welcher unter anderem in der Legende der Grafik angezeigt wird. Zum Hinzufügen eines weiteren Datensatzes, beispielsweise der oberen Hysteresegrenze, im selben Grafen kann der «+ Query» in der unteren linken Ecke betätigt und anschliessend wie oben beschrieben vorgegangen werden. Durch Betätigen des «Apply» Buttons, wird das Panel gespeichert und das gesamte Dashboard sichtbar. Dort kann durch Ziehen an der rechten unteren Ecke eines Panels, die Grösse davon angepasst werden. Des Weiteren kann durch Ziehen am oberen Rand des Panels die Position verändert werden.

In den technischen Dokumenten unter «Proficloud.io» befindet sich das in Abbildung 56 gezeigte Beispiel Dashboard als JSON-File. Am rechten Fensterrand im Time Series Data Service kann im «Create» Menü unter «Import» dieses File importiert und somit jenes Dashboard geladen werden.

# 6. REST Client Klasse und Anwendung

Dieses Kapitel enthält die Dokumentation des Aufbaus und der Anwendung der selbst erstellten REST C#-Konsolenapplikation. Mit dieser Anwendung sollen Daten über REST von der zuvor konfigurierten Proficloud geladen werden können. Als erstes wird dabei kurz beschrieben was REST ist, wie es aufgebaut ist und wofür es verwendet werden kann. Anschliessend folgt der Aufbau der Applikation mit einem Beschrieb der eigenen REST_Client Klasse und potenziellen Erweiterungsmöglichkeiten. Zum Schluss wird noch die Benutzung der Applikation anhand eines Beispielprogramms erwähnt.

## 6.1. REST Grundlagen

Ein Representional State Transfer, kurz REST, ist ein Weg zur Kommunikation über ein Netz oder das Internet. In der heutigen Zeit werden viele Web-Dienste mit REST realisiert, da der Aufbau von REST auf dem World Wide Web (WWW) gestützt ist. Grundsätzlich funktioniert REST so, dass der Client mit einer API einen Request an einen Server sendet, welcher den Befehl darin dann ausführt und unter Umständen eine Response in Form eines JSON, HTML oder XML an den Client zurücksendet. Sehr oft geschieht dieses Versenden von Befehlen und Nachrichten über http und den dazugehörenden Methoden. Tabelle 3 zeigt die wichtigsten http-Methoden, wobei in der ersten Spalte der Methoden-Name und in der zweiten eine kurze Methoden-Beschreibung aufgelistet ist. (Srocke, 2022)

*Tabelle 3: http-Methoden*

| Methoden-Name | Methoden-Beschreibung |
|---|---|
| GET | Laden von Daten vom Server. |
| POST | Senden von Daten an den Server. |
| PUT | Anpassen von Daten auf dem Server. |
| DELETE | Löschen von Daten auf dem Server. |

Eine solches http-Paket besitzt jeweils einen Header und einen Body. Im Body befindet sich die eigentliche Nachricht, welche vom Server verarbeitet wird. Dabei handelt es ich oft um einen Pfad des zu ladenden Files oder eine URL einer Webseite. Im Header befinden sich zusätzliche Informationen, wie zum Beispiel die Grösse des Body-Inhalts, die Informationen zur Authentifikation oder Angaben zum Typ des Inhalts. In den folgenden Kapiteln wird beispielsweise die User-Authentifikation für die Cloud in den Header gepackt.

## 6.2. REST Konsolenanwendung

In diesem Abschnitt wird der Aufbau, die Implementation und die Benutzung der REST C#-Konsolenanwendung dokumentiert. Dabei liegt der Fokus auf einer eigenen Klasse, mit welcher eine Verbindung zu der Proficloud hergestellt werden kann und dann die gewünschten Daten von der Proficloud geladen werden können.

Zur Entwicklung der Anwendung wurde Microsoft Visual Studio Enterprise 2019 Version 16.8.3 verwendet. Wiederum wurde ein Konsolenanwendungs-Projekt (Console App) verwendet. In den technischen Dokumenten im Ordner «REST\Project» befindet sich das Projekt mit der eigens erstellten REST_Client Klasse. Zudem befindet sich in diesem Projekt auch ein Programm als Beispiel zur Verwendung der REST_Client Klasse. Dieses Programm entspricht dem in Abbildung 58 gezeigten Ablauf. Die detailliertere Funktionsweise, der Aufbau und die Implementation der Klasse werden in den folgenden Kapiteln beschrieben.

Abbildung 58 zeigt ein Beispiel, wie die Klasse angewendet werden könnte. Dabei wird mit der Get_Tokens() Methode als erstes der Cloud-Benutzer verifiziert. Bei einer erfolgreichen Verifikation sendet der Cloud-Server den Access Token und den Refresh Token als Antwort. In den folgenden Methoden muss der Access Token verwendet werden zur Authentifikation bei jeder Anfrage an den Server, damit sichergestellt wird, dass es sich um einen berechtigten Benutzer handelt. Der Refresh Token wird verwendet, um den Access Token zu aktualisieren, weil dieser nach einer bestimmten Zeit (standartmässig 24 Stunden) abläuft. Auf der Cloud im Time Data Series Service werden die Variablen Metrics genannt. In diesem Fall wäre «auto~LightIntensity» zum Beispiel das Metric der Variable «LightIntensity». Um sich einen Überblick zu verschaffen, welche Metrics auf der Cloud zur Verfügung stehen, werden mit der Methode «Show_All_Metrics(…)» alle vorhandenen Metrics angezeigt. Dann wird der letzte Wert des Metrics «auto~LightIntensity» mit der Methode «Get_Last_Value(uuid, "auto~LightIntensity")» gelesen. Wobei der Übergabeparameter «uuid» der in Kapitel 5.3 erwähnte eindeutige Identifier ist. Schlussendlich werden mit der Methode Get_Last_Values(uuid, "auto~LightIntensity", 10) die zehn letzten Werte des Metrics «auto~LightIntensity» vom Server gelesen. Aus Sicherheitsgründen können über die Cloud nur Werte gelesen werden.

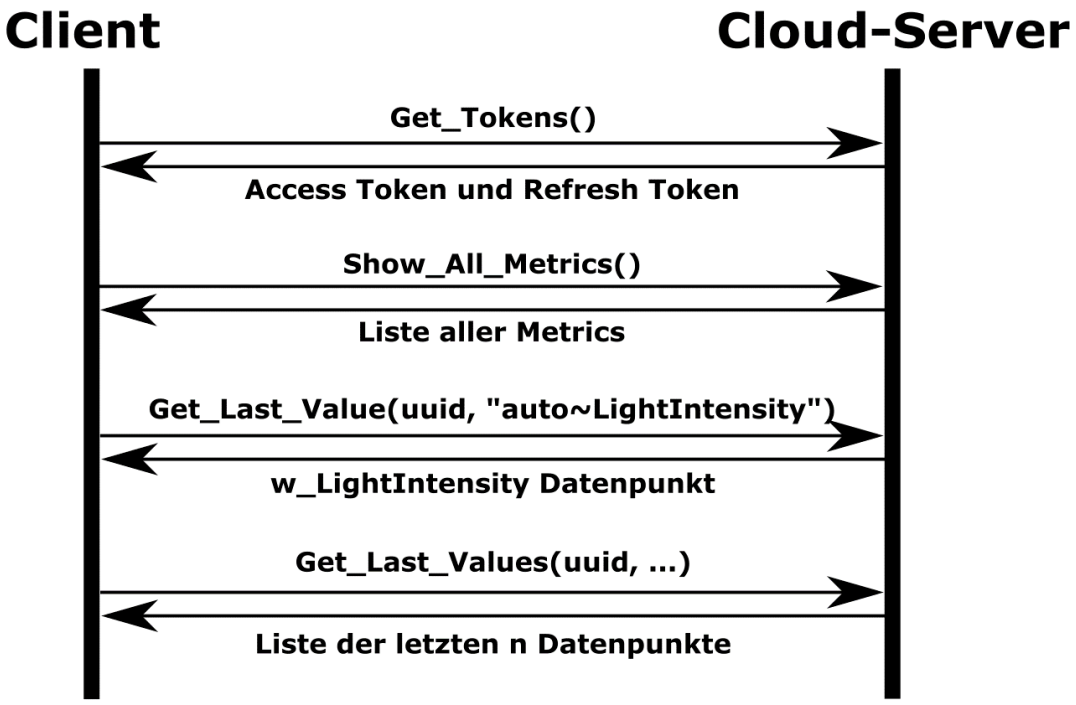



*Abbildung 58: REST_Client Klasse Beispiel*

### 6.2.1. REST Client Klasse

Die eigens erstellte C# REST_Client Klasse beinhaltet die wichtigsten Methoden, für die Authentifikation und das Laden von Daten der Proficloud. In diesem und den folgenden Abschnitten der Klassenbeschreibung wird nur auf die wesentlichen Punkte eingegangen. Der gesamte Code, welcher auch Konsolenausgaben beinhaltet, befindet sich in den technischen Dokumenten unter «REST\Project». Der folgende Codeabschnitt zeigt die Membervariablen und den Konstruktor. Die Variablen «username» und «password» sind der Benutzername und das Passwort um auf dem Proficloud Server zu Authentifizieren. Dies sind die Logindaten, welche auch auf der Proficloud.io Webseite verwendet werden müssen. Die Membervariablen «accessToken» und «refreshToken» beinhalten die Token, welche bei einem Request an den Server für die Authentifizierung benötigt werden. Bewusst wurde keine Membervariable für den UUID angelegt, da es vorkommen kann, dass in einem Proficloud.io Device Management mehrere unterschiedliche Geräte sind. Und wenn die UUID eine Membervariable wäre, könnte mit einem REST_Client Objekt jeweils nur ein Gerät angesprochen werden.

```
private string username;
private string password;
private string accessToken;
private string refreshToken;

public REST_Client(string username, string password)
{
    this.username = username;
    this.password = password;
}
```

Die folgende Zeile Code zeigt, wie der Konstruktor der REST_Client Klasse im Beispielprogram aus Kapitel 6.2.2 aufgerufen wird.

```
REST_Client client = new REST_Client("hans.muster@hslu.ch", "12345678");
```

In den folgenden Kapiteln werden die einzelnen Methoden genauer beschrieben.

#### 6.2.1.1. REST Client Get_Tokens()

Damit beim Lesen der Daten vom Cloud-Server sichergestellt werden kann, dass nicht beliebige Personen die Daten anfordern und einsehen können, werden mit der Get_Tokens() Methode die Authentifizierungs-Token vom Server angefordert und abgespeichert. Nur bei einer erfolgreichen Verifikation retourniert der Server die Token. Diese Tokens werden benötigt, damit der Cloud-Server zu einem späteren Zeitpunkt, wie beispielsweise in Kapitel 6.2.1.3 beim Lesen eines Wertes, mit Sicherheit weiss, dass es sich um einen verifizierten Benutzer handelt. Im Folgenden ist die Signatur der Methode zu sehen. Sie besitzt keine Rückgabe- und Übergabeparameter, jedoch speichert sie den Access Token und den Refresh Token in den jeweiligen Membervariablen.

```
public void Get_Tokens();
```

Als erstes wird das http-Request Packet für die REST-Anfrage an den Cloud-Server vorbereitet. Dabei handelt es sich um eine POST-Methode, wobei der Benutzername und dass Passwort des Proficloud.io Kontos im Request-Body an die URL «https://tsd.proficloud.io/epts/token» gesendet wird. Dafür wird in C# gemäss den folgenden Zeilen Code ein HttpWebRequest Objekt mit dem URL-String «tokenURL» erstellt. Anschliessend wird der Content Type im Header auf «application/json» gesetzt. Damit wird die Antwort des Servers als JSON-Objekt ankommen. (Proficloud.io, 2022) (Ademar, 2022)

```
string tokenURL = "https://tsd.proficloud.io/epts/token";
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(tokenURL);
httpWebRequest.Method = "POST";
httpWebRequest.ContentType = "application/json";
```

Anschliessend wird ein String mit dem Benutzernamen und dem Passwort des Proficloud.io Accounts wie folgt zusammengesetzt: «{"username":"maxmuster@hslu.ch", "password":"12345678"}». Dieser «data» String wird dann mit einem StreamWriter Objekt an den Request-Stream des oben definierten HttpWebRequest Objekts gesendet.

```
string data = "{\"username\":\"" + username + "\"," +
    "\"password\":\"" + password + "\"}";

using (StreamWriter streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
    streamWriter.Write(data);
}
```

Daraufhin wir aus dem «httpWebRequest» Objekt mit der Methode GetResponse() das HttpWebResponse «httpResponse» Objekt erstellt. Mit diesem Objekt wird dann das StreamReader Objekt «streamReader» mit der Methode GetResponseStream() erstellt. Der «streamReader» wird dann verwendet, um die über den Stream ankommende Response des Servers zu lesen und in die String-Variable «result» zu speichern.

```
HttpWebResponse httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
StreamReader streamReader = new StreamReader(httpResponse.GetResponseStream());

string result = streamReader.ReadToEnd();
```

Da der in «result» abgespeicherte String in Form eines JSON-Objekts vorliegt und nebst den beiden benötigten Token noch zusätzliche Informationen wie zum Beispiel die Ablaufzeit der Tokens besitzt, werden nun wie folgt noch die beiden Token aus dem «result» String ausgefiltert und in den jeweiligen Membervariablen abgespeichert. Dabei wird als erstes die Variable «result» an allen vorhandenen Kommas aufgeteilt und in den String-Array «results» abgespeichert. Zur Überprüfung, ob die Token richtig empfangen wurden, und somit die Verifizierung erfolgreich war, wird mit einer if-else-Schleife überprüft, ob die Keywords «access_token» und «refresh_token» an den richtigen Stellen vorhanden sind. Falls dies zutrifft, wird noch nach «"» aufgeteilt, um dann die reinen Token in den dafür vorgesehenen Membervariablen abzuspeichern. Falls die if-Bedinung nicht zutrifft, konnte der übergebene Benutzer nicht verifiziert werden.

```
string[] results = result.Split(',');
if (results[0].Contains("access_token") && results[1].Contains("refresh_token"))
{
    accessToken = results[0].Split('"')[3];
    refreshToken = results[1].Split('"')[3];
    Console.WriteLine("Successfully retreived tokens");
}
else
{
    Console.WriteLine("Error while requesting access and refresh token from cloud");
}
```

Die folgende Zeile Code zeigt, wie die Get_Tokens() Methode im Beispielprogramm aus Kapitel 6.2.2 aufgerufen wird.

```
client.Get_Tokens();
```

In Abbildung 59 ist der Konsolenoutput bei einer erfolgreichen Verifizierung und somit einem erfolgreichen Abspeichern der beiden Token in den zugehörigen Membervariablen zu sehen.

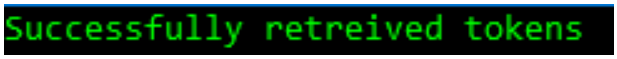



*Abbildung 59: REST Client Klasse Beispiel Output Get_Tokens()*

#### 6.2.1.2. REST Client Get_All_Metrics(…)

Um sich einen Überblick zu verschaffen, welche Metrics auf der Proficloud zur Verfügung stehen, wurde die Methode Get_All_Metrics(…) entwickelt. Diese Methode kann mit dem Befehl «ls» in Linux verglichen werden, mit welchem alle Ordner und Dateien in einem Verzeichnis angezeigt werden können. Im Folgenden ist die Signatur der Methode zu sehen. Sie besitzt die UUID als String als Übergabeparameter und keinen Rückgabeparameter. Am Ende der Ausführung dieser Methode werden alle Metrics-Namen auf der Konsole ausgegeben.

```
public void Show_All_Metrics(string uuid);
```

Verglichen mit den anderen verwendeten GET-Requests ist dies ein wenig anders. Dies, weil mit der REST API ein Datenbank Query, also eigentlich eine Datenbankabfrage gemacht wird, welche dann die vorhandenen Metrics ausgibt. Somit wird in einem ersten Schritt dieser Query im String «query» mit der UUID zusammengesetzt. Die Form dieses Strings sollte wie folgt aussehen:

«SHOW MEASUREMENTS WHERE "uuid" = ‘8d8…689’»

Daraufhin kann mit dem String «query» der eigentlich URL-String «url», wie im folgenden Codeabschnitt gezeigt, erstellt werden. Dieser String sollte dann folgende Form haben:

« https://tsd.proficloud.io/query?q=SHOW MEASUREMENTS WHERE "uuid" = ‘8d8…689’»

```
string query = "SHOW MEASUREMENTS WHERE \"uuid\" = '" + uuid + "'";
string url = "https://tsd.proficloud.io/query?q=" + query;
```

Im nächsten Schritt wird, wie im folgenden Abschnitt gezeigt, das HttpWebRequest Objekt «httpWebRequest» für die Ausführung des GET-Befehls mit dem «url» erstellt. In diesem Objekt wird die HTTP-Methode auf GET gesetzt und der «ContentType» auf «application/json», damit die Antwort (Response) des Servers in Form eines JSON-Objekts gesendet wird. Zum Schluss wird noch das Keyword «Bearer» mit dem «accessToken» dem Request-Header zur Authentifikation beigefügt.

```
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(url);
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "GET";
httpWebRequest.Headers.Add(HttpRequestHeader.Authorization, "Bearer " + accessToken);
```

Um die Antwort des Servers zu empfangen, wird auf den folgenden Zeilen Code zuerst das HttpWebResponse Objekt «httpResponse» mit der GetResponse() Methode des «httpWebRequest» Objekts erstellt. Dann kann mit der GetResponseStream() Methode des «httpResponse» Objekt der StreamReader «streamReader» erstellt werden. Aus diesem «streamReader» wird danach der gesamte ankommende String in die Variable «response» geschrieben. Da dieser String nun in Form eines JSON-Objekts abgespeichert ist, wird mit der Methode Split("\"") der String «response» an allen «"» aufgeteilt und die einzelnen Teile in den String-Array «response_splitted» abgespeichert. Dadurch wird das anschliessende Auslesen der einzelnen Metrics einfacher.

```
HttpWebResponse httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
StreamReader streamReader = new StreamReader(httpResponse.GetResponseStream());

string response = streamReader.ReadToEnd();
string[] response_splitted = response.Split("\"");
```

Weil in dem angekommenen JSON-Objekt ausser den gewünschten Variablen noch weitere Elemente zu finden sind, werden, wie auf den folgenden Zeilen Code gezeigt, auf eine simple Art und Weise die Namen der Metrics im String-Array gesucht und ausgegeben. Dabei wird durch jedes Element von «response_splittet» durchiteriert und falls dieser String «s» das Stichwort «auto~», welches jeweils am Anfang des Metrics-Namen steht, enthält, wird der String «s» auf der Konsole ausgegeben.

```
foreach (string s in response_splitted)
{
    if (s.Contains("auto~"))
    {
        Console.WriteLine(s);
    }
}
```

Der folgende Code zeigt, wie die Show_All_Metrics(…) Methode angewendet werden kann.

```
client.Show_All_Metrics(uuid);
```

In Abbildung 60 ist der Konsolenoutput nach dem Ausführen der Show_All_Metrics(…) Methode gemäss Kapitel 6.2.2 zu sehen. Hierbei ist zu sehen, dass nun alle Metrics und nicht nur jene, die in Kapitel 5.3 selektiert wurden, angezeigt werden.

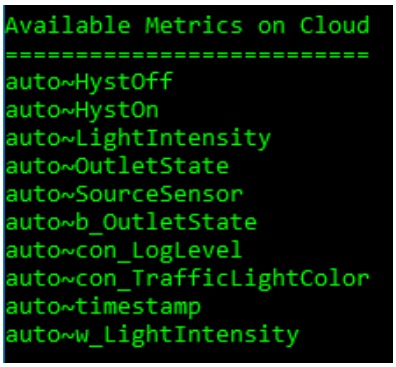



*Abbildung 60: REST Client Klasse Beispiel Output Show_All_Metrics(…)*

#### 6.2.1.3. REST Client Get_Last_Value(…)

Mit der Get_Last_Value(…) Methode kann der letzte vorhandene Wert einer Metrics vom Server geladen werden. Auf der folgenden Zeile Code ist die Signatur der Methode zu sehen. Als Übergabeparameter besitzt sie die UUID des Geräts und den Namen der Metrics als String. Als Rückgabeparameter hat sie ein Integer, welcher den aktuellen Wert beinhaltet.

```
public int Get_Last_Value(string uuid, string metricsName);
```

Zuerst wird der gesamte URL aus der UUID «uuid» und dem Namen der Metrics «metricsName» zusammengesetzt. Dabei resultiert der String «url» welcher beispielsweise folgende Form haben sollte: «https://tsd.proficloud.io/epts/last?uuid=8d8d…4689&timeSeriesName=auto~LightIntensity».
In diesem Fall wird nur der Wert eines Metrics angefragt, es wäre aber auch möglich, der letzte Wert mehrerer Metrics anzufragen. Dafür müssen die Namen der gewünschten Metrics lediglich durch Kommas getrennt aneinandergereiht werden. Jedoch müsste diese Methode noch erweitert werden, da sie im Moment lediglich einen Wert verarbeiten kann. Mit diesem «url» String wird anschliessend wieder das Objekt «httpWebRequest» vom Typ HttpWebRequest erstellt. In diesem Objekt wird dann zum einen der Methodentyp GET gewählt und zum andern der Content Type auf «application/json» gesetzt, dass die Antworten des Servers als JSON-Objekte gesendet werden. Ausserdem wird im «httpWebRequest» Objekt, wie in Kapitel 6.1 erwähnt, die Authentifizierung dem Request Header hinzugefügt. Wiederum handelt es sich dabei um einen String, welcher wie folgt aus dem Keyword «Bearer» und der Membervariable «accessToken» aufgebaut ist: «Bearer eyJhbGde…zuxciOiJSU» (Proficloud.io, 2022) (Ademar, 2022)

```
string url = "https://tsd.proficloud.io/epts/last?uuid="
    + uuid + "&timeSeriesName=" + metricsName;
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(url);
httpWebRequest.Method = "GET";
httpWebRequest.ContentType = "application/json";
httpWebRequest.Headers.Add(HttpRequestHeader.Authorization, "Bearer " + accessToken);
```

Wie bereits im vorherigen Kapitel (6.2.1.3) wird zum Einlesen der Antwort des Servers mit der GetResponse() Methode des Objekts «httpWebRequest» das HttpWebResponse Objekt «httpResponse» erstellt. Mit diesem Objekt wird daraufhin mit der GetResponseStream() Methode das StreamReader-Objekt «streamReader» erstellt. Dann wird der gesamte ankommende String aus dem «streamReader» in den String «response» geschrieben. Da sich in der Variable «response» im Moment noch ein String in Form eines JSON-Objekts befindet, muss der eigentliche Wert noch aus diesem String extrahiert werden und in den Integer «value» abgespeichert werden. Dafür wird der gesamte String an allen «:» aufgeteilt und auf das achte Element ([7]) wird dann die Methode Match(…) aus der Klasse Regex (Regular Expression) angewendet. Diese Methode kann aus einem String, welcher nicht nur Zahlen enthält, ein Integer machen. Nebst dem umzuwandelnden String enthält diese Methode einen String als Übergabeparameter, mit welchem definiert werden kann, welche Zeichen und wie diese extrahiert werden sollen. In diesem Beispiel werden zum Beispiel alle Zahlen zwischen null und neun ([0-9]) beibehalten. Mit dem «+» wird signalisiert, dass nicht nur ein Zeichen, sondern alle Zahlen im vorgegebenen Bereich extrahiert werden sollen. So wird aus dem String «123}]» zum Beispiel der String «123» erstellt. Der Output der Match(…) Methode ist ein String welcher nur Zahlen beinhaltet und somit in den Integer «value» geparst werden kann. Zum Schluss wird der «streamReader» geschlossen und der «value» zurückgegeben. (Microsoft, 2022)

```
HttpWebResponse httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
StreamReader streamReader = new StreamReader(httpResponse.GetResponseStream());
string response = streamReader.ReadToEnd();
int value = int.Parse(Regex.Match(response.Split(":")[7], "[0-9]+").Value);
streamReader.Close();
return value;
```

Die folgende Zeile Code zeigt, wie die Get_Last_Value(…) Methode beispielsweise angewendet werden könnte, um den letzten Wert der Variable LightIntensity auf der Konsole auszugeben. Der String «uuid» könnte zum Beispiel folgendermassen aussehen: 8d8d1a3a-9683-4144-abdf-48a12fc04689.

```
Console.WriteLine("LightIntensity:\t"+client.Get_Last_Value(uuid,"auto~LightIntensity"));
```

In Abbildung 61 ist der Konsolenoutput des Beispielprogramms aus Kapitel 6.2.2 zu sehen, wobei alle vorhandenen Variablen vom Server gelesen und auf der Konsole ausgegeben werden. Da die Variable «OutletState» vom Typ Boolean ist, wird «true» als eine Eins und «false» als eine Null ausgegeben.

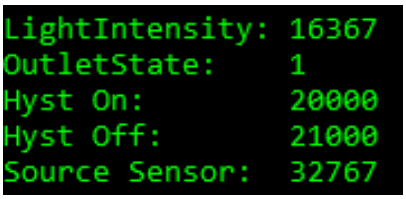



*Abbildung 61: REST Client Klasse Beispiel Output Get_Last_Value(…)*

#### 6.2.1.4. REST Client Get_Last_Values(…)

Damit eine bestimmte Anzahl an Werten einer Metrics auf einmal vom Server geladen werden können und nicht die in Kapitel 6.2.1.3 beschriebene Methode mehrmals ausgeführt werden muss, wurde die Methode Get_Last_Values entwickelt. Mit ihr ist es möglich, alle Werte der letzten X Sekunden eines Metrics vom Server zu laden. Wie auf der folgenden Zeile gezeigt, besitzt die Methode drei Übergabeparameter und einen Rückgabeparameter. Wiederum wird der UUID und der Name des Metrics als String übergeben. Des Weiteren wird der Integer «seconds» zur Angabe der Anzahl Sekunden übergeben. Der Rückgabeparameter ist ein sogenanntes Dictionary mit einem String als Key und einem Integer als Value. Ein Dictionary ist eine Liste von Werten (Value), welche jeweils einem eindeutigen Key zugeordnet sind und auch mit diesem Key abgefragt werden können. Der Dictionary, welcher hier zurückgegeben wird, hat den Zeitstempel des Werts als Key (String) und Wert als Value (Integer).

```
public Dictionary<string, int> Get_Last_Values(string uuid, string metricsName,
    int seconds);
```

Bei der Entwicklung dieser Methode ergab sich bereits früh das Problem, dass zum einen das Hochladen eines Werts auf die Cloud eine gewisse Zeit in Anspruch nimmt und zum andern, dass die Zeit des Laptops, nicht genau mit der Zeit der SPS beziehungsweise der Cloud übereinstimmt. Diese Faktoren führen dazu, dass sich der aktuellste Zeitstempel der Cloud und die Zeit des Laptops bis zu einer Minute unterscheiden. Um dieses Problem zu beheben, wird mit einer simplen Synchronisation gewährleistet, dass die jeweils aktuellsten Werte angefordert werden. Dafür wird in einem ersten Schritt der aktuellste Zeitstempel, welcher auf der Cloud vorhanden ist, mit der Methode Get_Last_Timestamp(…) angefordert und in die String-Variable «currentTimeIso» abgespeichert. Diese Methode wird in Kapitel 6.2.1.5 detaillierter beschrieben. Anschliessend wird das Objekt «currentTime» des Typs DateTime erstellt und die «currentTimeIso» darin abgespeichert. Dabei muss aus unbekannten Gründen die Zeitverschiebung von zwei Stunden mit der Methode AddHours(…) subtrahiert werden, denn die DateTime.Parse(…) parst zum Beispiel die ISO-Zeit der Cloud von «2022-06-06T19:06:48.315Z» in ein DateTime-Objekt mit dem Wert «06.06.2022 21:06:48». Hier ist zu beachten, dass während der Sommerzeit nur eine Stunde subtrahiert werden muss. Daraufhin wird mit der AddSeconds(…) Methode die gewünschte Anzahl einzusehender Sekunden plus einer zusätzlichen, damit kein Wert fehlt, von der «currentTime» subtrahiert und in das DateTime-Objekt «pastTime» gespeichert. Zum Schluss werden die Variablen «currentTime» und «pastTime» noch mit der eigens implementierten Methode Convert_Time_To_ISO(…) in einen String umgewandelt und abgespeichert («currentTimeIso» & «pastTimeIso»), welcher dem ISO Zeitformat entspricht. Auf diese Methode wird in Kapitel 6.2.1.6 genauer eingegangen. Somit sind die beiden benötigten Zeitstempel für die URL im richtigen Format vorbereitet.

```
string currentTimeIso = Get_Last_Timestamp(uuid, metricsName);
DateTime currentTime = DateTime.Parse(currentTimeIso).AddHours(-2);
DateTime pastTime = currentTime.AddSeconds(-seconds-1);

currentTimeIso = Convert_Time_To_ISO(currentTime);
string pastTimeIso = Convert_Time_To_ISO(pastTime);
```

Im nächsten Schritt wird der URL für den Request aus dem Metrics Namen, der aktuellen und der vergangenen Zeit zusammengesetzt und als String in der Variable «url» gespeichert. Der resultierende URL sollte folgende Form haben:

«https://tsd.proficloud.io/epts/data?uuid=8d8…689&timeSeriesName=auto~LightIntensity&fromDate=2022-06-06T21:35:22.102Z&toDate=2022-06-06T21:35:32.102Z»

Daraufhin wird wie bereits in Kapitel 6.2.1.3 mit dem «url» das HttpWebRequest Objekt «httpWebRequest» mit dem Methodentyp GET und dem Content Type «application/json» erstellt. Auch hier wird die Membervariable «accessToken» mit dem Keyword «Bearer» wieder zur Authentifikation dem Request-Header hinzugefügt. (Proficloud.io, 2022) (Ademar, 2022)

```
string url = "https://tsd.proficloud.io/epts/data?uuid=" + uuid + "&timeSeriesName=" +
metricsName +"&fromDate=" + pastTimeIso + "&toDate=" + currentTimeIso;

HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(url);
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "GET";
httpWebRequest.Headers.Add(HttpRequestHeader.Authorization, "Bearer " + accessToken);
```

Anschliessend wird, wie im folgenden Codeabschnitt gezeigt, mit der GetResponse() Methode des «httpWebRequest» Objekts das «httpResponse» Objekt vom Typ HttpWebResponse erstellt. Und dann wird noch der StreamReader «streaReader» mit der Methode GetResponseStream() aus der «httpResponse» erstellt. Die gesamten ankommenden Daten über den «streamReader» werden dann als String in die Variable «response» gespeichert. Da dieser «response» String noch die Form eines JSON-Objekts hat, wird dieser bei allen Kommas aufgetrennt und die einzelnen Teilstrings werden in den String-Array «response_splittet» abgespeichert. Am Ende des folgenden Codes, wird noch der «streamReader» geschlossen.

```
HttpWebResponse httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
StreamReader streamReader = new StreamReader(httpResponse.GetResponseStream());

string response = streamReader.ReadToEnd();
string[] response_splitted = response.Split(",");
streamReader.Close();
```

In einem letzten Schritt werden die Zeitstempel und die dazugehörigen Werte aus dem String-Array «response_splitted» rausgefiltert. Dafür wird durch jedes Element an einer geraden Position im Array durchiteriert und in den geraden Positionen (0, 2, 4, …) befindet sich jeweils der Zeitstempel und in den ungerade Positionen (1, 3, 5, …) der Wert. Wobei der Zeitstempel mit der Substring(…) Methode extrahiert wird und der Wert mit der Match(…) Methode der Regex (Regular Expression) Klasse, um überflüssige Zeichen zu entfernen. Diese Methode kann aus einem String, welcher nicht nur Zahlen enthält, ein Integer machen. Nebst dem umzuwandelnden String enthält diese Methode einen String als Übergabeparameter, mit welchem definiert werden kann, welche Zeichen und wie diese extrahiert werden sollen. In diesem Beispiel werden zum Beispiel alle Zahlen zwischen null und neun ([0-9]) beibehalten. Mit dem «+» wird signalisiert, dass nicht nur ein Zeichen, sondern alle Zahlen im vorgegebenen Bereich extrahiert werden sollen. So wird aus dem String «123}]» zum Beispiel der String «123» erstellt. Der Output der Match(…) Methode ist ein String welcher nur Zahlen beinhaltet und somit in den Integer «val» geparst werden kann. Zu guter Letzt wird in der for-Schleife der Zeitstempel und der Wert dem Dictionary<string, int> «values» zugewiesen und nach der for-Schleife wird der Dictionary «values» zurückgegeben. (Microsoft, 2022)

```
Dictionary<string, int> values = new Dictionary<string, int>();

for (int i = 0; i < response_splitted.Length; i+=2)
{
    string time = response_splitted[i].Substring(response_splitted[i].Length - 25, 24);
    int val = int.Parse(Regex.Match(response_splitted[i + 1], "[0-9]+").Value);
    values.Add(time, val);
}
return values;
```

Der folgende Abschnitt zeigt, wie mit der Get_Last_Values(…) Methode im Beispielprogramm in Kapitel 6.2.2 alle Werte der Metrics «auto~LightIntensity», also der Variable «LightIntensity, von den letzten zehn Sekunden auf der Konsole ausgegeben werden. (Microsoft, 2022)

```
Dictionary<string, int> values = client.Get_Last_Values(uuid,"auto~LightIntensity",10);

foreach (var val in values)
{
    Console.WriteLine("Lightintensity:\t" + val.Value + " at " + val.Key);
}
```

Abbildung 62 zeigt den zu erwartenden Output nach Ausführung des obigen Codes.

```
Lightintensity: 14058 at 2022-06-06T20:12:11.315Z
Lightintensity: 14033 at 2022-06-06T20:12:12.315Z
Lightintensity: 14038 at 2022-06-06T20:12:13.315Z
Lightintensity: 14043 at 2022-06-06T20:12:14.315Z
Lightintensity: 14043 at 2022-06-06T20:12:15.315Z
Lightintensity: 14036 at 2022-06-06T20:12:16.315Z
Lightintensity: 14033 at 2022-06-06T20:12:17.315Z
Lightintensity: 14048 at 2022-06-06T20:12:18.315Z
Lightintensity: 14043 at 2022-06-06T20:12:19.315Z
Lightintensity: 14023 at 2022-06-06T20:12:20.315Z
```

*Abbildung 62: REST Client Klasse Beispiel Output Get_Last_Values(...)*

#### 6.2.1.5. REST Client Get_Last_ Timestamp(…)

Mit der Get_Last_Timestamp(…) Methode kann der letzte vorhandene Zeitstempel einer Metrics vom Server geladen werden. Auf der folgenden Zeile Code ist die Signatur der Methode zu sehen. Als Übergabeparameter besitzt sie die UUID des Geräts als String und den Namen der Metrics als String. Als Rückgabeparameter hat sie ein Integer, welcher den aktuellen Wert beinhaltet.

```
private string Get_Last_Timestamp(string uuid, string metricsName);
```

Zuerst wird der gesamte URL aus der UUID «uuid» und dem Namen der Metrics «metricsName» zusammengesetzt. Dabei resultiert der String «url» welcher beispielsweise folgende Form haben sollte: «https://tsd.proficloud.io/epts/last?uuid=8d8d…4689&timeSeriesName=auto~LightIntensity».
Mit diesem «url» String wird anschliessend wieder das Objekt «httpWebRequest» vom Typ HttpWebRequest erstellt. In diesem Objekt wird dann zum einen der Methodentyp GET gewählt und zum andern der Content Type auf «application/json» gesetzt, dass die Antworten des Servers als JSON-Objekte gesendet werden. Ausserdem wird im «httpWebRequest» Objekt wieder die Authentifizierung mit dem Keyword «Bearer» und dem «accessToken» dem Request Header hinzugefügt. (Proficloud.io, 2022) (Microsoft, 2022) (Ademar, 2022)

```
string url = "https://tsd.proficloud.io/epts/last?uuid="
    + uuid + "&timeSeriesName=" + metricsName;
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(url);
httpWebRequest.Method = "GET";
httpWebRequest.ContentType = "application/json";
httpWebRequest.Headers.Add(HttpRequestHeader.Authorization, "Bearer " + accessToken);
```

Wie bereits zuvor wird zum Einlesen der Antwort des Servers mit der GetResponse() Methode des Objekts «httpWebRequest» das HttpWebResponse Objekt «httpResponse» erstellt. Mit diesem Objekt wird daraufhin mit der GetResponseStream() Methode das StreamReader-Objekt «streamReader» erstellt. Dann wird der gesamte ankommende String aus dem «streamReader» in den String «response» geschrieben. Da sich in der Variable «response» im Moment noch ein String in Form eines JSON-Objekts befindet, muss der eigentliche Zeitstempel aus diesem String extrahiert werden und in den String «timestamp» abgespeichert werden. Dafür wird der gesamte String an allen «{» aufgeteilt und auf das vierte Element ([3]) wird dann die Methode Substring(…) angewendet, um den Zeitstempel zu extrahieren. Zum Schluss wird der «streamReader» geschlossen und der «timestamp» zurückgegeben.

```
HttpWebResponse httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();
StreamReader streamReader = new StreamReader(httpResponse.GetResponseStream());
string response = streamReader.ReadToEnd();
string timestamp = response.Split("{")[3].Substring(13, 24);
streamReader.Close();
return timestamp;
```

Die Get_Last_Timestamp(…) Methode wird bis anhin nur in der Methode Get_Last_Values(…) verwendet. Ausserdem handelt es sich bei dieser Methode um eine private Methode, weshalb diese Methode nur in der REST_Client Klasse und nicht im Beispielprogramm verwendet werden kann. Auf der folgenden Zeile Code wird gezeigt, wie die Methode in Get_Last_Values(…) (Kapitel 6.2.1.4) angewendet wird.

```
string currentTimeIso = Get_Last_Timestamp(uuid, metricsName);
```

#### 6.2.1.6. REST Client Convert_Time_To_ISO(…)

Da der Proficloud.io Server die Zeitstempel gemäss der ISO-8601-Formatierung speichert, die DateTime-Klasse, welche Zeiten in einer anderen Form angibt, aber viele gute Methoden zum Rechnen mit Daten und Zeiten besitzt, wurde die Convert_Time_To_ISO(…) Methode implementiert. Mit dieser Methode können DateTime-Objekte in Strings umgewandelt werden, welche die Zeiten in der ISO-8601-Formatierung repräsentieren. Auf der folgenden Zeile Code ist die Signatur der Methode zu sehen. Wiederum handelt es sich um eine private Methode, die nur in der REST_Client Klasse verwendet werden kann. Der Übergabeparameter ist ein DateTime Objekt und der Rückgabeparameter ein String.

```
private string Convert_Time_To_ISO(DateTime time);
```

Als erstes wird mit den einzelnen Membervariablen der Jahre, Monate etc. des DateTime-Objekts «time» ein String in Form der ISO-8601 Norm zusammengebaut. Dieser String hat dann beispielsweise folgende Form: «2022-6-7T5:55:26.466Z».

```
string timeIso = time.Year + "-" + time.Month + "-" + time.Day + "T" + time.Hour + ":" +
time.Minute + ":" + time.Second + "." + time.Millisecond + "Z";
```

Jedoch ist im ISO-8601 Norm vorgegeben, dass alle Komponenten (Jahr, Monat, …) jeweils zwei Stellen besitzen und die Millisekunden-Angabe sogar drei. Wie am Form-Beispiel im obigen Textabschnitt zu sehen ist, ist diese Vorgabe nach der einfachen Konvertierung in einen String noch nicht gewährleistet. Deshalb wird in einem letzten Schritt für jede einzelne Komponente geschaut, ob noch eine vorgängige Null benötigt wird, damit der String dann zum Beispiel folgende korrekte Form besitzt:

«2022-06-07T05:55:26.466Z».

Zur Überprüfung, ob noch eine Null benötigt wird, wird jede Komponente geschaut, ob sie kleiner als zehn ist. Falls dies der Fall ist, wird eine Null eingefügt. Bei den Millisekunden wird zudem noch überprüft, ob die Zahl kleiner als 100 ist, damit festgestellt werden kann, ob eine oder zwei Nullen benötigt werden. Die Implementation dieses Hinzufügen allfälliger Nullen ist im folgenden Abschnitt gezeigt.

```
if (time.Month < 10) { timeIso = timeIso.Insert(5, "0"); }
if (time.Day < 10) { timeIso = timeIso.Insert(8, "0"); }
if (time.Hour < 10) { timeIso = timeIso.Insert(11, "0"); }
if (time.Minute < 10) { timeIso = timeIso.Insert(14, "0"); }
if (time.Second < 10) { timeIso = timeIso.Insert(17, "0"); }
if (time.Millisecond < 10) { timeIso = timeIso.Insert(20, "00"); }
else if (time.Millisecond < 100) { timeIso = timeIso.Insert(20, "0"); }

return timeIso;
```

Die Convert_Time_To_ISO(…) Methode wird bis anhin nur in der Methode Get_Last_Values(…) verwendet. Auf der folgenden Zeile Code wird gezeigt, wie die Methode in Get_Last_Values(…) (Kapitel 6.2.1.4) angewendet wird.

```
string pastTimeIso = Convert_Time_To_ISO(pastTime);
```

### 6.2.2. REST Client Beispielanwendung

In den technischen Dokumenten unter «REST \Project» befindet sich ein C#-Projekt mit dem Namen «REST\Project\BachelorThesis_REST», welches die REST Client Klasse (REST_Client.cs) und ein Beispielprogramm (Program.cs), gemäss dem Aufbau von Abbildung 58Abbildung 41, beinhaltet. Dieses Beispielprogram enthält Anwendungsbeispiele der Methoden, welche in den Kapiteln 6.2.1.1 bis 6.2.1.4 dokumentiert wurden. In Abbildung 63 ist der zu erwartende Output beim Ausführen dieses Programms gezeigt.

```
Successfully retreived tokens


Available Metrics on Cloud
==========================
auto~HystOff
auto~HystOn
auto~LightIntensity
auto~OutletState
auto~SourceSensor
auto~b_OutletState
auto~con_LogLevel
auto~con_TrafficLightColor
auto~timestamp
auto~w_LightIntensity


LightIntensity: 18179
OutletState:    1
Hyst On:        20001
Hyst Off:       21001
Source Sensor:  32767


Lightintensity: 18376 at 2022-06-08T17:44:23.713Z
Lightintensity: 18344 at 2022-06-08T17:44:24.713Z
Lightintensity: 18314 at 2022-06-08T17:44:25.713Z
Lightintensity: 18319 at 2022-06-08T17:44:26.713Z
Lightintensity: 18302 at 2022-06-08T17:44:27.713Z
Lightintensity: 18258 at 2022-06-08T17:44:28.713Z
Lightintensity: 18231 at 2022-06-08T17:44:29.713Z
Lightintensity: 18228 at 2022-06-08T17:44:30.713Z
Lightintensity: 18186 at 2022-06-08T17:44:31.713Z
Lightintensity: 18179 at 2022-06-08T17:44:32.713Z
```

*Abbildung 63: REST Client Klasse Beispielanwendung Konsolenoutput*

# 7. Schlussdiskussion

In diesem letzten Kapitel ist zum einen ein Fazit enthalten, welches das gesamte Projekt zusammenfasst und festhält, was gut gemacht wurde und wo noch Verbesserungspotential besteht. Zum andern enthält es ein Ausblick, in welchem festgehalten wird, was am bestehenden Projekt noch fehlt und wie mit gewissen Erweiterungen anders verwendet werden könnte. Und zum Schluss noch ein paar dankende Worte für alle Beteiligten dieser Bachelor-Thesis.

## 7.1. Fazit

Zusammengefasst kann gesagt werden, dass ein zweckmässiger Prototyp gebaut wurde, an welchem die Kommunikation zwischen einer SPS und einem Client-Computer über OPC UA sowie über die Proficloud mit REST mit den eigens implementierten Klassen anschaulich gezeigt werden kann. Die beiden dafür erstellten C#-Konsolenanwendungen enthalten die grundlegenden Befehle, um zum Beispiel Daten zu lesen oder zu schreiben. Bereits zu Beginn des Projekts konnten schnell gewisse Erfolge verbucht werden, da bereits früh der Prototyp aufgesetzt war und auch das Lesen und Schreiben über OPC UA funktionierte relativ schnell, trotz mühsamem Zusammensuchen der benötigten Informationen auf verschiedensten Seiten. Andererseits musste beim Erstellen einiger Methoden auch viel erprobt werden, weil im Internet nicht ausreichend Informationen gefunden wurden. Dieses Erproben führt zu Bedenken, dass die Implementationen auf eine andere beziehungsweise bessere Art gemacht werden könnten.

Zudem ist bekannt, dass vor allem OPC UA, ein riesiger Service mit äusserst vielen Möglichkeiten ist. In diesem Projekt wurden jedoch nur einige davon angeschaut und möglichst verständlich umgesetzt. Deshalb kann diese Arbeit noch mit vielen weiteren interessanten Befehlen und Möglichkeiten erweitert werden. Des Weiteren wurde eine möglichst einfache C#-Programmierung angewendet. Deshalb wurden Aspekte wie die Fehlermeldung und Behandlung beinahe komplett weggelassen. Zum Beispiel gäbe es zum Arbeiten mit JSON-Objekten sicher eine schönere Möglichkeit als das Auftrennen an bestimmten Zeichen.

Eine weitere Schwierigkeit dieses Projekts war der Fakt, dass der Code und die Dokumentation von Studierenden oder Kunden als Einstieg in die Programmierung dieser beiden Kommunikationen wiederverwendet werden soll. Da gerade einige Elemente des Codes in mehreren Methoden der identisch ist, bestand deshalb eine gewisse Unsicherheit, ob es sich lohnt, in jedem Kapitel dasselbe zu schreiben. Obwohl dadurch nun einige Informationen mehrmals vorhanden sind, ist jedes Kapitel in sich komplett.

Grundsätzlich kann aber gesagt werden, dass trotz einigen Schwierigkeiten und Unsicherheiten mit diesem Projekt ein sehr guter Grundbaustein für die Wiederverwendung und/oder Weiterentwicklung in diversen Arbeiten zur Integration von SPS in das Industrial Internet of Things gelegt werden konnte.

## 7.2. Ausblick

In diesem Kapitel werden potentielle Ergänzungen zum bestehenden Projekt erläutert.

Wie bereits erwähnt, sollte in beiden C#-Anwendungen eine Fehlermeldung implementiert werden, damit einfacher festgestellt werden kann, wo und wieso ein gewisser Fehler entstand, sodass das Problem einfach behoben oder umgangen werden kann.

Ausserdem wäre es aus Sicherheitsgründen besser, wenn nicht ein Benutzer mit Adminrechten für die Kommunikation über OPC UA und über REST verwendet wird, sondern ein Benutzer, welche nur OPC

UA Rechte hat und einer welcher nur REST Rechte hat. Diese Benutzer können im WBM unter «User Authentication» verwaltet werden. Für den OPC UA Benutzer benötigt man die Rollen CertificateManager, DataViewer und DataChanger. Für den REST Benutzer die Rollen EHmiViewer und EHmiChanger.

Zudem ist die Wiederverwendbarkeit der implementierten Methoden eingeschränkt. Die Set_Value(…) Methode aus der OPC_UA_Client Klasse kann in diesem Moment zum Beispiel nur UInt16 Werte auf dem OPC UA Server setzen. Dies müsste so umgesetzt werden, dass auch weitere Datentypen wie Boolean oder Float geschrieben werden können. Auch der Namespace Index ist zum jetzigen Zeitpunkt «hard coded». Falls auf einer anderen SPS der Namespace mit den benötigten Variablen einen anderen Index besitzt, können diese Methoden nicht mehr verwendet werden. Deshalb sollte dieser Index auch ein Übergabeparameter sein oder der Code könnte allenfalls sogar automatisch den benötigten Namespace Index festlegen.

Eine zusätzliche interessante Erweiterung wäre das Zusammenführen der OPC UA und der REST Client Klasse in einem Projekt. Dies könnte, wie beispielsweise in Abbildung 64 gezeigt, angewendet werden. Dabei gibt es zum einen eine externe SPS, welche sich auf einem anderen Kontinent befinden könnte. Zum andern steht eine weitere SPS lokal in der Fabrik. Nun wäre es dadurch möglich, anhand der Daten der externen SPS, welche via REST von der gezogen werden, die lokale SPS über OPC UA zu steuern. Dies könnte gerade in der Industrie 4.0, bei welcher auch firmenübergreifende Vernetzung entstehen soll, von grosser Bedeutung sein.

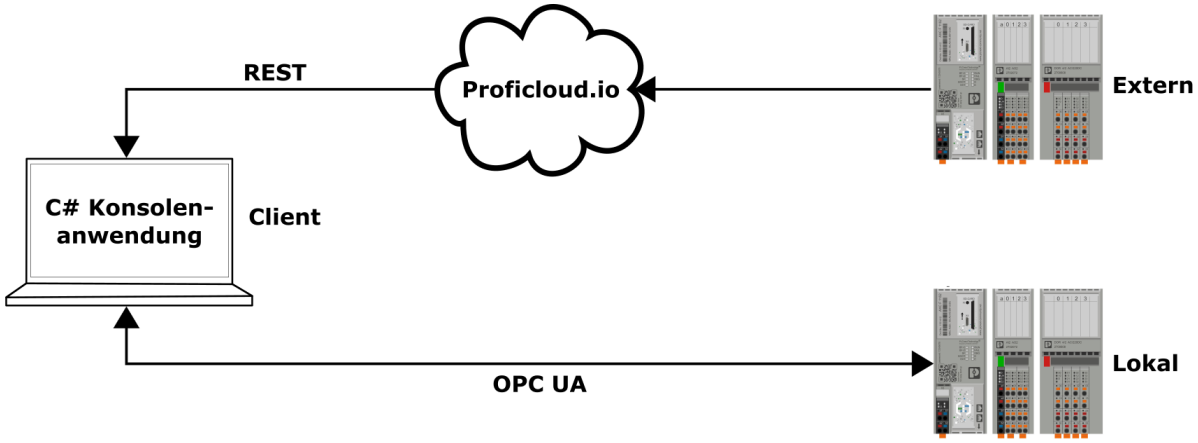



*Abbildung 64: Beispiel Zusammenführung der OPC UA und der REST Client Klasse*

## 7.3. Danksagung

Zu guter Letzt gebührt gewissen Personen, welche unentbehrlich für den Erfolg dieses Projekts waren, ein grosser Dank. Als Erstes möchte ich Dr. Oliver Kasten danken, welcher mich stets tatkräftig unterstützte, sei es bei technischen oder dokumentarischen Anliegen und auch immer ein sehr konstruktives Feedback einbrachte. Des Weiteren möchte ich mich bei den Herren Urs Thönen und Michael Brändli von der Firma Phoenix Contact bedanken. Ohne die Bereitstellung der Klemmen und den wertvollen internen Tipps und Tricks hätte das Projekt wohl nicht in diesem Rahmen durchgeführt werden können.

# 8. Referenzen

## 8.1. Abbildungsverzeichnis

## 8.2. Tabellenverzeichnis



## 8.3. Quellenverzeichnis

Ademar. (15. Mai 2022). *StackOverflow*. Von https://stackoverflow.com/questions/9145667/how-to-post-json-to-a-server-using-c abgerufen

Aro, J. (10. Mai 2022). *ProsysOPC*. Von https://www.prosysopc.com/blog/opc-ua-sessions-subscriptions-and-timeouts/ abgerufen

*Digitec*. (26. März 2022). Von https://www.distrelec.ch/de/komponentengehaeuse-22-5x99x114-5mm-gruen-polyamid-phoenix-contact-2907114/p/30117776 abgerufen

*Distrelec*. (26. März 2022). Von https://www.distrelec.ch/en/ir-photodiode-950nm-mm-vishay-tefd4300f/p/30119197 abgerufen

*HotExamples*. (27. März 2022). Von https://csharp.hotexamples.com/de/examples/Opc.Ua/WriteValue/-/php-writevalue-class-examples.html abgerufen

*HotExamples*. (15. Mai 2022). Von https://csharp.hotexamples.com/de/examples/Opc.Ua.Client/MonitoredItem/-/php-monitoreditem-class-examples.html abgerufen

Industry40tv. (8. Mai 2022). *Youtube*. Von 2022 abgerufen

Industry40tv. (8. Mai 2022). *Youtube*. Von https://www.youtube.com/watch?v=cL5Tq7a1gwo abgerufen

*Ligitek*. (26. März 2022). Von https://www.ligitek.com/de/News_detail/50/# abgerufen

Mercado, J. (29. März 2022). *StackOverflow*. Von https://stackoverflow.com/questions/4982807/c-sharp-insert-a-variable-number-of-spaces-into-a-string-formatting-an-outpu abgerufen

*Microsoft*. (15. Mai 2022). Von https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/async abgerufen

*Microsoft*. (17. Mai 2022). Von https://docs.microsoft.com/en-us/dotnet/api/system.net.httpwebrequest.getresponse?view=net-6.0 abgerufen

*Microsoft*. (16. Mai 2022). Von https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regex.match?view=net-6.0 abgerufen

*Microsoft*. (17. Mai 2022). Von https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-6.0 abgerufen

Moldovean, A. (24. März 2022). *Github*. Von https://github.com/OPCFoundation/UA-.NETStandard-Samples/blob/master/Workshop/DataAccess/Client/WriteValueDlg.cs abgerufen

*OPC Foundation*. (31. Mai 2022). Von https://reference.opcfoundation.org/v104/Core/docs/Part4/5.12.1/ abgerufen

*OPC Foundation*. (31. Mai 2022). Von https://reference.opcfoundation.org/Core/docs/Part9/ abgerufen

OPC Foundation. (5. Mai 2022). *OPC Foundation*. Von https://opcfoundation.org/about/opc-technologies/opc-ua/ abgerufen

*OPC Labs*. (4. Juni 2022). Von https://opclabs.doc-that.com/files/onlinedocs/QuickOpc/Latest/User's%20Guide%20and%20Reference-QuickOPC/OPC%20UA%20Data%20Change%20Filter.html abgerufen

*Phoenix Contact*. (26. März 2022). Von https://www.phoenixcontact.com/de-ch/produkte/steuerung-axc-f-1152-1151412 abgerufen

*Phoenix Contact*. (26. März 2022). Von https://www.phoenixcontact.com/online/portal/de?uri=pxc-oc-itemdetail:pid=2868635 abgerufen

*Phoenix Contact*. (26. März 2022). Von https://www.phoenixcontact.com/de-ch/produkte/io-komponente-axl-f-ai2-ao2-1h-2702072 abgerufen

*Phoenix Contact*. (26. März 2022). Von https://www.phoenixcontact.com/de-ch/produkte/io-komponente-axl-f-dor42-ac220dc-1f-2700608 abgerufen

*PLCnext*. (9. April 2022). Von https://www.plcnext.help/te/PLCnext_Runtime/ESM.htm abgerufen

*PLCnext*. (10. Mai 2022). Von https://www.plcnext.help/te/Service_Components/OPC_UA/OPCUA_server_configuration.htm, abgerufen

*Proficloud*. (21. Mai 2022). Von https://proficloud.io/smart-services/ abgerufen

*Proficloud.io*. (15. Mai 2022). Von https://proficloud.io/technical-documentation/apis-services/time-series-data/rest-api/ abgerufen

*QT*. (4. Juni 2022). Von https://doc-snapshots.qt.io/qtopcua/qopcuamonitoringparameters-datachangefilter.html abgerufen

Singh, R. (18. Mai 2022). *Youtube*. Von https://www.youtube.com/watch?v=zgijGxFLH6E abgerufen

Srocke, D. (31. Mai 2022). *Cloudcomputing Insider*. Von https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/ abgerufen

TopView, E. (31. Mai 2022). *Youtube*. Von https://www.youtube.com/watch?v=bwVkvvzRCSs abgerufen

*Unified Automation*. (10. Mai 2022). Von https://www.unified-automation.com/products/development-tools/uaexpert.html abgerufen

# 9. Anhang

## 9.1. OPC UA Client Konfigurations-File

```
<?xml version="1.0" encoding="utf-8"?>
<ApplicationConfiguration
  xmlns:ua="http://opcfoundation.org/UA/2008/02/Types.xsd"
  xmlns="http://opcfoundation.org/UA/SDK/Configuration.xsd"
  schemaLocation="./Schema/ApplicationConfiguration.xsd">

  <ApplicationName>BAT_Client</ApplicationName>
  <ApplicationUri>urn:localhost:OPCFoundation:BAT_Client</ApplicationUri>
  <ProductUri>http://opcfoundation.org/UA/CoreSampleClient</ProductUri>
  <ApplicationType>Client_1</ApplicationType>

  <SecurityConfiguration>
    <!-- Where the application instance certificate is stored (MachineDefault) -->
    <ApplicationCertificate>
      <StoreType>X509Store</StoreType>
      <StorePath>CurrentUser\My</StorePath>
      <SubjectName>CN=UA Core Sample Client, C=US, S=Arizona, O=OPC Foundation,
       DC=localhost</SubjectName>
    </ApplicationCertificate>
  </SecurityConfiguration>

  <TransportQuotas>
    <OperationTimeout>600000</OperationTimeout>
    <MaxStringLength>1048576</MaxStringLength>
    <MaxByteStringLength>4194304</MaxByteStringLength>
    <MaxArrayLength>65535</MaxArrayLength>
    <MaxMessageSize>4194304</MaxMessageSize>
    <MaxBufferSize>65535</MaxBufferSize>
    <ChannelLifetime>300000</ChannelLifetime>
    <SecurityTokenLifetime>3600000</SecurityTokenLifetime>
  </TransportQuotas>

  <ClientConfiguration>
  </ClientConfiguration>

</ApplicationConfiguration>
```