

Hochschule für Technik und Wirtschaft - Berlin

**Fachbereich 2
Ingenieurwissenschaften - Technik und Leben
im Studiengang Ingenieurinformatik**

Technische Dokumentation

Einrichtung von OPC UA mit open62541
auf einem Raspberry Pi 4

Dieses Dokument dient als Anhang einer Bachelorarbeit,
unterliegt allerdings nicht dessen Sperrvermerk

Thema: Umsetzung der digitalen Durchgängigkeit
im Product Lifecycle Management (PLM)
mittels MBSE und OPC UA

Autor: Klemens Körner

Version vom: 29.04.2021

Inhaltsverzeichnis

Abbildungsverzeichnis	II
1 Einleitung	1
2 Einrichtung des Raspberry Pi	2
2.1 Voreinstellungen	2
2.2 Update und Upgrade	2
2.3 OPC UA Nutzer und Hostname anpassen	3
2.4 SSH	3
2.5 Sicherheit	4
3 Open62541 SDK am Raspberry kompilieren	5
3.1 Erster Server mit OPC UA als Test	6
3.2 Test-Server mit Client verbinden	7
4 Informationsmodellierung eines OPC UA-Servers	10
4.1 FreeOpcUa Modeler	10
5 Aus einem UADataSet.xml den Source-Code erstellen	15
6 Erstellen der Main-Methode und erster Server -Test	16
7 Methoden und Variable mit Logic verbinden	18
7.1 Ändern der Server-Laufzeit-Routine	18
7.2 Variable	18
7.3 Methoden	20
7.4 Modifizierte main.c	21
7.5 Kompilieren und testen	24
8 Fortgeschritten Erweiterungen	26
8.1 Callbacks für Variablen	28
8.2 Methoden mit Multithreading	29
8.3 Browse	31
9 Custom Cmake File	33
10 OPC UA Server als Linux Service	36
11 Git Repository	38

Abbildungsverzeichnis

1	Raspberry Pi Imager	2
2	ccmake-Einstellungen	6
3	Terminal-Ausgabe nach dem Start des OPC UA-Servers	7
4	Erster Start UA Expert	8
5	Daten-Struktur Test-Server in UA Expert	9
6	Start FreeOpcUa Modeler	11
7	Namespace erstellen	11
8	Objekt-Typ anlegen	12
9	Erweitern der Informationen des Motor-Typs	12
10	Erstellen eines Objekts aus einem Objekt-Typ	13
11	UA Expert Aufruf einer Methode	13
12	Auszug aus der gespeicherten xml	14
13	Erfolgreiche Erstellung des Source Codes	15
14	Log Ausgabe der abgefragten Node-ID	17
15	Anzeige Reihenfolge Input-Argumente	21
16	Abruf von Methoden	25
17	Klassendiagramm Painting Staion Belt	27
18	MBSE Informationsmodell	28

1 Einleitung

Diese Technische Dokumentation dient als Anleitung zur Einrichtung eines OPC UA Servers auf einem Raspberry Pi4. Als Basis dient hierbei die Open Source-Variante von OPC UA open62541. Um das Verständnis dieser Dokumentation zu erhöhen, wird empfohlen, die dazugehörige Bachelor-Arbeit zu lesen. Dort werden sehr viele Informationen aus den Übungen von <https://opcua.rocks/> verarbeitet. Diese Seite bietet ausführliche Informationen zu open62541 und den Arbeiten mit offiziellen Informationsmodellen der OPC UA Fundation. Als Grundvoraussetzung wird folgendes gesetzt:

- Raspberry Pi4 mit 2GB Ram
- Unified Automation UA Expert
(www.unified-automation.com/products/development-tools.html)
- Unified Automation UA Modeler
(www.unified-automation.com/products/development-tools.html)
- FreeOpcUA Modeler (github.com/FreeOpcUa/opcua-modeler)
- Raspberry Pi Imager (www.raspberrypi.org/software/)
- Für Windows Putty als SSH Client ([https://www.putty.org/](http://www.putty.org/))
- Cyberduck, um Zugriff via SSH auf Datei-System zu bekommen
(<https://cyberduck.io/download/>)

2 Einrichtung des Raspberry Pi

Als ersten Schritt wollen wir den Raspberry Pi einrichten. Hierfür wird mit dem Raspberry Pi Imager das aktuelle Raspberry OS (nicht Lite oder Full) auf die SD Karte geschrieben (Abbildung 1). Wenn ein schon vorhandener Raspberry benutzt wird, kann man im Kapitel 3 fortfahren. Da ab Kapitel 3 nur noch mit dem Terminal gearbeitet wird, kann man auch mit der OS Lite Version ohne Desktop fortfahren.

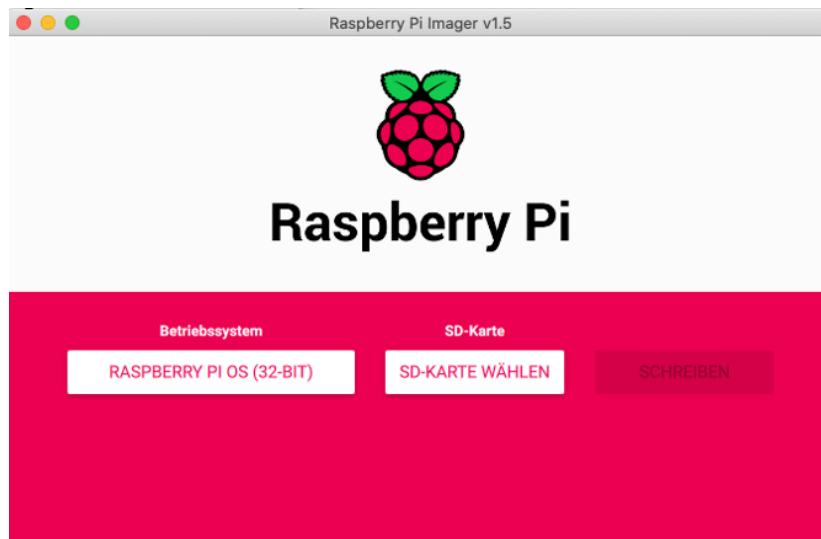


Abbildung 1: Raspberry Pi Imager

Danach kann man die SD Karte in den Raspberry Pi einsetzen und ihn an einen Monitor, eine Tastatur, eine Maus anschließen und ihn schließlich mittels eines USB C-Netzkabels mit Spannung versorgen.

2.1 Voreinstellungen

Nach dem Start wird man durch ein kleines Wizard geführt, um diverse Einstellungen der Sprache und Lokalisierung, das Standard-Passwort des Nutzers pi, Bildschirm und Wlan zu ändern. Falls es nach dem Wizard zu einem Fehler kommt oder das Tastaturlayout nicht stimmt, kann man dies nach dem Neustart in den OS-Einstellungen ändern oder natürlich auch das Wizard beenden und die Einstellungen manuell durchführen.

2.2 Update und Upgrade

Nun wechseln wir im OS in das Terminal und geben den Befehl

```
1 sudo apt update #die Quellen des Paketmanagers aktualisieren  
2 sudo apt upgrade #installierte Pakete auf den aktuellen Stand setzen  
3 #Gegebenenfalls muss man das Installieren der Pakete mit Y Enter  
bestätigen
```

2.3 OPC UA Nutzer und Hostname anpassen

Da es grundsätzlich nie ideal ist, auf einem bestehenden System bekannte Nutzer für den Zugriff von außen zu haben, benötigen wir einen neuen Benutzer. Außerdem wollen wir den hostname des Raspberry an unsere Bedürfnisse anpassen, da dieser für das Auffinden von Geräten im Netzwerk nützlich ist. Hostname ändern:

```

1 hostname #abrufen des aktuellen hostname
2 sudo hostname -b {neuerName} #ändern des hostname
3 hostname #prüfen, ob Änderung erfolgreich
4 sudo grep -lr „{alter hostname}“ /etc/* #prüfen, ob der alte Host noch in
   anderen Konfigurationsdateien verzeichnet ist
5
6 sudo nano /etc/{Dateiname}
7 #nach dem hostname suchen, ändern und mit ctrl x speichern, hier darauf
   achten, dass man keine Dateien ändern muss, wo in der Zeichenkette der
   hostname auftaucht(Bsp. raspi.list, hier findet man http://archive.
   raspberrypi.org/debian/ das natürlich nicht ändern)
8 Außerdem muss man nicht die Keys von SSH ändern, diese erstellen wir
   gleich neu mit folgenden Befehlen
9
10 sudo rm /etc/ssh/ssh_host_* #alte Keys löschen
11 sudo dpkg-reconfigure openssh-server #neue Keys erstellen
12
13 sudo reboot #System-Neustart

```

Benutzer hinzufügen und Pi-Benutzer löschen:

```

1 #Nutzer hinzufügen
2 sudo adduser {USERNAME}
3 #Nutzer-Rechte hinzufügen, am wichtigsten ist sudo
4 sudo usermod -a -G adm,dialout,cdrom,sudo,ssh,audio,video,plugdev,games,
   users,input,netdev,gpio,i2c,spi {USERNAME}
5 #Testen, ob man sich mit dem neuen User als Superuser einloggen kann
6 sudo su - {USERNAME}
7 #den pi-Nutzer löschen
8 sudo pkill -u pi
9 #Nicht wundern, da wir im Desktop als PI-Nutzer eingeloggt sind und
   diesen gerade löschen, werden wir abgemeldet und gefragt, ob wir uns
   als {USERNAME} Nutzer einloggen wollen

```

2.4 SSH

Damit wir einen Fernzugriff auf den Raspberry bekommen, benötigen wir SSH. Dies können wir im Terminal mit folgenden Befehlen aktivieren.

```

1 sudo systemctl start ssh #ssh starten
2 sudo systemctl enable ssh #ssh Autostart nach dem boot Vorgang

```

```
3  
4 systemctl status ssh #Status abfragen  
5 systemctl is-enabled ssh #abfragen, ob Autostart aktiviert  
6  
7 ifconfig #damit sieht man die Einstellungen der Netzwerk Adapter, um sich  
    für später die IP-Adresse (inet) des wlan0-Adapters zu merken.
```

Nun können wir mit einem SSH Client auf den Raspberry zugreifen mit

```
1 ssh {NUTZERNAME}@{IPADRESSE}
```

2.5 Sicherheit

Da davon ausgegangen wird, dass der Raspberry zunächst im lokalen Netz betrieben wird und wir während des Prototypings noch keine erhöhte Sicherheit benötigen, werden keine weiteren Sicherheitseinstellungen vorgenommen. Da aber später möglicherweise eine erhöhte Sicherheit benötigt wird, wird hiermit auf die offizielle Seite von <https://www.raspberrypi.org/documentation/configuration/security.md> mit einer kleinen Guideline zu den Sicherheitseinstellungen, Firewall, SSH, Nutzer usw. verwiesen.

3 Open62541 SDK am Raspberry kompilieren

Die folgenden benötigten Pakete müssen auf dem Raspberry installiert werden. Einige Pakete sind bereits vorinstalliert.

```
1 sudo apt install git build-essential gcc pkg-config cmake python
2
3 # Nützliche Extra-Pakete
4 sudo apt install cmake-curses-gui # ccmake UI
5 sudo apt install libbedtls-dev # Verschlüsselungs-Optionen
6 sudo apt install check libsubunit-dev # unit tests
7 sudo apt install python-sphinx graphviz #für Dokumentationen
8 sudo apt install python-sphinx-rtd-theme # Dokumentstyle-Erweiterung
```

Download des Source Codes

```
1 #Repository Clone
2 git clone https://github.com/open62541/open62541.git ~/open62541
3 #Gehen in den Ordner
4 cd ~/open62541 #Submodule Laden
5 git submodule update --init --recursive
```

Kompilierung vorbereiten mit CCmake

```
1 #Pfad erstellen und in Pfad gehen
2 mkdir ~/open62541/build && cd ~/open62541/build
3 #Öffne ccmake UI aus den Informationen im unteren Pfad
4 ccmake ..
```

Nun öffnet sich die UI. In dieser betätigen wir zunächst die Taste „c“, um die Konfigurations-Seite zu öffnen. Wie in Abbildung 2 zu sehen, ändern wir den CMAKE_BUILD_TYPE gegen RelWithDebInfo (Release Version mit Debug-Informationen), das CMAKE_INSTALL_PREFIX gegen den Installationspfad und den UA_NAMESPACE_ZERO gegen UA_NAMESPACE_FULL (Volle „namespace zero“-Erstellung von den offiziellen XML-Definitionen.). Nachdem wir fertig sind, beenden wir die Einstellungen mit der Taste „c“ und erstellen ein make file mit der Taste „g“. Danach bauen wir den Code mit dem Befehl

```
1 make -j #das kann einige Zeit dauern
2 make install #den Build ins vorher gesetzte Installationsverzeichnis
   verschieben
```

```

BUILD_SHARED_LIBS           *OFF
CLANG_FORMAT_EXE            *CLANG FORMAT EXE-NOTFOUND
CMAKE_BUILD_TYPE             *RelWithDebInfo
CMAKE_INSTALL_PREFIX          */home/{USERNAME}/install
UA_ARCHITECTURE              *posix
UA_BUILD_EXAMPLES             *OFF
UA_BUILD_TOOLS                  *OFF
UA_BUILD_UNIT_TESTS             *OFF
UA_ENABLE_AMALGAMATION          *OFF
UA_ENABLE_DA                      *ON
UA_ENABLE_DISCOVERY                *OFF
UA_ENABLE_DISCOVERY_MULTICAST        *OFF
UA_ENABLE_ENCRYPTION                *OFF
UA_ENABLE_ENCRYPTIONMBEDTLS        *OFF
UA_ENABLE_ENCRYPTION_OPENSSL         *OFF
UA_ENABLE_HISTORIZING               *OFF
UA_ENABLE_METHODCALLS               *ON
UA_ENABLE_NODEMANAGEMENT             *ON
UA_ENABLE_PARSING                      *ON
UA_ENABLE_SUBSCRIPTIONS               *ON
UA_ENABLE_SUBSCRIPTIONSALARMS        *OFF
UA_ENABLE_SUBSCRIPTIONSEVENTS        *OFF
UA_ENABLE_WEBSOCKET_SERVER           *OFF
UA_LOGLEVEL                         *300
UA_MULTITHREADING                   *1
UA_NAMESPACE_ZERO                     *FULL

```

Abbildung 2: ccmake-Einstellungen

Anmerkung: Entgegen der Empfehlung kann es gerade am Anfang Sinnvoll sein nicht den NamespaceFull zu benutzen, sondern Reduced. Dies spart gerade am Anfang beim Ausprobieren viel Zeit beim Kompilieren, wenn man auf einige vordefinierte Typen verzichten kann. Ein Überblick der beinhalteten Typen vom Namespace Reduced findet man hier: <https://github.com/open62541/open62541/blob/master/tools/schema/&Opc.Ua.NodeSet2.Reduced.xml>

3.1 Erster Server mit OPC UA als Test

Nun wollen testen, ob alles funktioniert. Deshalb kopieren wir uns ein Beispiel eines Servers und erstellen daraus einen lauffähigen Server auf dem Raspberry

```

1 mkdir ~/firstTest-Server
2 cp ~/open62541/examples/tutorial_server_firststeps.c ~/firstTestServer/
   firstTestServer.c
3 cd ~/firstTestServer/
4
5 #Bauen des TestServers
6 gcc -std=c99 -fno=1 -I$HOME/install/include -L$HOME/install/lib
   firstTestServer.c -lopen62541 -lmbedtls -lmbedx509 -lmbedcrypto -o
   firstTestServer

```

Dies kann über eine Stunde dauern. Was wurde im gcc-Befehl gemacht:

- std=c99
gcc benutzt den C99-Komplilerer
- fno=1
limitiert die parallel ablaufenden Linking-Jobs auf eins
- -I{PFAD}
fügt den vorhin gebauten Include-Pfad hinzu

- `-L{PFAD}`
fügt den vorhin gebauten Library-Pfad hinzu
- `$Home`
Verweis auf das home-Verzeichnis des aktuellen Nutzers ähnlich wie der Befehl
~/ nur universeller einsetzbar.
- `firstTest-Server.c`
C-Datei, die gebaut werden soll
- `-lopen62541`
open62541 Library
- `-lbedtls -lbedx509 -lbedcrypto`
für Kryptographie
- `-o firstTestServer`
Ausgabe-Datei

3.2 Test-Server mit Client verbinden

Nun können wir den Server testen, indem wir die gerade erstellte Datei ausführen

```
1 ./firstTestServer
```

```
[opcua@opcua01:~/firstTestServer $ ./firstTestServer
[2021-03-03 17:01:07.603 (UTC+0100)] warn/server
permissions.
[2021-03-03 17:01:07.603 (UTC+0100)] info/server
[2021-03-03 17:01:07.603 (UTC+0100)] warn/server
licy. This can leak credentials on the network.
[2021-03-03 17:01:07.603 (UTC+0100)] warn/userland
will be accepted.
[2021-03-03 17:01:07.604 (UTC+0100)] info/network
AccessControl: Unconfigured AccessControl. Users have all
AccessControl: Anonymous login is enabled
Username/Password configured, but no encrypting SecurityPo
AcceptAll Certificate Verification. Any remote certificate
TCP network layer listening on opc.tcp://opcua01:4840/
```

Abbildung 3: Terminal-Ausgabe nach dem Start des OPC UA-Servers

In Abbildung 3 sieht man den Server, der sich gestartet hat sowie die Adresse mit Port, unter der er erreichbar ist. Nun starten wir auf unserem Computer die Software UA Expert und fügen, wie in Abbildung 4 zu sehen ist, den gerade gestarteten Server hinzu.

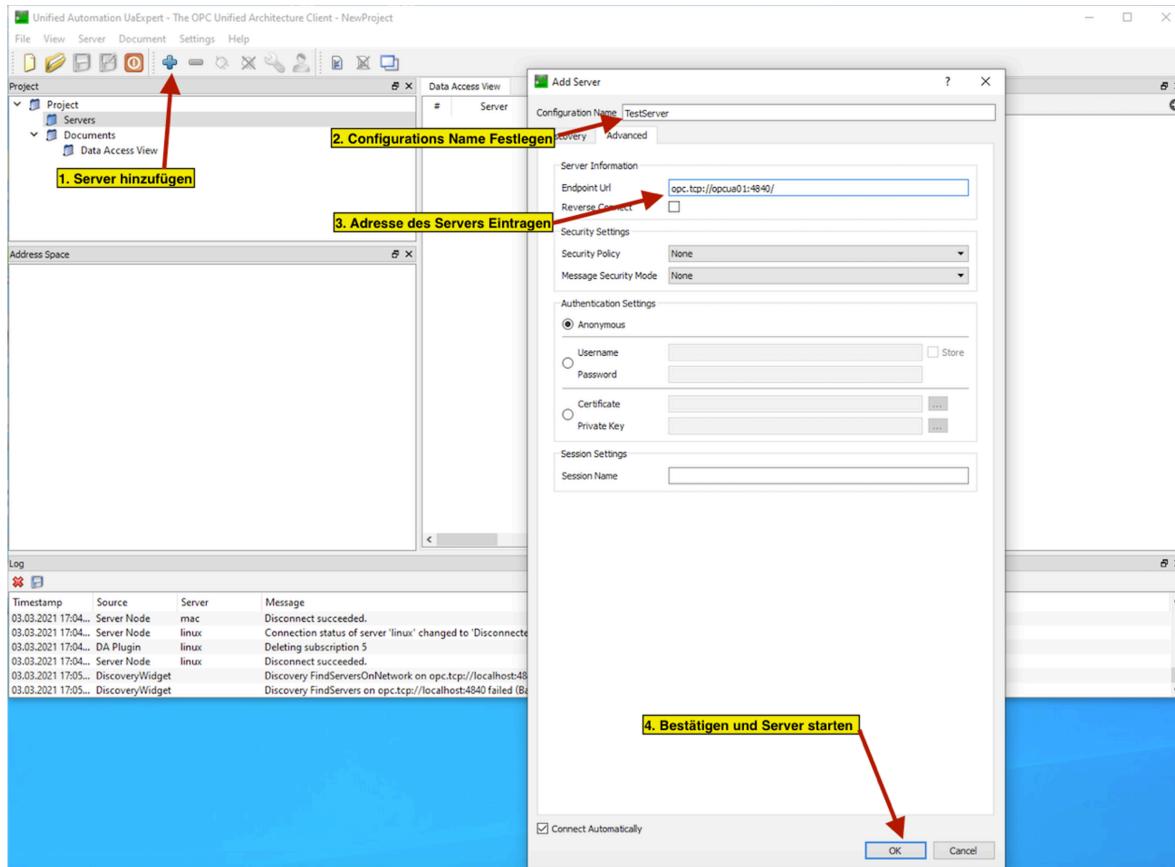


Abbildung 4: Erster Start UA Expert

Danach sieht man in die Datenstruktur des erstellten Test-Servers wie in Abbildung 5

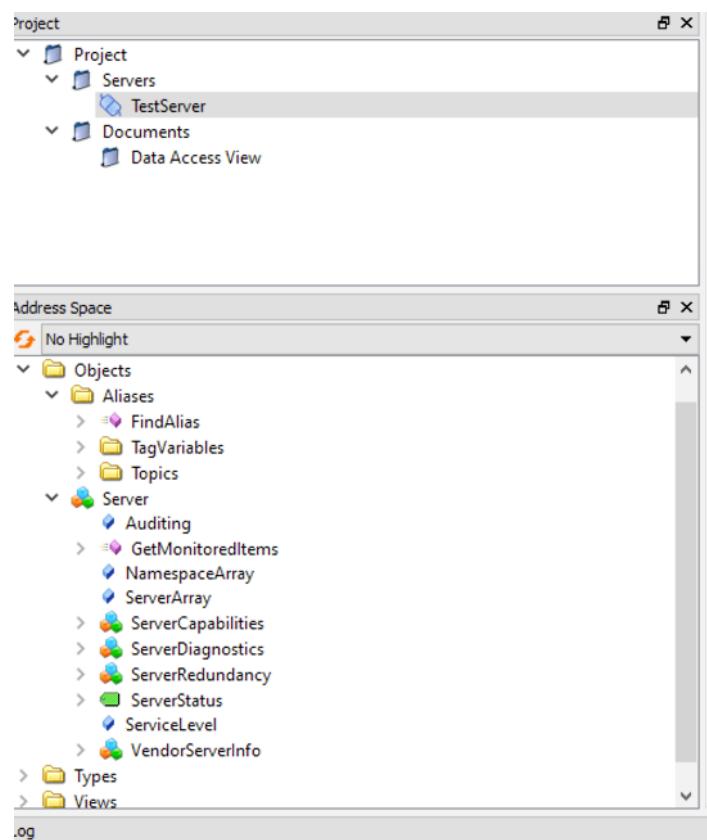


Abbildung 5: Daten-Struktur Test-Server in UA Expert

4 Informationsmodellierung eines OPC UA-Servers

Nachdem jetzt die Grundlagen geschaffen sind, einen lauffähigen OPC UA-Server zu erstellen, wird in diesem Kapitel erklärt, wie man mit Hilfe des FreeOpcUa Modeler ein UANodeSet.xml für seine Server-Struktur erstellt. Dies wird im nächsten Kapitel benötigt, um einen Server mit einem eigenen Nodeset zu bauen. Eine detaillierte Erklärung zu Informationsmodellen und auch noch andere Wege zur Erstellung seiner Informationsmodelle findet man unter <https://opcua.rocks/>

4.1 FreeOpcUa Modeler

Das Programm können wir entweder direkt auf dem Raspberry installieren und ausführen oder auch auf dem Projekt-Rechner. Nachdem wir diesen gestartet haben, läuft im Hintergrund auch schon automatisch ein OPC UA-Server, mit dem man sich verbinden kann. Dieser Server läuft auf dem Open Source Projekt FreeOPC UA. Es soll auch möglich sein, hier direkt einen open62541-Server anzusprechen. Da wir dies in unserem Beispiel nicht benötigten, wurde das nicht getestet. Ein Blick nach dem Start im Terminal zeigt auch hier die generierte opc-Adresse mit Port, wie in Abbildung 6 zu sehen. 0.0.0.0 bedeutet, dass dieser am local Host erreichbar ist. Um sich also mit ihm über einen anderen Rechner im Netzwerk zu verbinden, wird die IP-Adresse des Netzwerk-Adapters benötigt!

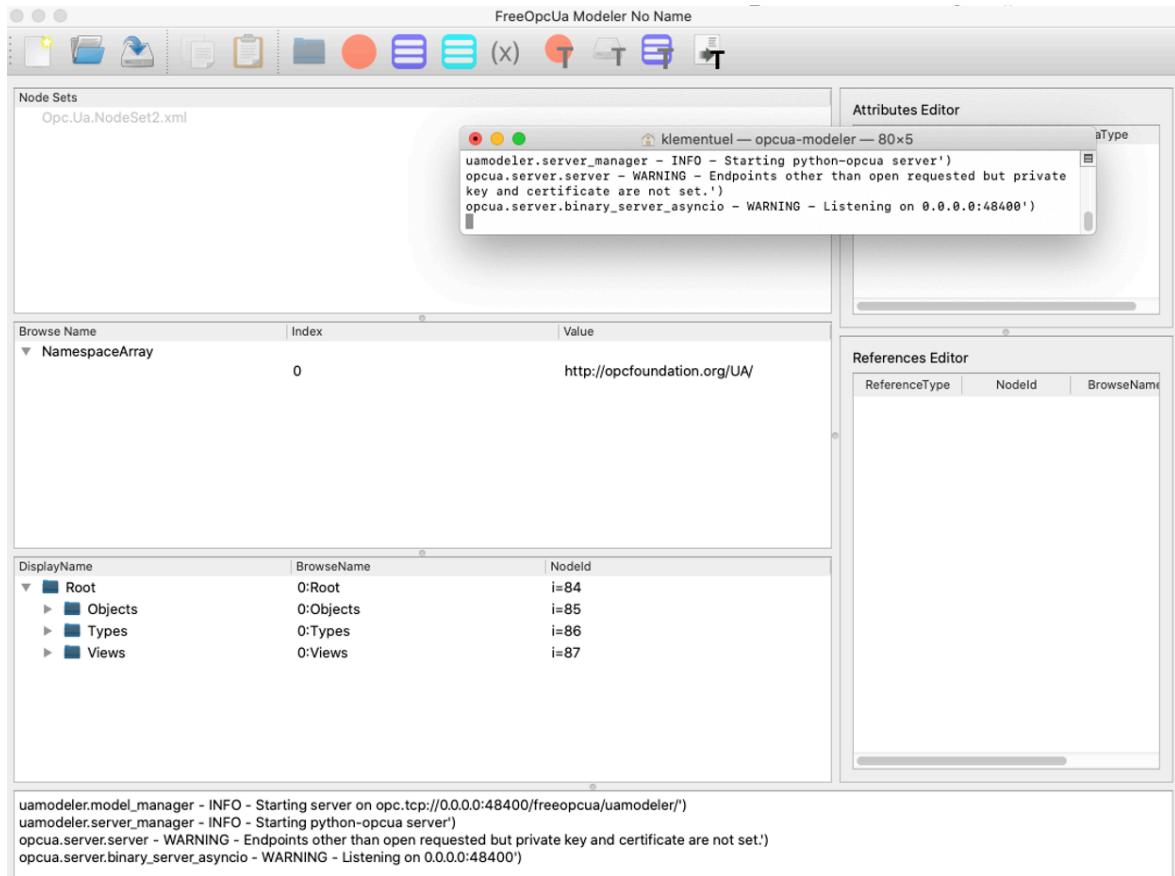


Abbildung 6: Start FreeOpcUa Modeler

Nun beginnen wir mit der Modellierung eines einfachen Beispiels: Wir haben einen Motor, der über das Datenmodell ansteuerbar sein und seinen Status anzeigen soll. Es kann auch sein, dass noch ein Motor hinzukommt. Als erstes benötigen wir einen Namespace, wo wir diese Typen, Objekte später zuordnen. Wie in Abbildung 7 zu sehen, wurde der Namespace 1 mit `http://motoren.test/UA/` erstellt.

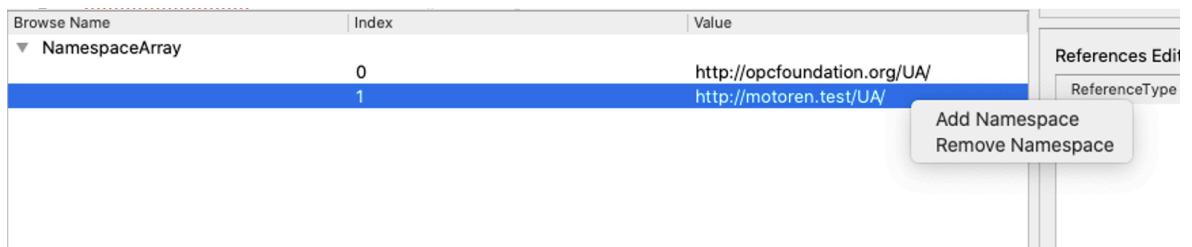


Abbildung 7: Namespace erstellen

Da es also sein kann, dass wir mehrere Motoren benötigen, erstellen wir als Erstes einen Motor-Typ, der ein Objekt eines Motors darstellt und beliebig oft als Objekt wieder verwendet werden kann. In Abbildung 8 ist der Ablauf einer Objekt-Typ-Erstellung zu sehen.

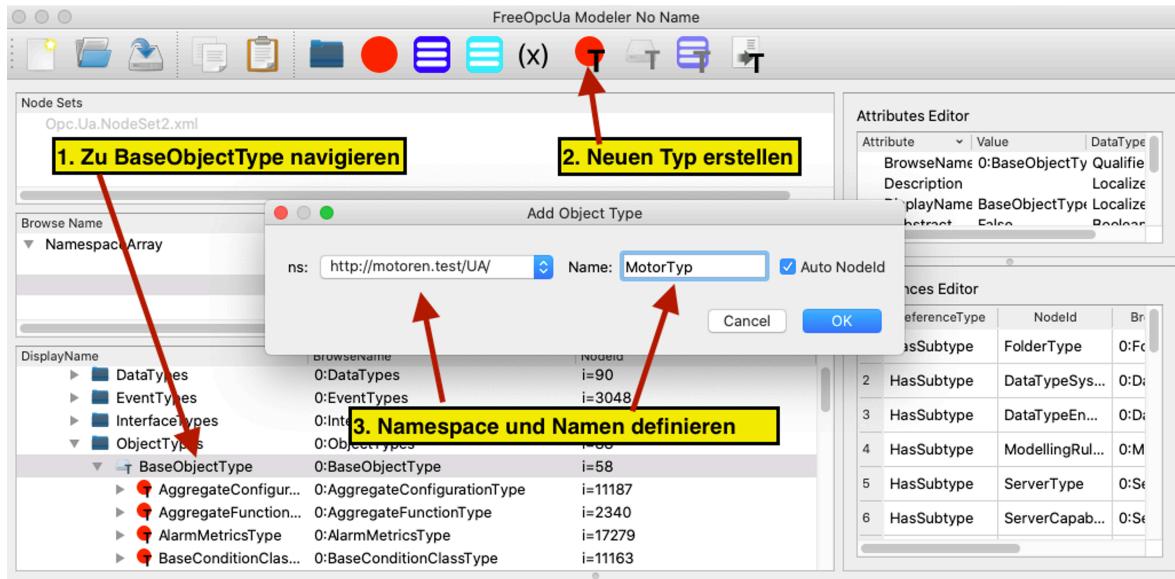


Abbildung 8: Objekt-Typ anlegen

Nun werden dem Objekt-Typ die nötigen Informationen gegeben. Im Beispiel wurden 3 Variablen erstellt, die der Nutzer nur lesen kann (Speed, Running, Direction) und 2 Methoden Control mit Eingangs- und Ausgangs-Variablen und Emergency Stop ohne Variablen, wie in Abbildung 9 zu sehen.

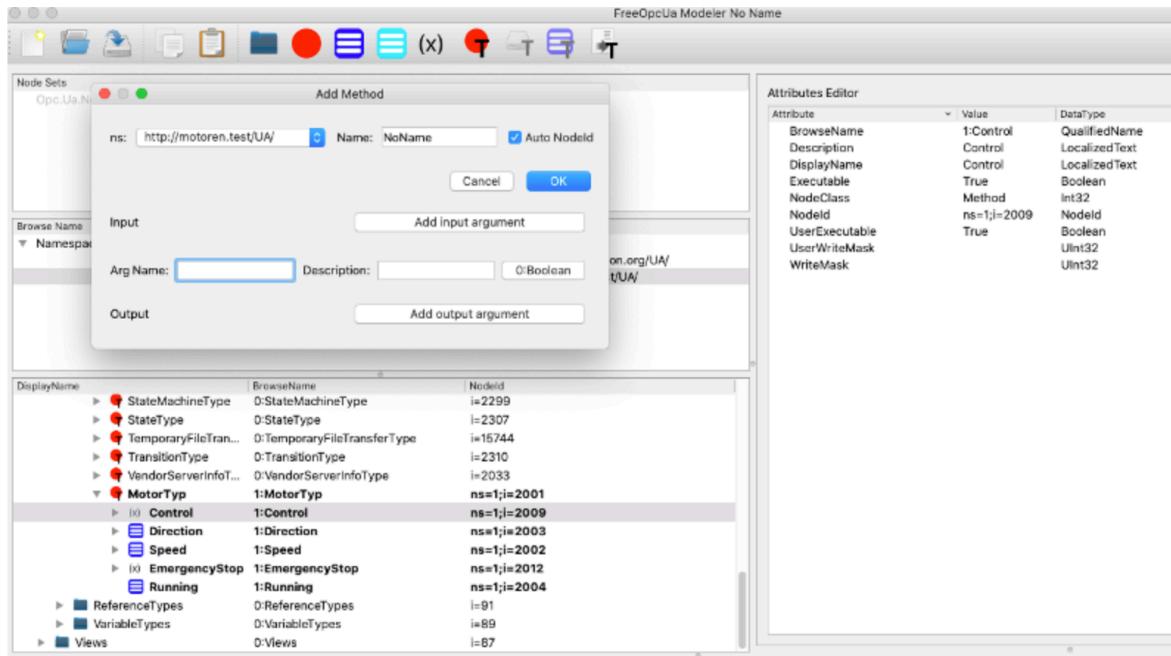


Abbildung 9: Erweitern der Informationen des Motor-Typs

Im nächsten Schritt wird aus dem erstellten Typ ein Objekt erstellt (Abbildung 10).

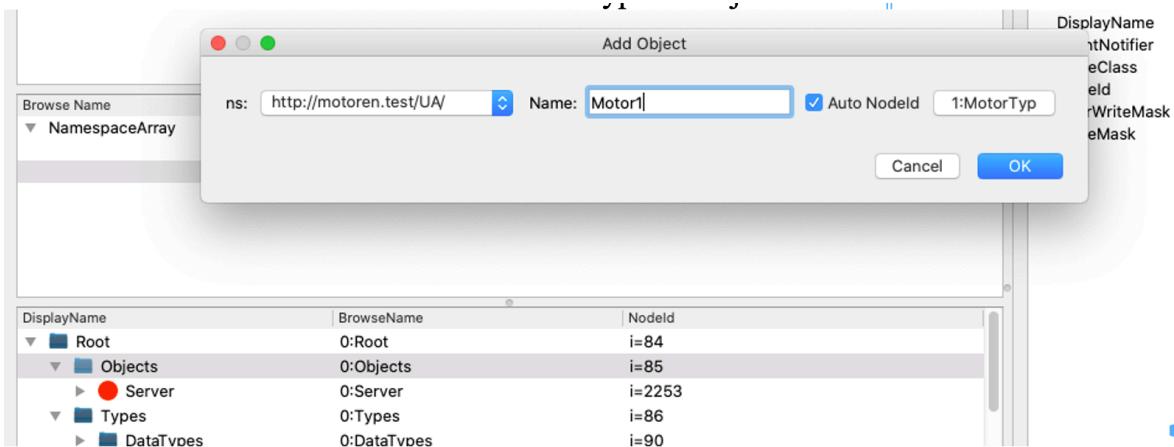


Abbildung 10: Erstellen eines Objekts aus einem Objekt-Typ

Und schon kann im UA Expert Client die Struktur validiert werden, wie Abbildung 11 zeigt.

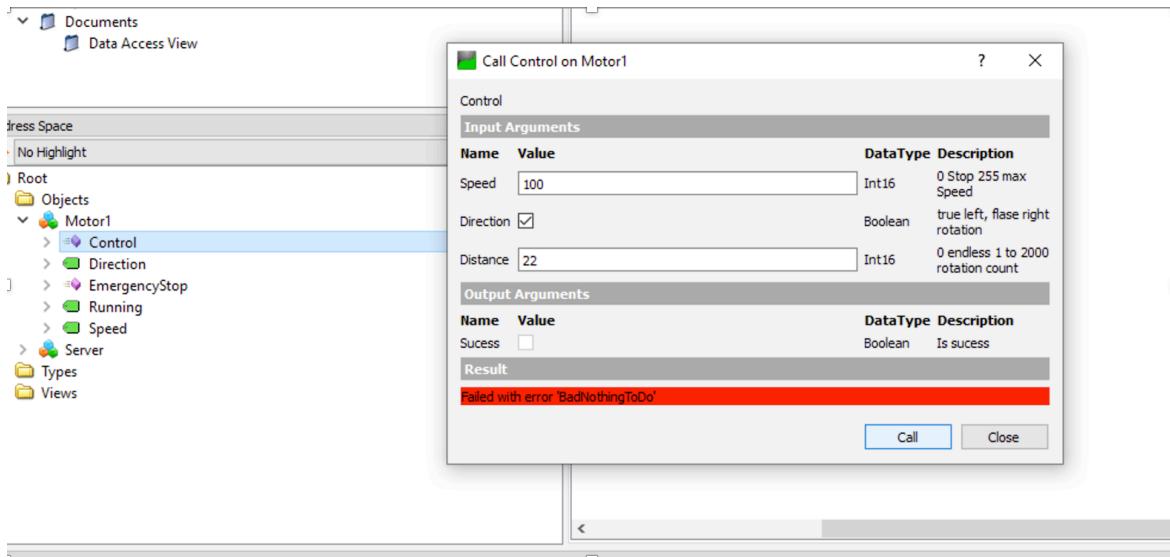


Abbildung 11: UA Expert Aufruf einer Methode

Natürlich führt der Aufruf einer Methode, wie auch zu sehen ist, noch zu einem Fehler, da noch keine Logik in den Methoden steckt. Nun wird das Projekt noch abgespeichert, wodurch sich auch das benötigte UANodeSet File als XML Datei generiert. (Abbildung 12)

```
Name
    MotorSteuerungNodeSet.uamodel
    MotorSteuerungNodeSet.xml

MotorSteuerungNodeSet.xml

<?xml version='1.0' encoding='utf-8'?>
<UANodeSet xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd" xmlns:uax="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema.xsd" xmlns:ns="http://www.w3.org/2001/XMLSchema">
    <NamespaceUris>
        <Uri>http://motoren.test/UA/</Uri>
    </NamespaceUris>
    <Aliases>
        <Alias Alias="HasModellingRule">i=37</Alias>
        <Alias Alias="Boolean">i=1</Alias>
        <Alias Alias="Int16">i=4</Alias>
        <Alias Alias="Organizes">i=35</Alias>
        <Alias Alias="HasTypeDefinition">i=40</Alias>
        <Alias Alias="HasSubtype">i=45</Alias>
        <Alias Alias="HasProperty">i=46</Alias>
        <Alias Alias="HasComponent">i=47</Alias>
        <Alias Alias="Argument">i=296</Alias>
    </Aliases>
    <UAObjectType BrowseName="1:MotorTyp" NodeId="ns=1;i=2001">
        <DisplayName>MotorTyp</DisplayName>
        <Description>MotorTyp</Description>
        <References>
            <Reference IsForward="false" ReferenceType="HasSubtype">i=58</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=2002</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=2003</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=2004</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=2009</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=2012</Reference>
        </References>
    </UAObjectType>
    <UAVariable BrowseName="1:Speed" DataType="Int16" NodeId="ns=1;i=2002" ParentNodeId="ns=1;i=2001">
        <DisplayName>Speed</DisplayName>
        <Description>0 stop 255 max speed</Description>
        <References>
            <Reference IsForward="false" ReferenceType="HasComponent">ns=1;i=2001</Reference>
            <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
        </References>
        <Value>
            <uax:Int16>0</uax:Int16>
        </Value>
    </UAVariable>
    <UAVariable BrowseName="1:Direction" DataType="Boolean" NodeId="ns=1;i=2003" ParentNodeId="ns=1;i=2001">
        <DisplayName>Direction</DisplayName>
        <Description>True Left, False Right rotation</Description>
        <References>
            <Reference IsForward="false" ReferenceType="HasComponent">ns=1;i=2001</Reference>
        </References>
    </UAVariable>

```

Abbildung 12: Auszug aus der gespeicherten xml

Anmerkung: Es ist nicht nötig, hier schon Objekte anzulegen, denn diese kann man auch später erstellen. Wenn man allerdings schon weiß, was man braucht, ist es nützlich, diese gleich zu modellieren. Des Weiteren ist es ebenso möglich, aus einem Typ Objekt-Varianten zu erstellen, wenn man z.B. danach noch Variablen hinzufügt, um einzelne Objekte zu spezialisieren.

5 Aus einem UANodeSet.xml den Source-Code erstellen

Im Grunde ist dies recht einfach, da im Git Repository von open62541 ein python skript hinterlegt ist, das die ganze Arbeit erledigt. Dies kann man also auch direkt am Raspberry erledigen oder auch auf seinem Projekt-Computer. Hierfür muss nun das Skript mit folgenden Zusätzen ausgeführt werden.

```
1 python ~/open62541/tools/nodeset_compiler/nodeset_compiler.py
2 --types-array=UA_TYPES --existing ~/open62541deps/ua-nodeset/Schema/Opc.
   Ua.NodeSet2.xml --xml MotorSteuerungNodeSet.xml MotorSteuerung
```

Kurze Erklärung zu den einzelnen Zusätzen

```
1 # python skript, das ausgeführt werden muss
2 python ~/open62541/tools/nodeset_compiler/nodeset_compiler.py
3 # als was sollen die Typen exportiert werden
4 --types-array=UA_TYPES
5 # auf welchem Schema baut das Ganze auf
6 --existing ~/open62541deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml
7 #Pfad zum erstellten NodeSet2.xml
8 --xml MotorSteuerungNodeSet.xml
9 #Ausgabe-Name, aus diesem werden dann die benötigte c- und h-Datei
   erstellt
10 MotorSteuerung
```

Wenn das Ganze erfolgreich war, sollte es wie in aussehen. Im oberen Teil sieht man die Ausgabe des Skripts, im unteren Teil die im Ordner erstellten c- und h-Dateien.

```
[opcua@opcua01:~/OPCUA/MotorController $ python ~/open62541/tools/nodeset_compiler/nodeset_compiler.py --types-array=UA_TYPES --existing ~/open62541deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml --xml MotorSteuerungNodeSet.xml Motot
Steuerung
INFO:__main__:Preprocessing (existing) /home/opcua/open62541deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml
INFO:__main__:Preprocessing MotorSteuerungNodeSet.xml
INFO:__main__:Generating Code for Backend: open62541
INFO:__main__:NodeSet generation code successfully printed
[opcua@opcua01:~/OPCUA/MotorController $ ls -l
total 48
-rw-r--r-- 1 opcua opcua 14556 Mar  4 12:52 MotorSteuerungNodeSet.xml
-rw-r--r-- 1 opcua opcua 27838 Mar  4 12:59 MototSteuerung.c
-rw-r--r-- 1 opcua opcua    360 Mar  4 12:59 MototSteuerung.h
```

Abbildung 13: Erfolgreiche Erstellung des Source Codes

6 Erstellen der Main-Methode und erster Server -Test

In dem vorhergehenden Kapitel wurde der Source Code aus einem Informationsmodell erstellt. Nun wollen wir das Ganze durch das Erstellen einer Main-Methode ergänzen, die als Startpunkt für den Server dient. Dazu erstellen wir in einem beliebigen Editor mit folgendem Inhalt eine main.c-Datei und speichern diese bei unserem Source Code ab.

```
1 #include <open62541/plugin/log_stdout.h>
2 #include <open62541/server.h>
3 #include <open62541/server_config_default.h>
4 #include <signal.h>
5 #include <stdlib.h>
6 #include "MotorSteuerung.h"
7
8 UA_Boolean running = true;
9
10
11 static void stopHandler(int sign) {
12     UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
13     running = false;
14 }
15
16 int main(int argc, char** argv) {
17     signal(SIGINT, stopHandler);
18     signal(SIGTERM, stopHandler);
19
20     UA_Server *server = UA_Server_new();
21     UA_ServerConfig_setDefault(UA_Server_getConfig(server));
22     UA_StatusCode retval;
23     /* create nodes from nodeset */
24     if(MotorSteuerung(server) != UA_STATUSCODE_GOOD) {
25         UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
26             "Could not add the MotorSteuerung nodeset. "
27             "Check previous output for any error.");
28         retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
29     } else {
30         // Do some additional stuff with the nodes
31         // this will just get the namespace index,
32         //since it is already added to the server
33         UA_UInt16 nsIdx = UA_Server_addNamespace(server, "http://helloWorld.
34         com/UA/");
35         // UA_MOTOR1_RUNNING_BOOL = 2010
36         UA_NodeId testInstanceId = UA_NODEID_NUMERIC(nsIdx, 2010);
37         UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
```

```

37     "Motor1 Bool Running is in ns=%d;id=%d",
38     testInstanceId.namespaceIndex, testInstanceId.identifier.numeric);
39     retval = UA_Server_run(server, &running);
40 }
41 UA_Server_delete(server);
42 return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;

```

Wie man erkennen kann, wird in der Main-Methode ein Server-Objekt mit den Standard-Einstellungen erstellt. Die Variable retval dient dazu, nach dem Beenden den Status-Code des Servers zurückzugeben, als ob der Server wegen eines Fehlers beendet oder absichtlich beendet wurde. Danach fragen wir in einem If-Block ab, ob das Erstellen des neuen MotorSteuerung No-deBlocks erfolgreich war. Wenn nicht, geben wir im Log einen Fehler aus. Wenn das erfolgreich war, können wir entweder gleich zur Zeile mit US_server_run übergehen, oder aber - wie in unserem Fall - wollen wir prüfen, ob eine Node-ID korrekt abgerufen wird. Dies wird in den Log File ausgegeben. Nun können wir wieder mit unserem leicht abgeänderten gcc-Befehl den Server-Code bauen.

```

1 gcc -std=c99 -fno-strict-aliasing -I$HOME/install/include -L$HOME/install/lib main.c
      MotorSteuerung.c -lopen62541 -lbedtls -lbedtlsx509 -lbedtlscrypto -o
      MotorSteuerung

```

Augenmerk sollte hier auf das Hinzufügen der main.c und der implementierten Klassen in diese MotorSteuerung.c gelegt werden. Nachdem der Server erstellt wurde, können wir ihn nun mit

```

1 ./MotorSteuerung

```

starten und prüfen, ob im Log auch unsere Abfrage der Node-ID ausgegeben wird. (Abbildung 14))

```

[2021-03-08 11:55:04.894 (UTC+0100)] warn/server AccessControl: Unconfigured AccessControl. Users have all permissions.
[2021-03-08 11:55:04.894 (UTC+0100)] info/server AccessControl: Anonymous login is enabled
[2021-03-08 11:55:04.894 (UTC+0100)] warn/server Username/Password configured, but no encrypting SecurityPolicy. This can leak credentials on the network.
[2021-03-08 11:55:04.894 (UTC+0100)] warn/userland AccentAll Certificate Verification. Any remote certificate will be accepted.
[2021-03-08 11:55:04.895 (UTC+0100)] info/server Motor1 Bool Running is in ns=2;id=2010
[2021-03-08 11:55:04.895 (UTC+0100)] info/network TCP network layer listening on opc.tcp://opcua01:4840/

```

Abbildung 14: Log Ausgabe der abgefragten Node-ID

Wenn ja, können wir noch prüfen, ob im UA Expert Client alles erwartungsgemäß angezeigt wird

7 Methoden und Variable mit Logic verbinden

Nachdem nun ein funktionierender Server-Code vorliegt, wird mit dem Anlegen der Logik für Variable und Methoden begonnen.

7.1 Ändern der Server-Laufzeit-Routine

Derzeit wird der Server mit

```
1 UA_Server_run(server, &running);
```

gestartet. Dieser Aufruf wird solange durchgeführt, bis die Variable running auf false gesetzt wird. Wenn wir allerdings nun während der Ausführung des Servers noch andere Aktionen ausführen wollen, z.B. das lesen eines Sensors und schreiben einer OPC UA Variable während des Vorgangs müssen wir die Server Routine ein wenig verändern (im Kapitel Unterabschnitt 8.1 kommen wir allerdings zu noch einer anderen Möglichkeit). Dies tun wir indem diese Zeile aus der main.c entfernen und den Code mit Folgendem erweitern.

```
1 retval = UA_Server_run_startup(server);
2 //https://open62541.org/doc/current/server.html#server-lifecycle
3 if(retval != UA_STATUSCODE_GOOD) {
4     UA_Server_delete(server);
5     return retval;
6 }
7 while(running == true)
{
9     //Server Routine
10    //https://open62541.org/doc/current/server.html#server-lifecycle
11    UA_Server_run_iterate(server, true);
12 }
13
14 return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
```

Hier sieht man, dass wir dem Server zunächst die Startup-Routine durchlaufen lassen, prüfen, ob dies erfolgreich war und dann eine while-Schleife starten, in der wir bei jedem Durchlauf einmal die Server-Routine durchlaufen.

7.2 Variable

Um nun z.B. in der while-Schleife jede Sekunde den speed um 1 zu erhöhen und alle 5 Sekunden den Zustand des bool running zu ändern, ergänzen wir den Code folgendermaßen. Wichtig: Hierfür muss noch `#include <time.h>` hinzugefügt werden.

```
1 int timestampM1Running = time(0) + 2;
2 int timestampM1Speed = time(0) + 1;
3 //https://open62541.org/doc/current/types.html
```

```

4 UA_Boolean m1running = false;
5 UA_Int16 m1speed = 0;
6 while(running == true)
7 {
8     //Server Routine https://open62541.org/doc/current/server.html#server-
9     //lifecycle
10    UA_Server_run_iterate(server, true);
11
12    //Multi Use Variables for Update
13    int time_now = time(0);
14    UA_Variant value;
15
16    //Update Variable Running every 2s
17    if(time_now > timestampM1Running)
18    {
19        timestampM1Running = time_now + 2;
20        m1running = m1running ? false : true;
21        //https://open62541.org/doc/current/plugin_log.html
22        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
23        "New Motor 1 Running Bool: %s",m1running ? "True" : "False");
24        //https://open62541.org/doc/current/types.html#generated-data-type-
25        //definitions
26        //https://open62541.org/doc/current/types.html#variant
27        UA_Variant_setScalar(&value, &m1running,&UA_TYPES[UA_TYPES_BOOLEAN]);
28
29        //https://open62541.org/doc/current/server.html#reading-and-writing-
30        //node-attributes
31        UA_Server_writeValue(server, UA_NODEID_NUMERIC(nsIdx, 2010), value);
32    }
33
34    //Update Variable Speed every 1s
35    if(time_now > timestampM1Speed)
36    {
37        timestampM1Speed = time_now + 1;
38        m1speed++;
39        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
40        "New M1 Speed int: %d", m1speed);
41        UA_Variant_setScalar(&value, &m1speed, &UA_TYPES[UA_TYPES_INT16]);
42        UA_Server_writeValue(server, UA_NODEID_NUMERIC(nsIdx, 2011), value);
43    }
44
45 return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;

```

Im Code sind auch die Verweise verzweigt, wo man dies in der Dokumentation findet. Wichtig ist zu wissen, dass zum Schreiben einer Variable das Erstellen einer Variante nötig ist. In dieser werden später der gewünschte Wert und die Information des Daten-

typs temporär abgespeichert und zwar mit der Function `UA_Variant_setScalar`. Diese Variante kann man dann mit der Funktion `UA_Server_writeValue` schreiben, die einen Pointer auf dem Server benötigt, die nötige Node-ID und die soeben erstellte Variante.

7.3 Methoden

Methoden benötigen eine Callback-Funktion. Wir erstellen diese vor der Main-Methode exemplarisch für die Methode Control.

```

1 static UA_StatusCode
2 ControlMethodCallback(UA_Server *server,
3 const UA_NodeId *sessionId, void *sessionHandle,
4 const UA_NodeId *methodId, void *methodContext,
5 const UA_NodeId *objectId, void *objectContext,
6 size_t inputSize, const UA_Variant *input,
7 size_t outputSize, UA_Variant *output) {
8     //Handle Input
9     UA_Int16 speed_in = *(UA_Int16*)input[0].data;
10    UA_Int16 distance_in = *(UA_Int16*)input[1].data;
11    UA_Boolean direction_in = *(UA_Boolean*)input[2].data;
12
13    //Handle Direction and set for test the return Value to direction value
14    UA_Boolean ret_val = direction_in;
15    UA_Variant value;
16    UA_Variant_setScalar(&value, &ret_val, &UA_TYPES[UA_TYPES_BOOLEAN]);
17    UA_Server_writeValue(server, UA_NODEID_NUMERIC(nsIdx, 2012), value);
18
19    //Return the Value to Methode Output
20    UA_Variant_setScalarCopy(output, &ret_val, &UA_TYPES[UA_TYPES_BOOLEAN])
21        ;
22
23    //End Methode
24    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Control was called")
25        ;
26    return UA_STATUSCODE_GOOD;
}

```

Um zu sehen, in welcher Reihenfolge die Input-Parameter abzurufen sind, kann man einen Blick in sein Modell werfen. Wie in Abbildung Abbildung 15 zu sehen, kann man im Attributes Editor die Argumente unter Values identifizieren.

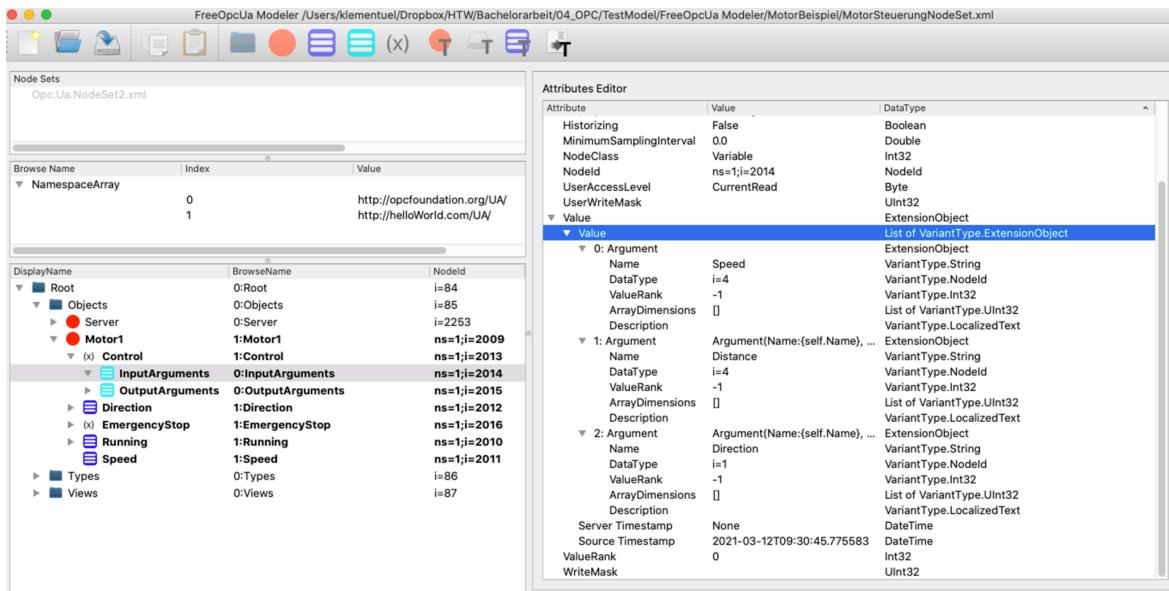


Abbildung 15: Anzeige Reihenfolge Input-Argumente

Damit kann man dann lokale Variable schreiben und seine Methoden Logic durchführen. Wir schreiben in diesem Fall einfach nur den Input Direction direkt in die Server Variable Direction und ändern damit auch je nach dem die output-Variable mit `UA_Variant_setScalarCopy`.

Im nächsten Schritt müssen wir der Methode noch sagen, dass sie diesen ein Callback aufrufen soll. Dies geschieht mit folgender Erweiterung in der Main-Methode.

```

1 //Set Callback for M1 Control https://open62541.org/doc/1.2/server.html#
  method callbacks
2 UA_Server_setMethodNode_callback(server,
3 UA_NODEID_NUMERIC(nsIdx, 2013),
4 &ControlMethodCallback);

```

Diesen Code-Teil kann man nach dem Server-Startup implementieren.

7.4 Modifizierte main.c

Hier findet man noch die gesamten Änderungen der main.c zusammengefasst.

```

1 #include <open62541/plugin/log_stdout.h>
2 #include <open62541/server.h>
3 #include <open62541/server_config_default.h>
4 #include <signal.h>
5 #include <stdlib.h>
6 #include "MotorSteuerung.h"
7
8 #include <time.h>
9
10 UA_Boolean running = true;
11 UA_UInt16 nsIdx; //Namespace "http://helloWorld.com/UA/"

```

```
12
13 static UA_StatusCode
14 ControlMethodCallback(UA_Server *server,
15 const UA_NodeId *sessionId, void *sessionHandle,
16 const UA_NodeId *methodId, void *methodContext,
17 const UA_NodeId *objectId, void *objectContext,
18 size_t inputSize, const UA_Variant *input,
19 size_t outputSize, UA_Variant *output) {
20     //Handle Input
21     UA_Int16 speed_in = *(UA_Int16*)input[0].data;
22     UA_Int16 distance_in = *(UA_Int16*)input[1].data;
23     UA_Boolean direction_in = *(UA_Boolean*)input[2].data;
24
25     //Handle Direction and set for test the return Value to direction value
26     UA_Boolean ret_val = direction_in;
27     UA_Variant value;
28     UA_Variant_setScalar(&value, &ret_val, &UA_TYPES[UA_TYPES_BOOLEAN]);
29     UA_Server_writeValue(server, UA_NODEID_NUMERIC(nsIdx, 2012), value);
30
31     //Return the Value to Methode Output
32     UA_Variant_setScalarCopy(output, &ret_val, &UA_TYPES[UA_TYPES_BOOLEAN])
33     ;
34
35     //End Methode
36     UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Control was called")
37     ;
38
39     static void stopHandler(int sign) {
40         UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
41         running = false;
42     }
43
44 int main(int argc, char** argv) {
45     signal(SIGINT, stopHandler);
46     signal(SIGTERM, stopHandler);
47
48     UA_Server *server = UA_Server_new();
49     UA_ServerConfig_setDefault(UA_Server_getConfig(server));
50     UA_StatusCode retval;
51     /* create nodes from nodeset */
52     if(MotorSteuerung(server) != UA_STATUSCODE_GOOD) {
53         UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the
54         Motor-Steuerung nodeset. "
55         "Check previous output for any error.");
56         retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
```

```
56 } else {
57     // Do some additional stuff with the nodes
58     // this will just get the namespace index, since it is already added
59     // to the server
60     nsIdx = UA_Server_addNamespace(server, "http://helloWorld.com/UA/");
61     // UA_MOTOR1_RUNNING_BOOL = 2010
62     UA_NodeId testInstanceId = UA_NODEID_NUMERIC(nsIdx, 2010);
63     UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Motor1 Bool
64     Running is in ns=%d;id=%d",
65     testInstanceId.namespaceIndex, testInstanceId.identifier.numeric);
66
67 }
68
69 retval = UA_Server_run_startup(server);
70 //Set Callback for M1 Control https://open62541.org/doc/1.2/server.html
71 //method-callbacks
72 UA_Server_setMethodNode_callback(server,
73     UA_NODEID_NUMERIC(nsIdx, 2013),
74     &ControlMethodCallback);
75
76 //https://open62541.org/doc/current/server.html#server-lifecycle
77 if(retval != UA_STATUSCODE_GOOD) {
78     UA_Server_delete(server);
79     return retval;
80 }
81
82 int timestampM1Running = time(0) + 2;
83 int timestampM1Speed = time(0) + 1;
84 //https://open62541.org/doc/current/types.html
85 UA_Boolean m1running = false;
86 UA_Int16 m1speed = 0;
87 while(running == true)
88 {
89     //Server Routine https://open62541.org/doc/current/server.html#server
90     //lifecycle
91     UA_Server_run_iterate(server, true);
92
93     //Multi Use Variables for Update
94     int time_now = time(0);
95     UA_Variant value;
96
97     //Update Variable Running every 2s
98     if(time_now > timestampM1Running) {
99         timestampM1Running = time_now + 2;
100        m1running = m1running ? false : true;
101        //https://open62541.org/doc/current/plugin_log.html
```

```
98     UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New Motor 1
99     Running Bool: %s",
100    m1running ? "True" : "False");
101    //https://open62541.org/doc/current/types.html#generated-data-type-
102    definitions
103    //https://open62541.org/doc/current/types.html#variant
104    UA_Variant_setScalar(&value, &m1running, &UA_TYPES[
105        UA_TYPES_BOOLEAN]);
106
107    //https://open62541.org/doc/current/server.html#reading-and-writing
108    -node-attributes
109    UA_Server_writeValue(server, UA_NODEID_NUMERIC(nsIdx, 2010), value)
110    ;
111    }
112
113    //Update Variable Speed every 1s
114    if(time_now > timestampM1Speed) {
115        timestampM1Speed = time_now + 1;
116        m1speed++;
117        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New M1 Speed
118        int: %d",
119        m1speed);
120        UA_Variant_setScalar(&value, &m1speed, &UA_TYPES[UA_TYPES_INT16]);
121        UA_Server_writeValue(server, UA_NODEID_NUMERIC(nsIdx, 2011), value)
122        ;
123    }
124
125    }
126
127
128
129    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
130 }
```

7.5 Kompilieren und testen

Nun kann der Code wie im Kapitel 6 kompiliert und getestet werden.

In UA Expert kann man sich die Variablen in das Data Access View-Fenster ziehen und beobachten, wie sich der Wert ändert. Die Methode kann mit einem rechten Mausklick aufgerufen werden, danach kann man die Input-Parameter setzen und mit Call den Aufruf erledigen. Hier sind nun die Änderungen der Direction und Success zu beobachten. In Abbildung 16 sieht man einen erfolgreichen Methoden-Aufruf.

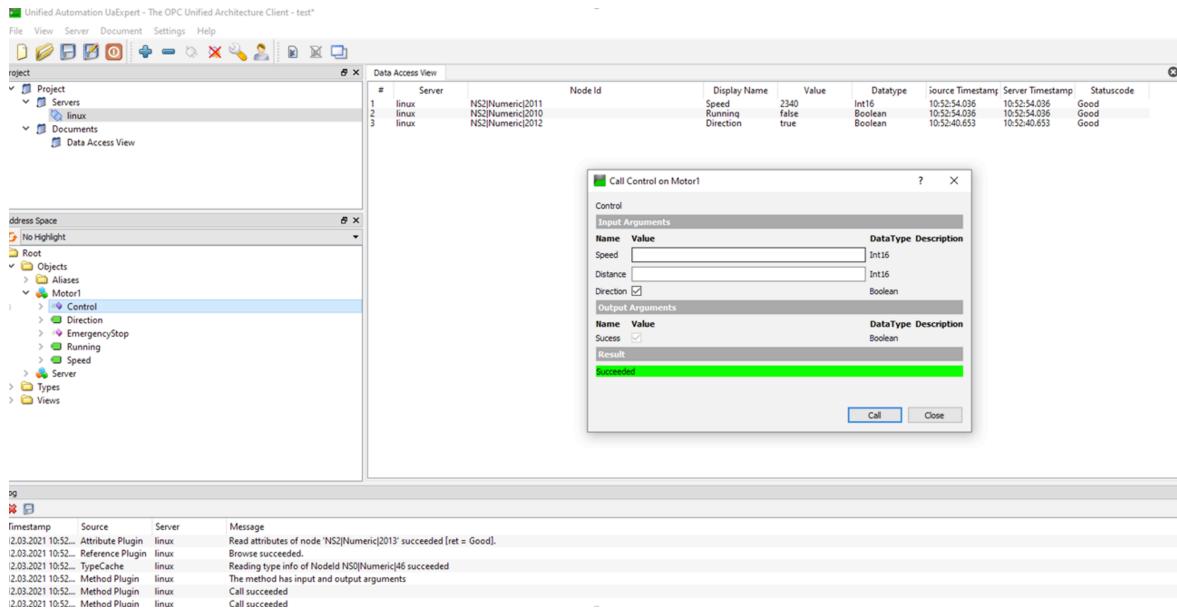


Abbildung 16: Abruf von Methoden

8 Fortgeschritten Erweiterungen

In diesem Kapitel wollen wir nur auf weitere Möglichkeiten eingehen, wie man mit Variablen vor oder nach dem Lesen bzw. Schreiben Aktionen auslöst, wie man Methoden, die durch lange Operationen den Server blockieren würden, asynchron bekommt oder die Möglichkeit die Schematische Struktur seines Servers zu durchsuchen ohne die NodeID's einzelner Objekte zu kennen. Hierzu dient uns ein Beispiel aus der Bachelor Arbeit des Umsetzungsteils des PaintingStationBelt. Dieses Beispiel ist vom Aufbau schon ein wenig komplexer in seiner Strukturiert um allerdings auch die Übersicht zu vereinfachen wie in Abbildung 17 zu sehen.

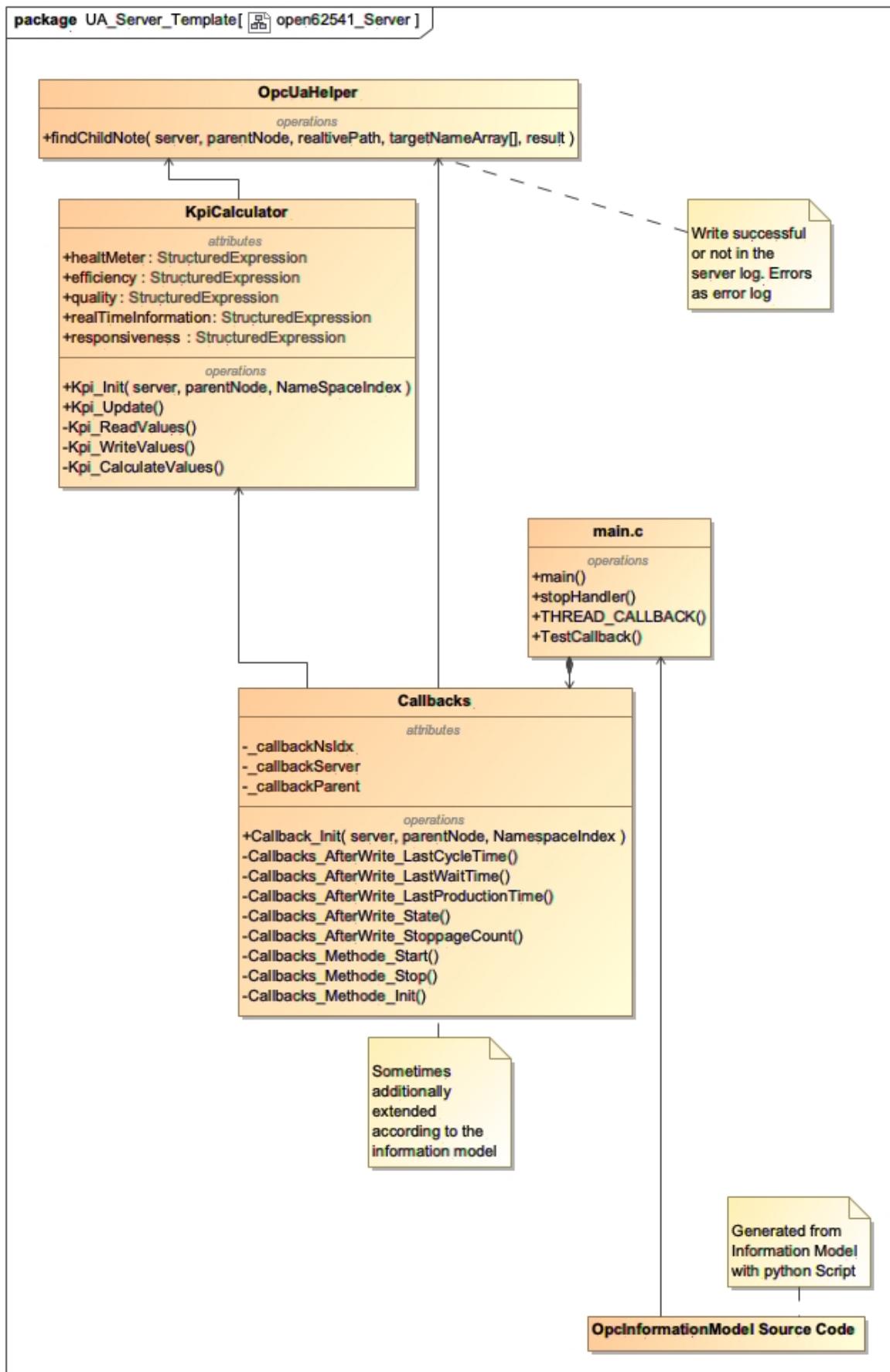


Abbildung 17: Klassendiagramm Painting Staion Belt

In Abbildung 18 sieht man dazu auch eine Übersicht des OPC UA Informationsmodells, allerdings nicht in der empfohlenen Schematischen Modellierung der OPC UA Fundation, da dies in der Bachelorarbeit mit SysML modelliert wurde.

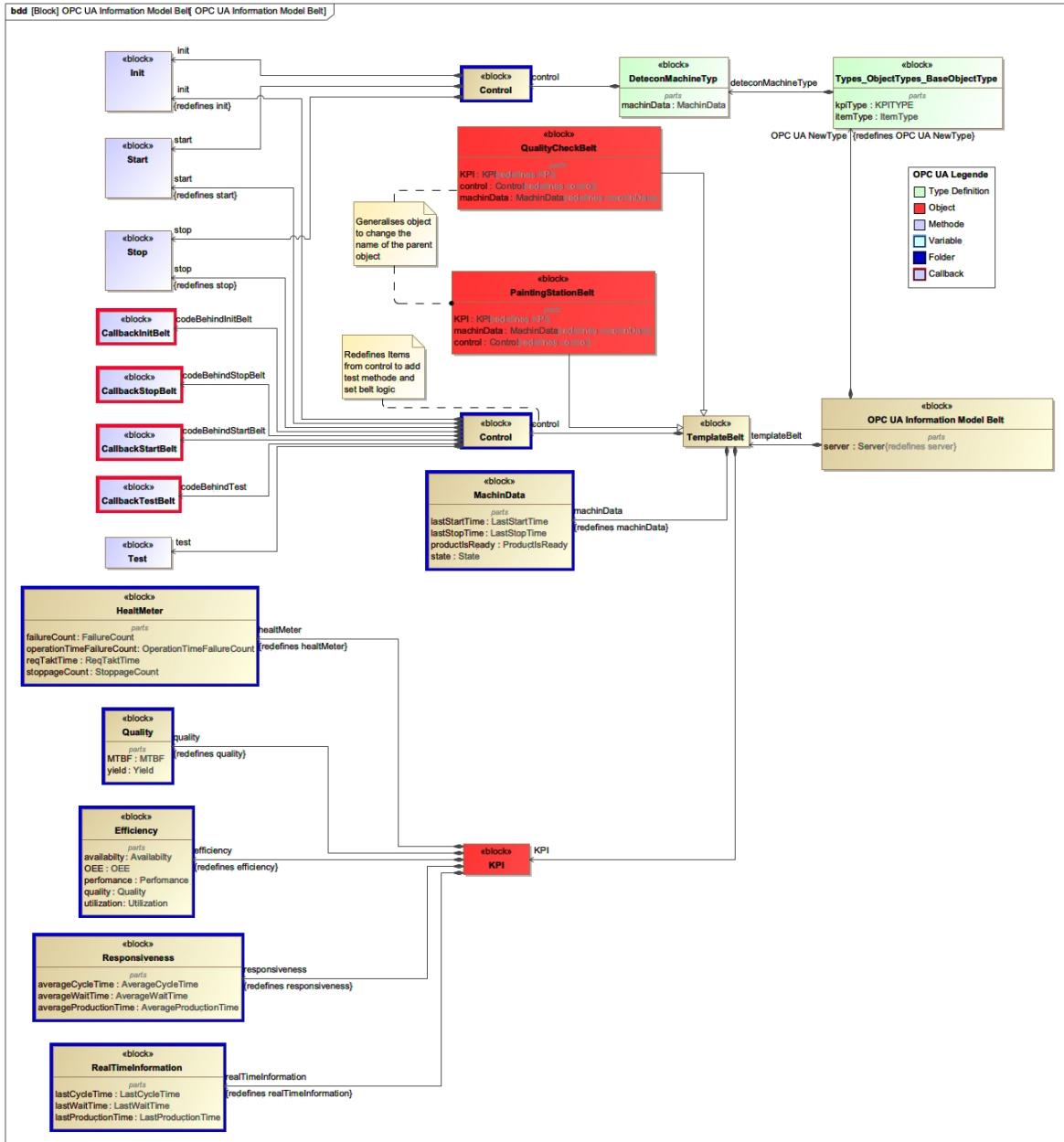


Abbildung 18: MBSE Informationsmodell

8.1 Callbacks für Variablen

Wie schon bei den Methoden ist es möglich Callbacks für Variablen zu setzen. Entweder das diese eine Aktion auslösen, bevor sie gelenkt werden, oder nachdem sie beschrieben wurden. Im Beispiel wollen wir nun das nach dem schreiben der Variable, der Status einer Server Log Nachricht ausgibt. Dazu schreiben wir als erstes eine passende Funktion

```
1 void
```

```

2     Callbacks_AfterWrite_State(UA_Server *server,
3     const UA_NodeId *sessionId, void *sessionContext,
4     const UA_NodeId *nodeId, void *nodeContext,
5     const UA_NumericRange *range, const UA_DataValue *data)
6     {
7         UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "State has Changed"
8     );
}

```

nun müssen wir dem Server noch das Callback zuordnen

```

1  /*Create and set the appropriate methods in the Callback variable.
   Important if you don't assign a function you have to set NULL,
   otherwise unexpected pointer errors can occur.*/
2  UA_ValueCallback callback;
3  callback.onRead = NULL;
4  callback.onWrite = Callbacks_AfterWrite_State;
//Set the callback variable to the NodeID
5  UA_Server_setVariableNode_valueCallback(server,
6  variable_State,
7  callback);

```

Damit ist das ganze schon erledigt und einsatzbereit.

8.2 Methoden mit Multithreading

Man kann sich sicher Vorstellen, dass es auch Callbacks gibt die den Server blockieren können, da diese eine gewisse Zeit benötigen um zu einem Ergebnis zu kommen. Wie z.B. in meiner Bachelorarbeit. Hier wurden Python Scripts von Methoden aufgerufen, die z.B. im Fall von dem PaintingStationBelt gut 10s für einen Aufruf benötigen. In dieser Zeit kann ein Client keine Variablen schreiben oder lesen, das natürlich dann zu Fehlern führen würde. Zum Glück gibt es bei open62541 SDK die Option Multithreading zu aktivieren. Dazu müsst ihr das SDK mit der entsprechenden Option im ccmake unter `UA_Multithreading = 100` setzen und neu bauen. Der Rest bleibt gleich wie in Abschnitt 3 beschrieben

Beginnend im main.c File musste man dann folgendes erweitern in der #Include Sektion:

```

1  #ifndef WIN32
2  #include <pthread.h>
3  #define THREAD_HANDLE pthread_t
4  #define THREAD_CREATE(handle, callback) pthread_create(&handle, NULL,
5  callback, NULL)
6  #define THREAD_JOIN(handle) pthread_join(handle, NULL)
7  #define THREAD_CALLBACK(name) static void * name(void *_)
8  #else
# include <windows.h>

```

```

9 #define THREAD_HANDLE HANDLE
10 #define THREAD_CREATE(handle, callback) { handle = CreateThread( NULL,
11   0, callback, NULL, 0, NULL); }
12 #define THREAD_JOIN(handle) WaitForSingleObject(handle, INFINITE)
13 #define THREAD_CALLBACK(name) static DWORD WINAPI name( LPVOID lpParam
14   )
15 #endif

```

Und mit zwei Funktionen vor der main Funktion erweitern.

```

1 THREAD_CALLBACK(ThreadWorker) {
2   while(running) {
3     UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
4     "Try to dequeue an async operation");
5     const UA_AsyncOperationRequest* request = NULL;
6     void *context = NULL;
7     UA_AsyncOperationType type;
8     if(UA_Server_getAsyncOperationNonBlocking(server, &type, &request, &
9       context, NULL) == true) {
10       UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "
11         AsyncMethod_Testing: Got entry: OKAY");
12       UA_CallMethodResult response = UA_Server_call(server, &request->
13         callMethodRequest);
14       UA_Server_setAsyncOperationResult(server, (
15         UA_AsyncOperationResponse*)&response,
16         context);
17       UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "
18         AsyncMethod_Testing: Call done: OKAY");
19       UA_CallMethodResult_clear(&response);
20     } else {
21       /* not a good style, but done for simplicity :-) */
22       sleep(5);
23     }
24   }
25   return 0;
26 }
27
28 /* This callback will be called when a new entry is added to the
   Callrequest queue */
29 static void
30 TestCallback(UA_Server *server) {
31   UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
32   "Dispatched an async method");
33 }

```

Das war schon der schwierigste Teil, im nächsten Schritt fügen wir in der Callback.c Datei noch den Eintrag hinzu, dass dieses Callback Asynchron ausgeführt werden muss.

```

1 UA_Server_setMethodNode_callback(server,

```

```

2 methode_Start,
3 &Callbacks_Methode_Start);
4 UA_Server_setMethodNodeAsync(server, methode_Start, UA_TRUE);

```

Nun kann man die Methoden Asynchron aufrufen.

8.3 Browse

Ein wichtiger Grund, warum man überhaupt ein Informationsmodel braucht ist die Möglichkeit Schematisch Objekte abzufragen. also das ich wie in einer Ordner Struktur Stück für Stück nach dem richtigen Eintrag suchen kann. Dazu hatte ich ihn der Bachelorarbeit eine kleine Helper Funktion geschrieben, mit der man dann auch Fehler einer Anfrage Filtern kann bzw. auch die erfolgreichen Abfragen im Server Log sieht. Als erstes wird ein Array, das die tiefer der Abfrage entspricht, angelegt als Typ UA_Qualified Name. In diesem Fall suchen wir in unserem Hauptobjekt den Ordner Control und danach die Methode Test

```

1 UA_QualifiedName targetNameArr[2] = {UA_QUALIFIEDNAME(*_callbackNsIdx, "Control"),
2 UA_QUALIFIEDNAME(*_callbackNsIdx, "Test"),

```

Danach müssen wir definieren, von wo wir anfangen zu suchen, also das Parent Objekt in dem Fall PaintingStationBelt, hier wissen wir die NodeID, natürlich ist auch eine Abfrage aus dem Root Ordner möglich, also wenn selbst die NodeID des Hauptobjekt unbekannt ist.

```
1 UA_NodeId parent = UA_NODEID_NUMERIC(nsIdx, 3001);
```

Dann benötigen wir auch eine NodeID, wo wir die Abfrage speichern wollen in diesem Fall

```
1 UA_NodeId methode_Test;
```

und können nun die Abfrage vollziehen mit der Helper Funktion

```
1 findChildNode(server, *parent, 2, targetNameArr, &methode_Test);
```

Nach der Abfrage gibt sie eine Rückmeldung als Server Log und beschreibt die NodeID Variable mit der man nun weiter arbeiten kann.

Nur was macht diese Helper Funktion? Dies schauen wir uns nun an.

```

1 int findChildNode(UA_Server *server,
2 UA_NodeId parentNode,
3 const int relativePathCnt,
4 UA_QualifiedName targetNameArr[],
5 UA_NodeId *result)
6 {
7     int ret = 0;
8

```

```
9  UA_BrowsePathResult bpr = UA_Server_browseSimplifiedBrowsePath(server,
10  parentNode, relativePathCnt, targetNameArr);
11
12 if (bpr.statusCode != UA_STATUSCODE_GOOD || bpr.targetsSize < 1)
13 {
14     char msg[200];
15     strcpy(msg, "Find Child Error from: ");
16     for(int tragents = 0; tragents < relativePathCnt; tragents++)
17     {
18         strcat(msg, "/");
19         strcat(msg, (char*)targetNameArr[tragents].name.data);
20     }
21     strcat(msg, " With status Code: ");
22     strcat(msg, UA_StatusCode_name(bpr.statusCode));
23     UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, msg);
24     ret = -1;
25 }
26 else
27 {
28     UA_NodeId_copy(&bpr.targets[0].targetId.nodeId, result);
29     char msg[200];
30     strcpy(msg, "Child found: ");
31     for(int tragents = 0; tragents < relativePathCnt; tragents++)
32     {
33         strcat(msg, "/");
34         strcat(msg, (char*)targetNameArr[tragents].name.data);
35     }
36     UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, msg);
37 }
38
39
40 UA_BrowsePathResult_clear(&bpr);
41
42 return ret;
43 }
```

Diese kann nun nach Belieben ergänzt werden um z.B. den Exception handling des Servers gerecht zu werden.

9 Custom Cmake File

Der bis jetzt benutzte gcc Befehl ist natürlich nur eine Möglichkeit sein Server zu kompilieren, allerdings da der Server mit der Zeit komplexer wird, mehr Dateien einzubinden sind, kann es schon mühselig werden diesen Befehl jedes Mal zu schreiben oder zu suchen und zu kopieren. Deshalb werden wir uns nun mit CMake auseinander setzen.

In unserem einfachen Beispiel des Motor Controller erstellen wir uns in dessen Pfad, deshalb eine Datei mit

```
1 nano CMakeLists.txt
```

und in dieser schreiben wir folgendes

```
1 make_minimum_required(VERSION 3.10)
2
3 # set the project name
4 project(MotorSteuerung)
5
6 set(CMAKE_PREFIX_PATH "/home/parallels/install")
7 message("Cmake Prefix Path is \"${CMAKE_PREFIX_PATH}\"")
8 # open62541 must be installed.
9 # If in custom path, then use -DCMAKE_PREFIX_PATH=/home/user/install
10 # Set with Required Components the Namespace like FullNamespace or
   ReducedNamespace
11 find_package(open62541 1.1 REQUIRED COMPONENTS FullNamespace)
12
13 #The following folder will be included
14 include_directories(${CMAKE_PREFIX_PATH}/include)
15 link_directories(${CMAKE_PREFIX_PATH}/lib)
16
17 # add the executable
18 add_executable(MotorSteuerung main.c MotorSteuerung.c)
19
20 #The following libraries will be linked
21 target_link_libraries(MotorSteuerung open62541 mbedtls mbedx509
   mbedcrypto)
22
23 #Limit Parallel Linking Jobs with -DMY_LIMIT_FLAG=ON
24 option(MY_LIMIT_FLAG "If Build fail you can set this ON to Limit parallel
   Linking Jobs to One" OFF) #OFF by default
25 if(MY_LIMIT_FLAG)
26 set(MY_LIMIT_FLAG "-fno-rtti")
27 message("Set Parallel Linking Jobs to One")
28 endif(MY_LIMIT_FLAG)
29 unset(MY_LIMIT_FLAG CACHE) # <---- this is the important!!
30
31 #Set the the C Standart
```

```
32 set_property(TARGET MotorSteuerung PROPERTY C_STANDARD 99)
```

dies speichern wir mit dem Befehl Control x danach erstellen wir den Ordner build und wechseln in diesen mit

```
1 mkdir build && cd build
```

nun können wir die benötigten Build Artefakte für Cmake erstellen mit der Eingabe

```
1 cmake .. -DCMAKE_PREFIX_PATH=/home/{user}/install -DMY_LIMIT_FLAG=ON
```

und nach dem erfolgreichen Abschluss kann das Projekt gebaut werden mit

```
1 make -j
```

Solange sich nun am CMakeLists.txt File nichts ändert können immer wieder neue Server Versionen bei Änderungen der Source Files mit make -j erstellt werden.

Man kann noch andere Bibliotheken und Klassen einbinden wie z.B. im PaintingStationBelt mit Python (wird in der finalen Version zwar nicht mehr benutzt, ich ließ es allerdings exemplarisch in dem Beispiel) und diversen anderen Klassen. Außerdem hat es sich bewährt auch auf die Eingabe von Präfix Pfad und der Flag zu verzichten und dies direkt in der CMakeLists.txt zu setzen. Die vorherige Version macht vor allem Sinn, wenn man seine Dateien teilt mit anderen, die dann andere Pfade oder Einstellungen benötigen.

```
1 cmake_minimum_required(VERSION 3.10)
2
3 # set the project name
4 project(PaintingStationBelt)
5
6 #Set open62541 Path directly
7 set(CMAKE_PREFIX_PATH "/home/parallels/install")
8 message("Cmake Prefix Path is \"${CMAKE_PREFIX_PATH}\\"")
9 # open62541 must be installed.
10 # If in custom path, then use -DCMAKE_PREFIX_PATH=/home/user/install
11 find_package(open62541 1.1 REQUIRED)
12 #find PythonLibs
13 find_package(PythonLibs REQUIRED)
14 message("Python Prefix Path is \"${PYTHON_INCLUDE_DIRS}\\"")
15
16 #The following folder will be included
17 include_directories(${CMAKE_PREFIX_PATH}/include ${PYTHON_INCLUDE_DIRS}
18 PyhtonScripts)
18 link_directories(${CMAKE_PREFIX_PATH}/lib)
19
20 # add the executable
21 add_executable(PaintingStationBelt main.c PaintingStationBelt.c Callbacks
21 .c OpcUaHelper.c KpiCalculator.c)
22
```

```
23 #The following libraries will be linked
24 target_link_libraries(PaintingStationBelt open62541 mbedtls mbedtlsx509
25     mbedcrypto pthread ${PYTHON_LIBRARIES})
26
27 #Limit Parallel Linking Jobs with -DMY_LIMIT_FLAG=ON
28 option(MY_LIMIT_FLAG "If Build fail you can set this ON to Limit parallel
29     Linking Jobs to One" OFF) #OFF by default
30 if(MY_LIMIT_FLAG)
31     set(MY_LIMIT_FLAG "-fno-lto=1")
32     message("Set Parallel Linking Jobs to One")
33 endif(MY_LIMIT_FLAG)
34 unset(MY_LIMIT_FLAG CACHE) # <---- this is the important!!
35
36 #Set the the C Standart
37 set_property(TARGET PaintingStationBelt PROPERTY C_STANDARD 99)
```

Danach muss man natürlich nur noch im build Ordner

```
1 cmake ..
```

aufrufen gefolgt von

```
1 make -j
```

10 OPC UA Server als Linux Service

Natürlich will man seinen OPC UA Server nicht jedes Mal starten wenn man in benötigt. Deshalb richten wir ein Service dazu ein, der dafür zuständig ist den Server zum richtigen Zeitpunkt zu starten.

Dazu legen wir eine Datei im system Ordner an, in unserem Fall mit dem Namen opcua. Diese Datei benötigt die Dateiendung service

```
1 sudo nano /etc/systemd/system/opcua.service
```

und schreiben in diese folgendes

```
1 [Unit]
2 Description=OPC UA Server Service
3 After=network.target
4 StartLimitIntervalSec=0
5 [Service]
6 Type=simple
7 Restart=always
8 RestartSec=5
9 User=opcua
10 WorkingDirectory=/home/opcua/PaintingStationBelt/build
11 ExecStart=/home/opcua/PaintingStationBelt/build/PaintingStationBelt
12
13 [Install]
14 WantedBy=multi-user.target
```

Im Detail machen die einzelnen Einträge folgendes

```
1 [Unit]
2 Description={Beschreibung des Servers}
3 After={Nach welchen anderen Services soll dieser Services gestartet
      werden}
4 StartLimitIntervalSec={Soll ein Zeitversatz des Starts erfolgen}
5 [Service]
6 Type=simple
7 Restart=always #Bei einem Absturz des Servers startet er immer neu
8 RestartSec=5 #Aber erst nach 5 Sekunden
9 User=opcua #der Nutzer dem der Services gehört
10 WorkingDirectory={Pfad zum Server Ordner}
11 ExecStart={Pfad zur Ausführbaren Server Datei }
12
13 [Install]
14 WantedBy=multi-user.target #kann von jedem Nutzer ausgeführt werden
```

Danach muss man noch die Einträge im Systemd Service neu anlegen lassen mit

```
1 sudo systemctl daemon-reload
```

Nun ist der Services mit folgenden Befehlen erreichbar:

Status Abfrage detailliert:

```
1 sudo systemctl status opcua.service
```

Einfache Abfrage ob der Service aktive ist (praktisch für Abfragen in anderen Programmen):

```
1 sudo systemctl is-active opcua.service
```

Start, Neustart, Stop:

```
1 sudo systemctl start opcua.service
2 sudo systemctl restart opcua.service
3 sudo systemctl stop opcua.service
```

Services Autostart aktivieren und deaktivieren:

```
1 sudo systemctl enable opcua.service
2 sudo systemctl disable opcua.service
```

Sich die Log Ausgabe des Servers Anzeigen lassen:

```
1 sudo journalctl -f -a -uopcua.service
```

11 Git Repository

Im GitRepository (<https://github.com/klementuel/open62541Tutorial>), findet man unter anderen diese Dokumentation, das OPC UA-MotorController und -PaintingStationBelt Beispiel.