**Hochschule Karlsruhe**
University of
Applied Sciences

Fakultät für
**Elektro- und**
**Informationstechnik**

Studiengang Elektro- und Informationstechnik –
Automatisierungstechnik

# Installation Guide

## Car interface with OPC UA

by
**Clemens Diener**

# Table of Contents

# 1 Pre Condition

To install the interface, a Linux System with docker is required.

For the installation it is helpful to have basic knowledge of ROS.

For modifying the interface additional basic knowledge of OPC UA is also helpful.

# 2 Installation and Running

**Note**: Terminal commands have a grey background.

For the first installation, it could be necessary to have internet access at the system.

1. Save the folder *catkin_ws* with alle the data wherever you want. Because the Interface uses docker, it doesn't need to be in the catkine workspace.
2. Start in one console a roscore with `roscore`
3. Open a new terminal.
4. Navigate to the folder with all the data of the interface.
5. To initiate the docker container, run `./init_docker.sh`
    a. The initialization is only necessary at the start and if something changed at the interface.
6. To start the interface run `./start_interfache.sh`

The interface is now running and subscribes to all ROS-topics defined in src/car_interface_opc_ua/config/mapping_rostopic_opcua.xml.
If you want to do changes on the interface, see chapter 3.1 and chapter 3.2.

If you want to use the modified *lidar_test.py* and/or the *login_listener.py* do the following steps:

1. Add the folder car_interface_opc_ua to your src folder of your catkin workspace
2. Run `catkin_make` in your workspace
3. Run in your catkin workspace `source ./devel/setup.bash`
4. Lidar_test: `rosrun car_interface_opc_ua lidar_test.py`
5. Login_listener: `rosrun car_interface_opc_ua login_test.py`

Remark: Both scripts where only created for demonstration purpose and maybe should be adapted in the future.

# 3 Explanation files and folders

| Folder/File | Path | Explanations |
|---|---|---|
| File | Dockerfile | File for construction of the docker container, is used through init_docker.sh |
| File | Init_docker.sh | Script to initiate the docker container |
| File | Start_interface.sh | Script to start the docker container with the interface |
| Folder | Src/car_interface_opc_ua/certificates | The automatically generated certificate and private key of the server is saved here. |
| Folder | Src/car_interface_opc_ua/config | Folder with the config files. Only folder where you should change something. |
| File | Src/car_interface_opc_ua/config/ mapping_rostopic_opcua.xml | File with the mapping of the ROS-topics to the OPC UA nodes. See chapter 3.1 for more information. |
| File | Src/car_interface_opc_ua/config/ opcua_model.xml | File with the structure of the OPC UA server. See chapter 3.2 for more information. |
| File | Src/car_interface_opc_ua/config/ settings.xml | File with settings of the OPC UA server. See chapter 3.3 for more information. |
| Folder | Src/car_interface_opc_ua/logfiles | Folder with logfiles of the interface |
| Folder | Src/car_interface_opc_ua/scripts | Folder with the python scripts |
| File | Src/car_interface_opc_ua/scripts/ create_opc_ua_server_struct.py | This file can be used to create the XML-file of the OPC UA Server structure. See chapter 3.4 for more information |
| File | Src/car_interface_opc_ua/scripts/ lidar_test.py | Adopted lidar_test script, so the measurement is running continuously. |
| File | Src/car_interface_opc_ua/scripts/ login_listener.py | Example how to subscribe to the login data of the interface. |
| File | Src/car_interface_opc_ua/scripts/ opc_ua_server.py | Source code of the interface. |

## 3.1 Mapping_rostopic_opcua.xml

In this file is the mapping from the ROS-topics to the OPC UA nodes parameterized.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<MappRosTopic>
    <ROSUAObject NodeId="ns=2;i=4" RosTopic="Target_1"/>
    <ROSUAObject NodeId="ns=2;i=5" RosTopic="Target_2"/>
</MappRosTopic>
```

(3.1)

The ROS-Topic *Target_1* is mapped to the OPC UA node with the id *ns=2;i=4*

The ROS-Topic *Target_2* is mapped to the OPC UA node with the id *ns=2;i=5*

You can expand this, with adding more lines of *ROSUAObject* with mapping from a ROS-Topic to a OPA UA node.

The datatype of the ROS-Topic needs to match the datatype of the OPC UA node.

## 3.2 Opcua_model.xml

This file contains the structure of the OPC UA server. It can be modified by hand (if you know what you are doing) or replaced with XML-files which are created with programs like UAModeler or by the script *create_opc_ua_server_struct.py*.

## 3.3 settings.xml

This file contains some general settings for the OPC UA Server.

```
<?xml version="1.0" encoding="utf-8"?>
<!--Don't change anything, if you are not sure, what you are doing!-->
<Settings>
    <SignIn>
        <TimeSignInValidInMinutes>100</TimeSignInValidInMinutes>
        <LenghtUsername>4</LenghtUsername>
        <LenghtPassword>4</LenghtPassword>
    </SignIn>
</Settings>
```

(3.2)

1. TimeSignInValidInMinutes: Duration how long the username and password are valid since the creation. Needs to be between 1 and 360.

2. LengthUsername: Length of the randomly created username.

3. LengthUsername: Length of the randomly created password.

The combined length of username and password needs to be at least 6.

In the parsing procedure of the interface, the settings are checked for validation. If a invalid number is configured, the interface will set it to a valid number. Example: If the *TimeSignInValidInMinutes* is set to 5000, the parsing procedure of the settings.xml will set the value in the interface to 360.

## 3.4 create_opc_ua_server_struct.py

This script models an OPC UA server and exports the structure to an XML-file. This XML-file can then be used instead of the *opcua_model.xml* file in the config folder. It can be helpful to copy this script somewhere else to modify it.

If you want to do some changes, it is helpful to have some basic knowledge of OPC UA.

**Note:** The used library opcua-asyncio 1.0.4 has a bug with the numbering of the namespace. Because of that, it is recommended to use the modified opcua-asyncio library. The modified library is found in *src/car_interface_opc_ua/scripts/asyncua*

Mainly two functions are used:

**Function to add an object:**

AddObject(Namespace, ParentObject, Objectname, AllNodesList):

- Namespace: namespace of the object
- ParentObject: The Parent object for the new object
- Objectname: Name of the new object
- AllNodesList: List with alle created nodes. Later used to create the XML-file

**Function to add a variable:**

AddVariable(Namespace, ParentObject, Variablename, InitValue, DataType, AllNodesList):

- Namespace: namespace of the variable
- ParentObject: The Parent object for the new variable
- Variablename: Name of the new variable
- InitValue: Value for the initialization of the variable. Only InitValue or DataType needs to be defined.
- DataType: Datatype of the variable. Only InitValue or DataType needs to be defined.
- AllNodesList: List with alle created nodes. Later used to create the XML-file

Example:

```
# Create main car object:
ObjCar, AllNodes = await AddObject(idx, server.nodes.objects, "Car", AllNodes)

# Create object for all the sensors:
ObjSensors, AllNodes = await AddObject(idx, ObjCar, "Sensors", AllNodes)

# Lidar:
ObjSensor1, AllNodes = await AddObject(idx, ObjSensors, "Lidar", AllNodes)
#_, AllNodes = await AddVariable(idx, ObjSensor1, "AllTargets", [[0.0, 0.0, 0.0],
[0.0, 0.0, 0.0]], None, AllNodes)
_, AllNodes = await AddVariable(idx, ObjSensor1, "Target_1", "Init", None, AllNodes)
_, AllNodes = await AddVariable(idx, ObjSensor1, "Target_2", "Init", None, AllNodes)
```

(3.3)

A main object called “Car” is used. To this object, a object “Sensors” is added an to that “Lidar” is added. Two variables are added to “Lidar”, “Target_1” and “Target_2W, both with the initialization value “Init”, so the datatype will be String.