Siehe auch:
https://www.youtube.com/watch?v=GVIS7GtqLpo&t=490s

Inhaltsverzeichnis

1 Vorbereitung von AppDeamon

1.1 Voraussetzungen

Es müssen folgende Erweiterungen installiert sein:

- HACS/appdeamon

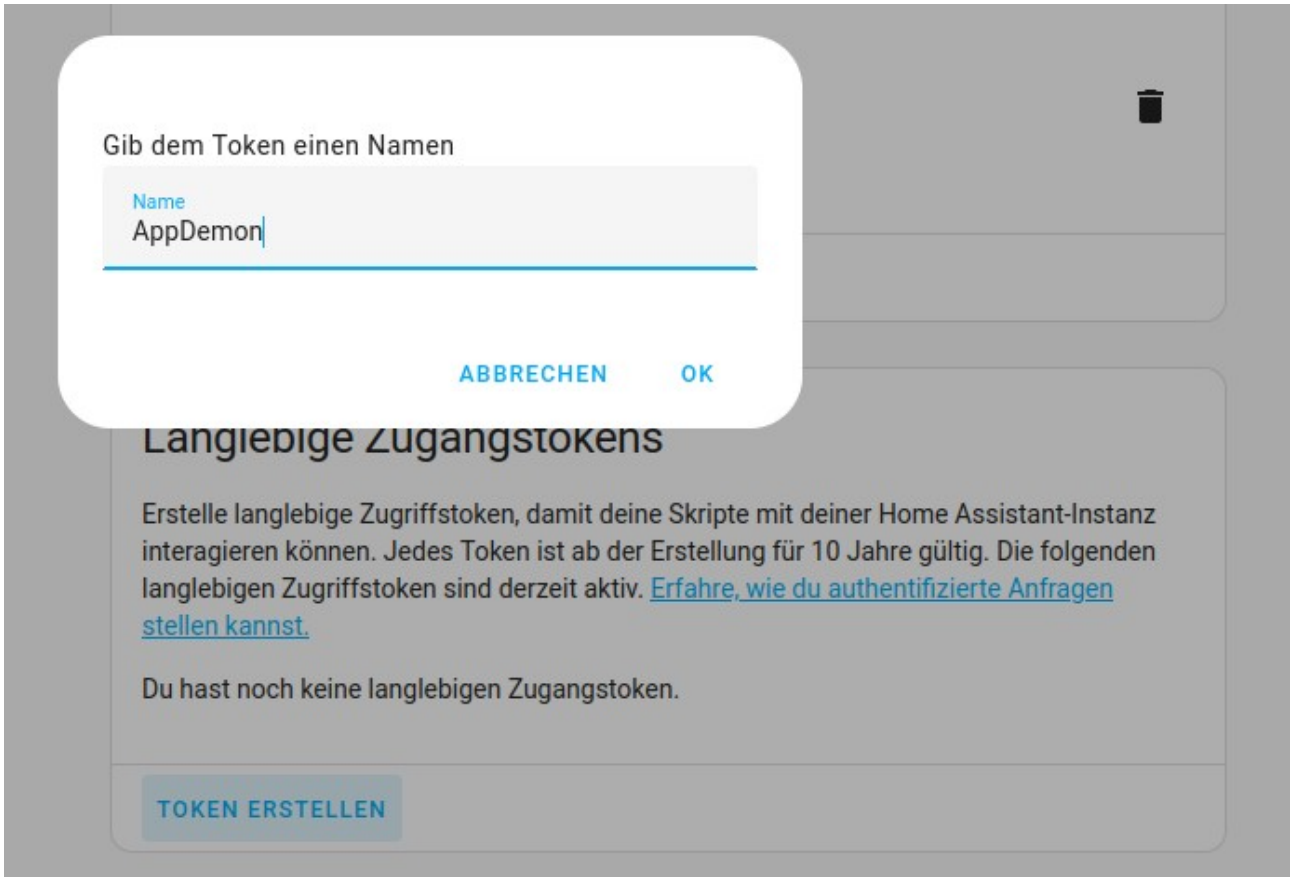
- Terminal

- Studio Code Server

1.2 Token für AppDeamon erstellen

appdeamon muss Zugang zu den Daten und Schnittstellen des HomeAssistent bekommen. Hierzu kann jeder Nutzer einen Token für die Authentifizierung erstellen. Hierzu geht man auf sein Nutzerprofil (letzter Menu-Eintrag im Hauptmenu, das den eigenen Namen trägt).

Als letzter Punkt im User-Profil kann man die Funktion “Token erstellen” wählen:



Der Token muss zwischen gespeichert werden. Bei der Konfiguration des appdeamon wird dieser String benötigt (muss in config/appdeamon/appdeamon.yaml eingefügt werden).

1.3 Sichtbarmachen des AppDeamon Verzeichnisses

Im Terminal (oder über das Terminal des Visual Code Servers) wird das AppDeamon-Verzeichnis des Docker-Containers in das Home-Assistant Verzeichnis über einen Link eingebunden:

[core-ssh ~]$ **ln -s ~/addon_configs ~/config/addon_configs**

Durch den Link erscheinen nun alle Konfigurations-Dateien der Addons in Visual Code Studio.

Überprüfung:

[core-ssh ~]$ **cd config**

[core-ssh config]$ **ls -al**

```
drwxr-xr-x 10 root root     4096 Dec 23 15:31 .
drwxr-xr-x  1 root root     4096 Dec 23 09:52 ..
lrwxrwxrwx  1 root root       19 Dec 23 15:31 addon_configs -> /root/addon_configs
-rw-r--r--  1 root root      830 Oct 10 12:29 automations.yaml
drwxr-xr-x  4 root root     4096 Jun 11  2023 blueprints
drwxr-xr-x  2 root root     4096 Jun 11  2023 .cloud
-rw-r--r--  1 root root     4769 Aug 13 12:09 configuration.yaml
drwxr-xr-x  5 root root     4096 Dec 23 12:45 custom_components
drwxr-xr-x  2 root root     4096 Jun 11  2023 deps
-rw-r--r--  1 root root        9 Dec 17 17:14 .HA_VERSION
-rw-r--r--  1 root root    25769 Dec 23 13:44 home-assistant.log
-rw-r--r--  1 root root     1980 Dec 22 21:32 home-assistant.log.1
-rw-r--r--  1 root root        0 Dec 23 09:53 home-assistant.log.fault
-rw-r--r--  1 root root 58982400 Dec 23 13:37 home-assistant_v2.db
-rw-r--r--  1 root root    32768 Dec 23 16:25 home-assistant_v2.db-shm
-rw-r--r--  1 root root  4132392 Dec 23 16:25 home-assistant_v2.db-wal
drwxr-xr-x  7 root root     4096 Nov 25 05:42 image
-rw-r--r--  1 root root      383 Oct 10 12:22 scenes.yaml
-rw-r--r--  1 root root        0 Jun 11  2023 scripts.yaml
-rw-r--r--  1 root root      161 Jun 11  2023 secrets.yaml
drwxr-xr-x  2 root root     4096 Dec 23 16:24 .storage
-rw-r--r--  1 root root      375 Aug  2 22:21 stromtankstelle.yaml
drwxr-xr-x  2 root root     4096 Jun 12  2023 tts
drwxr-xr-x  3 root root     4096 Jul 29 00:18 www
```

1.4 Erstellen des AppDeamon Arbeitsverzeichnisse

Über den Studio Code Server werden folgende Verzeichnisse erstellt:

v **ADDON_CONFIGS**

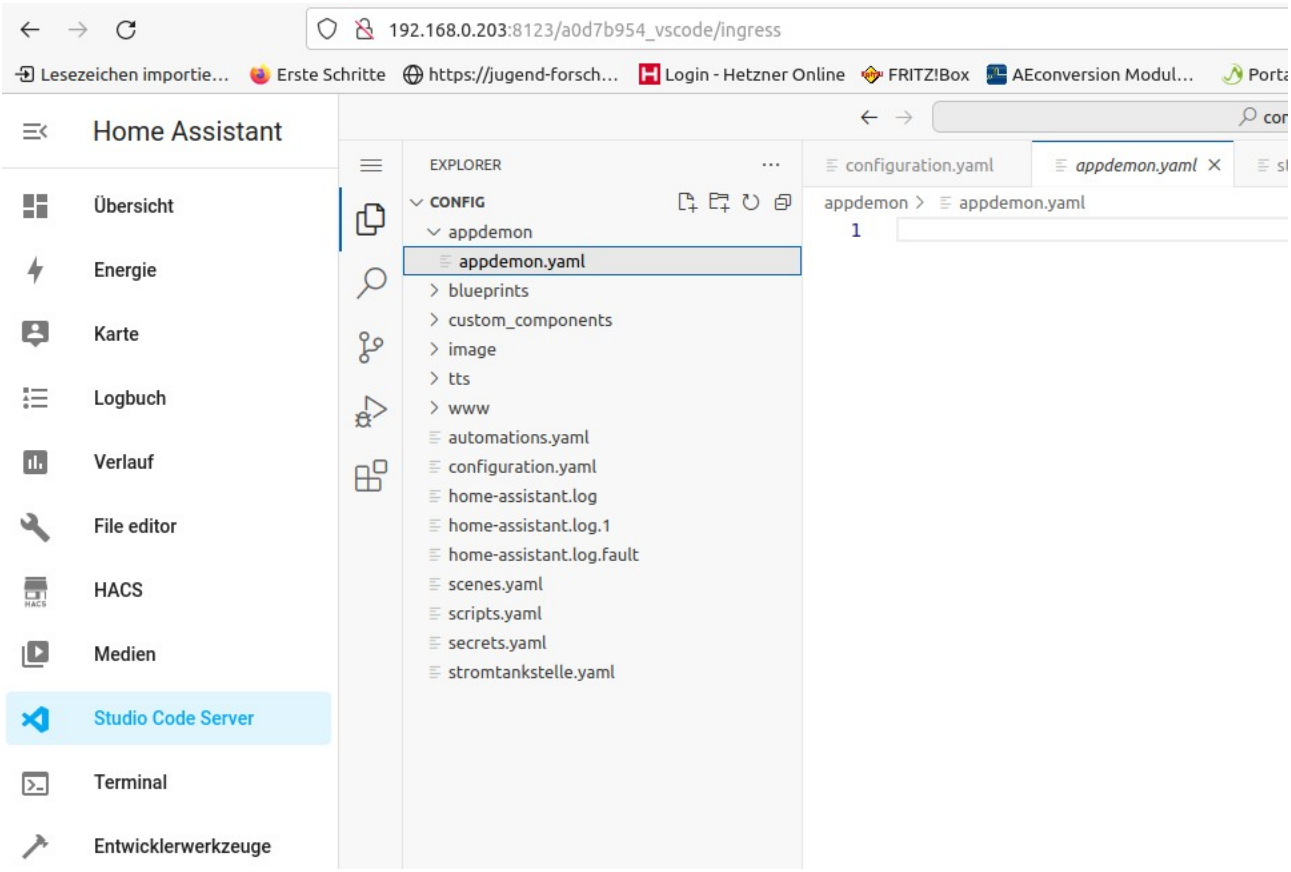
- v **XXXXX_appdeamon**
  - > **apps**
  - > **compiled**
  - > **dashboards**
  - > **namespaces**
  - > **www**

Die roten Verzeichnisse werden neu angelegt. Zu beachten ist hierbei, dass die Verzeichnisse “apps”, “compiled”, “dashboards”, “namespaces” und “www” Unterverzeichnisse von “XXXXX_appdeamon” sind.

1.5 Erstellen der Konfigurationsdatei appdeamon.yaml

Bei richtiger Installation sollte die Konfigurations-Datei bereits vorhanden sein. Wenn die Datei noch nicht vorhanden ist, kann sie mit dem Visual Code Studio Server erstellt werden:

Hierzu wird eine neue “appdeamon.yaml” Datei angelegt und diese mit dem folgenden Code gefüllt:




```yaml
# ADDON_CONFIGS/xxxxx_appdeamon/appdaemon.yaml
appdaemon:
  time_zone: Europe/Berlin
  latitude: 49.78154000
  longitude: 11.18168000
  elevation: 0
  plugins:
    HASS:
      type: hass
      ha_url: http://192.168.0.203:8123
      # Dieser Token dient zur Authentifizierung der App gegenüber Home Assistant.
      # Anstelle hier den User-Name und Password einzutragen (diese Daten könnten auch
      # für Angriffe genutzt werden, wird hier ein Token verwendet.
      # Den Home Assistant Token kann man in der Home Assistant Web-Oberfläche im
      # Menü „User Profil“ (ist immer der letzte Menüpunkt ganz links) generieren lassen.
```

```
    token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJiYWRjMjgwMjlhNjg0MmNjOWEwYTQzNTN
jYWFkYzA5NyIsImlhdCI6MTcwMzMyMzA4MiwiZXhwIjoyMDE4NjgzMDgyfQ.jOQjpSccniE5MQL
XXIm0kgyKj2JxCxNib8Ri8XHpoU4
  # Während des Hochlaufs ist das System nicht vollständig verfügbar.
  # Wenn das Pyton Script erst gestartet werden soll, wenn Home Assistant vollständig
  # gestartet ist, dann trägt man hier die Wartezeit ein. Dies erspart viel Ärger mit dem
  # Fehlermanagement beim Start
  app_init_delay: 1


hadashboard:
  dash_url: http://127.0.0.1:5050
  dash_dir: /config/appdaemon/dashboards

http:
  url: http://127.0.0.1:5050

admin:

api:

#############################################
```

1.6 Anpassen der Konfigurationsdatei apps.yaml

Über den Studio Code Server kann nun die apps.yaml Konfiguration angepasst werden:

config/appdeamon/apps.yaml

```
# ADDON_CONFIGS/xxxxx_appdeamon/APPS/apps.yaml
hello_world:
  module: hello
  class: HelloWorld
#############################################
```

2 Erstellen der Application hello.py

Über den Studio Code Server kann nun die hello.py Applikation erstellt werden:

```
# ADDON_CONFIGS/xxxxx_appdeamon/apps/hello.py
import appdaemon.plugins.hass.hassapi as hass

#
# Hello World App
#
# Args:
#

class HelloWorld(hass.Hass):

  def initialize(self):
    self.log("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
    self.log("Hello from AppDaemon")
    self.log("Hello JUGEND-FORSCHT-TEAM! you are now ready to run Apps!")
    self.log("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
########################################
```

2.1 Aufgaben der Dateien

appdeamon.yaml:

Festlegung wie appdeamon mit dem Home Assistant kommunizieren kann: die URL von Home Assistant und der Zugangs Token werden festgelegt. Zusätzlich werden auch andere wichtige Konfigurationen von appdeamon in dieser Datei bereit gestellt.

apps.yaml:

Die Datei legt fest welche Applications wie gestartet werden dürfen und welche Resourcen diese Apps nutzen dürfen. Über diese Konfiguration können auch Konfigurationsparameter an die App übergeben werden.

hello.py:

Das ist die App. Damit die App mit Home-Assistant kommunizieren kann wird die Hass-Api benötigt:

```
import appdaemon.plugins.hass.hassapi as hass
```

Die hello.py App verwendet als Beispiel nur die Logging-Funktion. Diese ist bei jeder App die wichtigste Funktion. Da es keinen Bildschirm gibt müssen alle Informationen über die Loggingfunktion ausgegeben werden.

2.2 Test der App

2.2.1 Neustart des AppDeamon

Über das Menü HACS->Add Ons->AppDaemon kann der AppDeamon verwaltet werden:

- Neu Starten
- Benutzeroberfläche öffnen (via port 5050)
- Logdatei ansehen (=Protokoll)

Folgende wichtige Punkte, die wir konfiguriert haben können wir nun überprüfen

1. In AppDeamon.yaml haben wir den Zugriffstoken unseres Home-Assistant-Servers eingetragen. D.h. AppDeamon kann sich bei Home Assistant (=HASS) anmelden.

2. Wir haben in der Datei apps.yaml die App die starten soll festgelegt. D.h. die App hello.py sollte gestartet werden.

3. Wir haben in unserer App hello.py Logausgaben eingebaut. Die können wir überprüfen.

Wir starten AppDeamon (ohne Watchdog; wenn AppDeamon nicht startet, wollen wir das sehen. Durch die Aktivierung des Wachhunds würde AppDeamon durch einen Fehler beendet. Der Wachhund würde dies aber merken und AppDeamon gleich wieder starten. Dieses Verhalten ist für die “echte” Applikation wichtig. Aber während der Entwicklung wollen wir sehen welcher Fehler passiert ist.

Wir staten AppDeamon und wechseln zur “Protokoll” Seite. Hier können wir den Fortschritt des Startvorgangs beobachten. Hierzu müssen wir alle paar Sekunden auf “Aktualisieren” klicken. Dann sehen wir life was AppDeamon gerade macht.

```
288152 INFO AppDaemon: AppDaemon Version 4.4.2 starting
288821 INFO AppDaemon: Python version is 3.11.6
289476 INFO AppDaemon: Configuration read from: /config/appdaemon.yaml
523200 INFO HASS: Connected to Home Assistant 2023.12.3

546004 INFO AppDaemon: App 'hello_world' added
752802 INFO AppDaemon: Loading app hello_world using class HelloWorld from module hello
262580 INFO AppDaemon: Calling initialize() for hello_world
980238 INFO hello_world: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
983911 INFO hello_world: Hello from AppDaemon
987819 INFO hello_world: Hello JUGEND-FORSCHT-TEAM! you are now ready to run Apps!
```

```
991445 INFO hello_world: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
996686 INFO AppDaemon: App initialization complete
```

Falls ein Fehler auftritt, kann man über den Visual Code Server den Fehler korrigieren, AppDeamon neu starten und den neuen Log prüfen. Wenn die App editiert wird, ändert sich zur gleichen Zeit das Laufzeitverhalten. Wird im Bereich der Initialisierung eine Veränderung vorgenommen, dann wird diese Anpassung sofort ausgeführt.

3 Erstellen einer „richtigen“ AppDeamon App

3.1 Zugriff auf die Entities von Home Assistant

AppDeamon ermöglicht den Zugriff auf Home Assistant über folgende Funktionen:

3.1.1 Anlegen eines Entity-Objekts

Home Assistant verwaltet alle Geräte über Entity-Ids. Jedes Entity hat eine eindeutige Bezeichnung und eine eindeutige Id.

Mit der Funktion get_entity kann das entsprechende Objekt für die Steuerung eines Entities erstellt werden:

```
self.my_meter = self.get_entity("counter.gfs_cdummymeter")
if (self.my_meter.exists()):
        self.log("meter found")
else:
        self.log("meter not found")
```

Da es neben der Betrachtung der Log-Files keine weitere Möglichkeit zur Fehleranalyse gibt, sollte immer überprüft werden, ob der gerade ausgeführte Befehl erfolgreich war.

3.1.2 Auslesen des Zustands der Entity

Über den Befehl get_state() können die Attribute einer Entity ausgelesen werden. Wird als Prameter <attribute=”all”> verwendet, dann erstellt get_state() eine Struktur in der alle aktuell bekannten attribute gelistet sind.

```
my_meter_all = self.my_meter.get_state(attribute="all")
self.log(f'all = {my_meter_all}')
```

Ausgabe:
all =
{'entity_id': 'counter.gfs_cdummymeter', 'state': '-600', 'attributes': {'editable': True, 'initial': 0, 'step': 100, 'minimum': -5000, 'maximum': 10000, 'icon': 'mdi:circle-slice-5',

'friendly_name': 'Leisungs Messung ändern'}, 'last_changed': '2023-12-25T15:27:32.265626+00:00', 'last_updated': '2023-12-25T15:27:32.265626+00:00', 'context': {'id': '01HJGSD7X8RGPWEW1S8ZRT5ZFM', 'parent_id': None, 'user_id': '50375cffaa724fc4bb65fccd8200244f'}}

Es gibt aber leider einige Attribute auf die der Zugang nicht so einfach möglich ist. Beispiel ist die IP-Adresse eines Sensors. Da diese Information nicht ständig aktualisiert wird, bekommt man als Ergebnis der Abfrage die Information: „None“

3.1.3 Ausgabe im Log-File

```
self.log("Zeile, die im Logfile ausgegeben wird")
```

Wie schon bei der App hello.py gesehen, hat AppDeamon keinen Bildschirm. D.h. alle Aufgaben, die von der App abgearbeitet werden müssen genau im Log beschrieben werden. Nur so kann der Entwickler von außen sehen, was gerade in der App abläuft.

3.1.4 Setzen eines Schaltzustands

Hierzu stellt AppDeamon 3 Funktionen bereit:

<entity>.turn_on()

<entity>.turn_off()

<entity>.toggle()

Mit diesen Funktionen kann ein O-Port, „ein“ oder „aus“ geschaltet werden. Bzw. mit „toggle“ kann der Zustand invertiert werden.

3.1.5 Auswertung der Änderung eines Zustands

Von AppDeamon kann sich das Python-Script über Änderungen informieren lassen. Hierzu gibt es die Funktion:

listen_state(<call_back>)

Diese Funktion kann zum Beispiel genutzt werden um sich bei Änderungen des aktuellen Solarüberschuss informieren zu lassen:

```
self.my_meter.listen_state(self.print_entity)
```

Wenn sich das Entity my_meter ändert, wird die Funktion print_entity aufgerufen. Die print_entity funktion könnte z.B. so aussehen:

```
def print_entity(self, entity, attribute, old, new, kwargs):
    try:
        self.log(" ..............................")
        self.log(f' Entity state change for {entity}')
        self.log(f' attr: {attribute}')
        self.log(f' old: {old}')
        self.log(f' new: {new}')
        self.log(f' kwargs: {kwargs}')
        self.log(" ..............................")
    except Exception as e:
        self.log(f'an exception occurred {e}')
```

3.2 Code Beispiel

```
# ADDON_CONFIGS/xxxxx_appdeamon/apps/cPVman.py
import hassapi as hass
import time
import urllib.request
import urllib.error
import datetime

from datetime import timedelta
#
# App implementing the GFS_cPVman Application
#############################################
#
# https://github.com/ReneTode/My-AppDaemon/tree/master/AppDaemon_for_Beginner
#
#
# Simple Version:
#
# We have 2 hardware items:
# - Endity:
# counter.gfs_cdummymeter
#
# Device: GFS_CSWITCHF01
# - Endity:
# switch.gfs_cswitchf01
# - Sensor
# sensor.gfs_cswitchf01_energy_apparentpower
# sensor.gfs_cswitchf01_energy_current
# sensor.gfs_cswitchf01_energy_factor
# sensor.gfs_cswitchf01_energy_power
# sensor.gfs_cswitchf01_energy_reactivepower
# sensor.gfs_cswitchf01_energy_today
# sensor.gfs_cswitchf01_energy_total
```

```
# sensor.gfs_cswitchf01_energy_totalstarttime
# sensor.gfs_cswitchf01_energy_voltage
# sensor.gfs_cswitchf01_energy_yesterday
# sensor.gfs_cswitchf01_last_restart_time
# sensor.gfs_cswitchf01_mqtt_connect_count
# sensor.gfs_cswitchf01_restart_reason
# sensor.gfs_cswitchf01_ssid
# sensor.gfs_cswitchf01_wifi_connect_count
# sensor.gfs_cswitchf01_firmware_version
# sensor.gfs_cswitchf01_ip
#
#
# Beim Start: counter wird ausgelsen und in lokale variable geschrieben
#
# Bei Änderung des Counters:
# wenn counter > 2100 W
# -> cswitchf01 wird eingeschaltet
# wenn counter < 1000 W
# -> cswitchf01 wird ausgeschaltet
#
#
# Version 1.0:
# Initial Version



class cpvman(hass.Hass):

    def initialize(self):
        self.log("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
        self.log("init cpvman")
        self.log("")

        my_trackers = self.get_trackers()
        self.log(f'trackers: {my_trackers}')
        self.log("");

        self.log("-------------------------")
        self.log("entity: switch.gfs_cswitchf01:")
        self.my_nous = self.get_entity("switch.gfs_cswitchf01")
        if (self.my_nous.exists()):
            self.log("nous power plug found")
        else:
            self.log("nous power plug not found")

        my_nous_all = self.my_nous.get_state(attribute="all")
        self.log(f'all = {my_nous_all}')

        self.my_nous_ip = self.my_nous.get_state(attribute="ip")
        self.log(f'ip = {self.my_nous_ip}')
```

```
        self.my_nous_ssid = self.my_nous.get_state(attribute="ssid")
        self.log(f'ssid = {self.my_nous_ssid}')

        self.my_nous_voltage = self.my_nous.get_state(attribute="energy_voltage")
        self.log(f'voltage = {self.my_nous_voltage}')

        self.my_nous_current = self.my_nous.get_state(attribute="energy_current")
        self.log(f'current = {self.my_nous_current}')

        self.my_nous_power = self.my_nous.get_state(attribute="power")
        self.log(f'power = {self.my_nous_power}')
        self.my_nous_state = self.my_nous.get_state(attribute="state")
        self.log(f'state = {self.my_nous_state}')
        self.my_nous_connect_count =
self.my_nous.get_state(attribute="mqtt_connect_count")
        self.log(f'connect count = {self.my_nous_connect_count}')
        self.log("-------------------------")


        self.log("")
        self.log("-------------------------")
        self.log("entity: counter.gfs_cdummymeter:")
        self.my_meter = self.get_entity("counter.gfs_cdummymeter")
        if (self.my_meter.exists()):
            self.log("meter found")
        else:
            self.log("meter not found")

        my_meter_all = self.my_meter.get_state(attribute="all")
        self.log(f'all = {my_meter_all}')


        my_meter_state = self.my_meter.get_state(attribute="state")
        self.log(f'state = {my_meter_state} W')
        self.log("-------------------------")

        self.log("")


        self.log("-------------------------")
        self.log("register state change: NOUS state")
        self.my_nous.listen_state(self.print_entity)
        self.log("register power change: METER state")
        self.my_meter.listen_state(self.print_entity)
        self.log("register power change: NOUS.Voltage level")
        self.listen_state(self.print_entity, "sensor.gfs_cswitchf01_energy_voltage")
        self.log("-------------------------")

        self.nous_switch_off(self.my_nous)

        self.log("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
```

```
def nous_switch_toggle_two_times(self, nous_entity):
    try:
        self.log("-----------------------------")
        self.log("nous_switch_toggle_two_times()")
        self.log(f'nous: {nous_entity}')
        nous_entity.toggle()
        time.sleep(5) # 5 seconds pause
        nous_entity.toggle()
        self.log("-----------------------------")
    except Exception as e:
        self.log("an exception occurred")

def nous_switch_on(self, nous_entity):
    try:
        self.log("-----------------------------")
        self.log("nous_switch_on()")
        self.log(f'nous: {nous_entity}')
        nous_entity.turn_on()
        self.log("-----------------------------")
    except Exception as e:
        self.log("an exception occurred")

def nous_switch_off(self, nous_entity):
    try:
        self.log("-----------------------------")
        self.log("nous_switch_off()")
        self.log(f'nous: {nous_entity}')
        nous_entity.turn_off()
        self.log("-----------------------------")
    except Exception as e:
        self.log("an exception occurred")


def print_entity(self, entity, attribute, old, new, kwargs):
    try:
        self.log(" ..............................")
        self.log(f' Entity state change for {entity}')
        self.log(f' attr: {attribute}')
        self.log(f' old: {old}')
        self.log(f' new: {new}')
        self.log(f' kwargs: {kwargs}')
        self.log(" ..............................")
    except Exception as e:
        self.log(f'an exception occurred {e}')
```

3.3 Apps.yaml Datei für die cPVman.py App

```
# ADDON_CONFIGS/xxxxx_appdeamon/APPS/apps.yaml
cpv_manager:
   module: cPVman
   class: cpvman

#########################################
```