```
#*****************************************************************************
#                          Phil's Home Automation                             *
#*****************************************************************************
# Program: Import / Export Power calculator for OpenEVSE using DSMRv5 P1 data  *
#*****************************************************************************
# Version log:                                                                *
# ----------------------------------------------------------------------------*
# Version 0.1 Beta: First operational version run on laptop                   *
#-----------------------------------------------------------------------------*
#                                                                             *
#*****************************************************************************
# Program Language : Python 3.8.10                                           *
# Other programs and modules used:                                            *
#   Eclipse Paho MQTT version 1.6.1                                           *
# ----------------------------------------------------------------------------*
# External Connections:                                                       *
# toke/mosquitto MQTT broker "latest" 2018 - docker version on Synology DS220+ *
# OpenEVSE 7.1.3.EU with WiFi 4.1.0                                           *
# Smart Gateways DSMRv5 Wifi with MQTT firmware 2022022001                    *
# Smart Utility Meter DSMRv5 Sagemcom S211 built 2020, installed Sep 2020     *
#                                    serial 253770234                         *
#*****************************************************************************
# Description:                                                                *
# The objective is to reduce the running cost of the BEV by using only        *
# excess solar power to charge the car so that grid import and export is      *
# reduced. This is achieved by throttling the EV charge current so that grid  *
# export is near zero and stop charging if grid current needs to be imported  *
# for charging.                                                               *
# The OpenEVSE charger can be set in the "Eco" mode for solar PV divert.      *
# The input it uses for that is "+I/-E", one number that indicates power import*
# when positive and power export when negative. The unit of measure is Watt.  *
# The input can be received over WiFi as MQTT packets.                        *
# Our house has a smart utility meter following the DSMRv5 standard.          *
# The meter can be read by a small device from the Dutch company "Smart       *
# Gateways" that can be plugged in the P1 port of the smart meter. The little *
# is powered through the P1 port and sends the data to a MQTT broker.         *
# The MQTT broker here is a DS220+ Synology NAS running toke/mosquitto in     *
# Docker. The data available is not in the required "+I/-E" format. The purpose*
# of this Python program is to transform the available data into the required *
# input for the OpenEVSE. To subscribe to MQTT topics and publish the desired *
# for use by the OpenEVSE, paho/MQTT from Eclipse is run on a laptop. The     *
# result is published to the MQTT broker and the OpenEVSE subscribes to that  *
```

```
# result.                                                                  *
#                                                                          *
# Beta testing shows the code works as intended. But there is a challenge with *
# the EV. Whenever the charger stops charging, it cannot resume charging when  *
# power is available again. Within one minute of stopping the charge, the car  *
# goes to sleep and ignores the pilot signal from the charger until the car is *
# "woken up" by e.g. opening and closing the doors.                        *
# This still needs to be addressed.                                          *
#**************************************************************************
#----------- Module Imports -------------------------------------------------
from paho.mqtt import client as mqtt
import time
#----------- End Module Imports ---------------------------------------------

# First we need to define the callback function we need to call to decode the messages
# we are getting back from the broker when they are received.
# Messages are not necessarily sent from the broker at the time the program is run.
# So the way this works is the broker sends a message we subscribed to and it gets
# stored in a buffer in our client. Whenever a new message arrives in the buffer,
# the client.on_message callback is triggered. For our program to be able to process that
# message, it needs to be running. That is why we need to use the client.loop to keep
# looping, i.e. waiting for the on_message callback to be triggered.
# When the callback is activated, the loop will trigger the function we define below.
# Once the function is defined, we then need to associate the callback with this
# callback function.

#----------- Function Definitions Begin -------------------------------------
# We define the read_power_topic function that is later associated with the on_message
# callback. Every time a topic message is received from the broker, this function will run.

def read_power_topic(client, userdata, message):
    # We subscribed to "dsmr/reading/electricity_currently_delivered", the power taken from the grid
    # and "dsmr/reading/electricity_currently_returned", the excess power exported to the grid.
    # Only one of the subscribed topics will be returned at any time this function is run.
    # The power data we receive from dsmr is always zero or a positive number.
    # When the dsmr returns 0.0 for a power variable, it means no power is flowing in that direction
    # so we can ignore it.
    # power1 is what we need to publish for OpenEVSE use. It is initialized to zero, so that is
    # the result if nothing goes. With 0.0 in power1, OpenEVSE will hold charge current at the
    # level it had before when in "eco" mode.

    power1 = 0.0
```

```
    # We look what topic was returned and store that in power1 if the returned variable is not 0.0.
    # power1 needs to be positive if power is imported and negative if (solar) power is exported.
    # Also dsmr UOM for power is kW, while OpenEVSE requires W. So we need to multiply the dsmr power
    # by 1000.0. That is what the OpenEVSE requires for proper solar divert to work properly.

    power = float(str(message.payload.decode("utf-8")))
    if power > 0.0:
        if str(message.topic) == "dsmr/reading/electricity_currently_delivered":
            power1 = power * 1000.0
        elif str(message.topic) == "dsmr/reading/electricity_currently_returned":
            power1 = power * (-1000.0)

    # If the dsmr power topic returned is zero, there is no power flow in the direction associated
    # with the power topic being processed. So then we do not publish power1.
    # Next time around we will get the power for the other direction and publish that if it is not 0.0.
    # OpenEVSE needs a power1 update every 10 seconds. The Smart Gateway publishes every 10 seconds,
    # so this function runs for both power directions every 10 seconds because it is invoked by the
    # broker publishing to the client, which triggers this callback function.
    # So every 10 seconds we can calculate a new power1 value to publish.
    # power1 is fit for use as the (+I/-E) value for the OpenEVSE solar divert.

    if power > 0.0:
        print("power1 = " + str(power1))
        client.publish("DS220/POWER1", str(power1))
        print("---------")

#----------- Function Definitions End --------------------------------------------

# We connect to the broker as Client with name DS220.
# DS220 stands for the Synology NAS where this program ultimately runs.
# The broker also runs on the NAS and has IP address 192.168.1.99.

mqttBroker ="192.168.1.99"
client = mqtt.Client("DS220")
client.connect(mqttBroker)

# We subscribe to the dsmr topics we need. The dsmr\ topics come from the Smart Gateway connected
# to the P1 port of the smart meter.
# We always get the topic messages from the broker one by one. Not in a list. So no need to complicate things
# with multiple topic subscriptions. We simply subscribe for each topic individually.
```

```
client.subscribe("dsmr/reading/electricity_currently_delivered")
client.subscribe("dsmr/reading/electricity_currently_returned")

# As soon as the subscribe method is executed, a separate thread is started that listens
# for messages to come back from the broker. That is what the mqtt.client package does.
# When the broker sends out a subscribed topic to the DS220 client, the client receives
# that message and stores it in a buffer. We need to run the callback function to get to
# the stored message in the buffer and process the message so we can use the data contained
# in the message in our program.
# This is what we do here: the callback on_message is triggered when a new message arrives
# in the buffer. We associate the on_message callback with the function that is defined
# with the def statement in the beginning of the code. That function will be invoked by
# the on_message callback and receive the information contained in the message, so we can
# decode that message for use in the program.

client.on_message = read_power_topic

# On each message returned, the on_message callback is run.
# This means that when the dsmr/reading/electricity_currently_delivered is returned
# the on_message function is run. And when the electricity_currently_returned is
# returned soon after, the on_message callback is run again.

# Every 10 seconds the two messages come back in sequence from the dsmr Smart Gateways client.
# So we need to sit tight and wait for them to come in with this loop.
# In this first test we run the loop for a short time so it quits automatically.

client.loop_start()
time.sleep(40000)
client.loop_stop()

# End of program
```