```
import asyncio
import logging
import json
import RPi.GPIO as GPIO
import time
import random

from azure.iot.device.aio import IoTHubDeviceClient
from azure.iot.device.aio import ProvisioningDeviceClient
from azure.iot.device import MethodResponse
import random
import pnp_helper
import time
import board
import adafruit_dht

logging.basicConfig(level=logging.ERROR)

# Initial the dht device, with data pin connected to:
dhtDevice = adafruit_dht.DHT22(board.D4)
dhtDevice = adafruit_dht.DHT22(board.D4, use_pulseio=False)

# Set up the GPIO pins
GPIO.setup(11, GPIO.IN)

################################
#Insert your device details

DEVICE_ID = "pdi59z0750"
DEVICE_ID_SCOPE = "0ne009BF32B"
DEVICE_KEY = "vQVKCf2uEFMU+1SDiOZL9THrNCoZ/u1luXgI7daEuCw="    #Primary Key

#copy and paste your interface id from your device template
model_id = "dtmi:patient1:patientmonitoring_5wt;1"

#Type your device template component name
sensorName1 = "DHTsensor"

########################################################
# TELEMETRY TASKS


async def send_telemetry_from_temp_controller(device_client, telemetry_msg,
component_name=None):
    msg = pnp_helper.create_telemetry(telemetry_msg, component_name)
    await device_client.send_message(msg)
    print("Sent message")
    print(msg)
    await asyncio.sleep(5)


########################################################

########################################################
# An # END KEYBOARD INPUT LISTENER to quit application


def stdin_listener():
    """
    Listener for quitting the sample
    """
    while True:
        selection = input("Press Q to quit\n")
        if selection == "Q" or selection == "q":
```

```
            print("Quitting...")
            break

######################################################
# MAIN STARTS
async def provision_device(provisioning_host, id_scope, registration_id,
symmetric_key, model_id):
    provisioning_device_client =
ProvisioningDeviceClient.create_from_symmetric_key(
        provisioning_host=provisioning_host,
        registration_id=registration_id,
        id_scope=id_scope,
        symmetric_key=symmetric_key,
    )

    provisioning_device_client.provisioning_payload = {"modelId": model_id}
    return await provisioning_device_client.register()


async def main():
    switch = "DPS"
    if switch == "DPS":
        provisioning_host = (
            "global.azure-devices-provisioning.net"
        )
        id_scope = DEVICE_ID_SCOPE
        registration_id = DEVICE_ID
        symmetric_key = DEVICE_KEY

        registration_result = await provision_device(
            provisioning_host, id_scope, registration_id, symmetric_key,
model_id
        )

        if registration_result.status == "assigned":
            print("Device was assigned")
            print(registration_result.registration_state.assigned_hub)
            print(registration_result.registration_state.device_id)
            device_client = IoTHubDeviceClient.create_from_symmetric_key(
                symmetric_key=symmetric_key,
                hostname=registration_result.registration_state.assigned_hub,
                device_id=registration_result.registration_state.device_id,
                product_info=model_id,
            )
        else:
            raise RuntimeError(
                "Could not provision device. Aborting Plug and Play device
connection."
            )
    else:
        raise RuntimeError(
            "At least one choice needs to be made for complete functioning of
this sample."
        )

    # Connect the client.
    await device_client.connect()

    ####################################################
    # Function to send telemetry every 8 seconds
    #Edit this to send your desired message

    async def send_telemetry():
        print("Sending telemetry from various components")
```

```
        while True:
            try:
                temperature_c = dhtDevice.temperature
                temperature_msg = {"Temperature": temperature_c}
                raw_data = GPIO.input(11)

                await send_telemetry_from_temp_controller(
                    device_client, temperature_msg, sensorName1
                )
                rate = random.randint(66,72)
                rate_msg = {"HeartRate": rate}


                await send_telemetry_from_temp_controller(
                    device_client, rate_msg, sensorName1
                )

            except RuntimeError as error:
        # Errors happen fairly often, DHT's are hard to read, just keep going
                print(error.args[0])
                time.sleep(2.0)
                continue
            except Exception as error:
                dhtDevice.exit()
                raise error

    send_telemetry_task = asyncio.ensure_future(send_telemetry())

    # Run the stdin listener in the event loop
    loop = asyncio.get_running_loop()
    user_finished = loop.run_in_executor(None, stdin_listener)
    # # Wait for user to indicate they are done listening for method calls
    await user_finished

    if not listeners.done():
        listeners.set_result("DONE")

    if not property_updates.done():
        property_updates.set_result("DONE")

    listeners.cancel()
    property_updates.cancel()

    send_telemetry_task.cancel()

    # Finally, shut down the client
    await device_client.shutdown()


#########################################################
# EXECUTE MAIN

if __name__ == "__main__":
    asyncio.run(main())

    # If using Python 3.6 use the following code instead of asyncio.run(main()):
    # loop = asyncio.get_event_loop()
    # loop.run_until_complete(main())
    # loop.close()
```