Firmware Work Product:
Unpacking hex data into a usable & scalable data structure

FW1.1

Purpose

Congratulations on making it to the work product stage of Plentify's Software Engineer interview process! The purpose of this stage is to give you the opportunity to demonstrate your abilities to apply software engineering tools to IoT problems.

Background

As you should know by now, Plentify is using a novel IoT device and data science-driven approach to operating household water heaters, with the goal of maximizing energy efficiency and delivering several other benefits.

In order to develop the IoT device, the Firmware Team has developed a collection of Python Tools to assist testing during production and monitoring post-production device data. One of these tools is a Backhaul Emulator, which listens for incoming messages on the MQTT Broker and **processes the hexadecimal data into a key-value pair data structure** that can be easily understood by a developer.

The objective of this Work Product is to build a Library that enables API's allowing a user to view the hexadecimal data in an easy to read format (such as a Key-Value Pair according to a specific hexadecimal packet structure).

Inputs

An example of a packet, in key-value format as well as Hexadecimal format, is provided in the Appendix. And example of the Program Output is shown in the Appendix

Deliverables

1. A Python Library that contains a structure definition for a given packet as well as supporting function(s) that expose an API for unpacking the given packet.
    a. This should include a second Python file that shows how the library would be used.
    b. Bonus:
        i. Code is well-written and has high readability.
        ii. The code is stored on a git repository and evidence of good use of Git Workflows is shown.
        iii. Library Supports a Build/Pack function as well as a Decode/Unpack function, where Packing transforms data from key-value to hexadecimal and Unpacking transforms data from hexadecimal to key-value pair.
2. A Short write up (not more than 2 pages) describing your thinking, choice of approach and comment on the scalability of the approach.

Appendix

Please note that our packets are built in Little Endian Format (with a word size of 32-bits), you can read more about Endianness here if you are unfamiliar.

Variable Name	**Variable Type**
packet_type	Unsigned 8-bit integer
packet_version	Unsigned 8-bit integer
total_energy_used_watt_hours	Unsigned 32-bit integer
time_drift_milli_seconds	Signed 32-bit integer
flags	Unsigned 8-bit integer, where the first 2 bits represent boolean flags
geyser_is_warm	Boolean flag from structure '*flags*'
geyser_is_drawing_power	Boolean flag from structure '*flags*'

Matching Key-Value Pair to Hexadecimal Data Structure

Key-value:

- packet_type = 1
- packet_version = 0
- total_energy_used_watt_hours = 4552
- time_drift_milli_seconds = 2003
- flags:
    - geyser_is_warn = False
    - geyser_is_drawing_power = True

Hexadecimal version:
0100c8110000d307000002

And example of the conversions shown as terminal output:

```
Hex version of Work Product Packet:
0100c8110000d307000002

Key-Value pair of Work Product Packet:
WorkProductV0(packet_type=1, packet_version=0, total_energy_used_watt_hours=4552, time_drift_milli_seconds=2
003, geyser_is_warm=False, geyser_is_drawing_power=True)
```