payload description version MEEpaper2024

Last edited by **Mohan JULIEN** 1 year ago

Payload description and decoding

Wiki home page

Payload content

Index	Designation	Valeurs	Interpretation
0	version	262	Message format version: 262 = 0x0106 = Version 1.6
1	dive_id	1	Dive index/counter: 1 = dive one
2	latitude	0	GPS Latitude: 0 = no fix
3	longitude	0	GPS Longitude: 0 = no fix
4	ehpe	0	ehpe = estimated horizontal position error, 0 = no fix thus no estimation. Here ehpe is x1000, divide by 1000 to get the true ehpe value.
5	ttf	0	ttf = time to fix (time to achieve a GPS fix), 0 = no fix, thus no time in seconds.
6	surfacetime_s	352	Time at surface: 352 = 352 sec
7	surfsensor_usetime	63	Ignore, No longer used
8	gnss_usetime	382	GPS on time: 382 = 382 seconds
9	gnss_nofix	2	Counter of no GPS fixes, reset if a viable GPS fix is achieved.
10	gnss_timeoutzerosat	1	GPS fix timeout counter. Counts how many times no GPS fix could be achieved during the allocated time window (timeout = 300sec)
11	gnss_nbsat	2	Number of satellites present: 2 satellites
12	gnss_nbsatpow	0	Number of satellites with a SNR >= 30. Here 0 = no viable satellites
13	temperature	2575	Temperature: 2575/100 = 25.75°C
14	battery_mv	4009	Battery voltage in mV: 4009/1000 = 4.009 Volts
15	diveDeepHisto	[10 0 0 0 0]	Raw dive histogram, decoding from index 17 of the table
16	profile	[7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	Raw dive profile: decoding from index 17 of the table
17	tdive_s	10.0	Dive time in seconds. It is the sum of dive times for each dive level
18	profile_tstep_s	15.0	Temporal step for measuring the dive profile: 15 = 15 seconds
19	maxdepth_m	0.7	Maximum dive depth reached in meters
20	avgdepth_m	0.7	Average dive depth value
21	profile_m	[0.7 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	Profiles in meters
22	profile_histotime_s	[285. 0. 0. 0. 0. 0. 0. 15. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	Table of 256 values, containing the time spent at each depth, step of 0.1m. Here 0m = 285 seconds, 0.7m = 15 seconds.
23	profile_histodepth_m	[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. 2	

Decoding the payload using python

Payload decoding can be easily done using Python. The payloads should be received as hexadecimal strings.

Principle of the IOT tag decoder

- Define a low-level function to properly parse the hexstring

```
##############################################
### Low-level function to parse hexa payload
### according to 'bytes_struct ' passed (should not be deited)
#
# input is an hexa string
# output is a 1-level dictionnary of float, int, string , datetime ...)
##############################################
def parse_unpack_hexstring(hexstr,bytes_struct):
    res = dict()
    for field in bytes_struct:
        # load first field values (N bytes = 2*N chars in hex string)
        buf = hexstr[0:(2*bytes_struct[field]['nbbyte'])]
        # pad with zero to have a 4 bytes representation. needed by struct.unpack()
        print(bytes_struct[field],buf)
        # while len(buf) < 8: #
        #     buf = '0'+buf
        # unpack value according to field type
        if bytes_struct[field]['type'][-1] != 's':
            buf =  struct.unpack(bytes_struct[field]['type'], bytes.fromhex(buf))[0]
        else:
            buf = str(buf)
        res[bytes_struct[field]['key']] = buf
        # Remove field read before looping again
        hexstr = hexstr[(2*bytes_struct[field]['nbbyte']):]
    return res
```

Then write a decoder similar to this one :

```
##############################################
# turtle tag payload decoder (from IOT 1)
# version with minimal processing
##############################################
def payload_decoder_turtle_tag_minimal(hexstr):
    print('Running payload decoder (turtle_tag_minimal) ...')
    bytes_struct = {0:{'key':'version', 'nbbyte':2, 'type':'<H'},
                    1:{'key':'dive_id', 'nbbyte':2, 'type':'<H'},
                    2:{'key':'dive_histo', 'nbbyte':2*5, 'type':'s'},
                    3:{'key':'latitude', 'nbbyte':4, 'type':'<i'},
                    4:{'key':'longitude', 'nbbyte':4, 'type':'<i'},
                    5:{'key':'ehpe', 'nbbyte':4, 'type':'<I'},
                    6:{'key':'ttf', 'nbbyte':4, 'type':'<I'},
                    7:{'key':'surfacetime_s', 'nbbyte':4, 'type':'<I'},
                    8:{'key':'surfsensor_usetime', 'nbbyte':4, 'type':'<I'},
                    9:{'key':'gnss_usetime', 'nbbyte':2, 'type':'<H'},
                    10:{'key':'gnss_nofix', 'nbbyte':1, 'type':'<B'},
                    11:{'key':'gnss_timeoutzerosat', 'nbbyte':1, 'type':'<B'},
                    12:{'key':'gnss_nbsat', 'nbbyte':1, 'type':'<B'},
                    13:{'key':'gnss_nbsatpow', 'nbbyte':1, 'type':'<B'},
                    14:{'key':'dive_profile', 'nbbyte':20, 'type':'s'},
                    15:{'key':'temperature', 'nbbyte':2, 'type':'<H'},
                    16:{'key':'battery_mv', 'nbbyte':2, 'type':'<H'},
                    }
    payload = dict()
    ## Decode payload according to data structure.
    ## Save field names and values in dict
    payload = parse_unpack_hexstring(hexstr,bytes_struct)
    return payload
```

Note:

> An example of a **more advanced payload decoder** that handles true dive profile reconstruction and other metrics is given with the git repository in folder `python_tools`.
>
> Link to file : [lib_decoder.py]

Decoding Cayenne LPP payloads

Install package with `pip install pycayennelpp`

Then write a decoder similar to this one :

```
import numpy as np
import pandas as pd
import datetime as dt
import struct
from cayennelpp import LppFrame, LppUtil

#############################################
# cayenne LPP payload decoder (wrapper for pycayennelpp package)
#############################################
def payload_decoder_cayennelpp(hexstr):
    print('Running payload decoder (cayenne lpp) ...')
    payload = dict()
    # build cayenne lpp frame from hexstr
    frame = LppFrame().from_bytes(bytearray.fromhex(hexstr))
    print(frame)
    # dump frame in json format into payload variable
    frame = json.dumps(frame, default=LppUtil.json_encode_type_int)
    frame = json.loads(frame)
    print(frame)
    # convert back to dict
    ref_key = { 0:{'name':'digital_input','count':0},
                1:{'name':'digital_output','count':0},
                2:{'name':'analog_input','count':0},
                3:{'name':'analog_output','count':0},
                101:{'name':'lum_sensor','count':0},
                102:{'name':'pres_sensor','count':0},
                103:{'name':'temp_sensor','count':0},
                104:{'name':'humid_sensor','count':0},
                113:{'name':['acc_x','acc_y','acc_z'],'count':0},
                115:{'name':'baro_sensor','count':0},
                134:{'name':['gyro_x','gyro_y','gyro_z'],'count':0},
                136:{'name':['gps_lat','gps_lng','gps_alt'],'count':0},
               }
    for field in frame:
        # ref_key[field['type']]['count'] = ref_key[field['type']]['count'] +1
        suffix = '_' + '{:02d}'.format(field['channel'])
        if (field['type']==113) or (field['type']==134) or (field['type']==136): # Correspond to GPS data (divided in 3, lat/
            payload[ref_key[field['type']]['name'][0]+suffix] = field['value'][0]
            payload[ref_key[field['type']]['name'][1]+suffix] = field['value'][1]
            payload[ref_key[field['type']]['name'][2]+suffix] = field['value'][2]
        else:
            payload[ref_key[field['type']]['name']+suffix] = field['value'][0]
    return payload
```

Comments