uRADMonitor INDUSTRIAL
Data Payload Structure

uRADMonitor HW106 motherboard

This document describes the uRADMonitor INDUSTRIAL data payload structure for direct data access.

Version

- The data structure corresponds to version HW106 and firmware version SW78 or newer.

Applications

- Decentralized monitoring
- Private monitoring networks
- LoRaWAN payload decoding
- Off grid monitoring

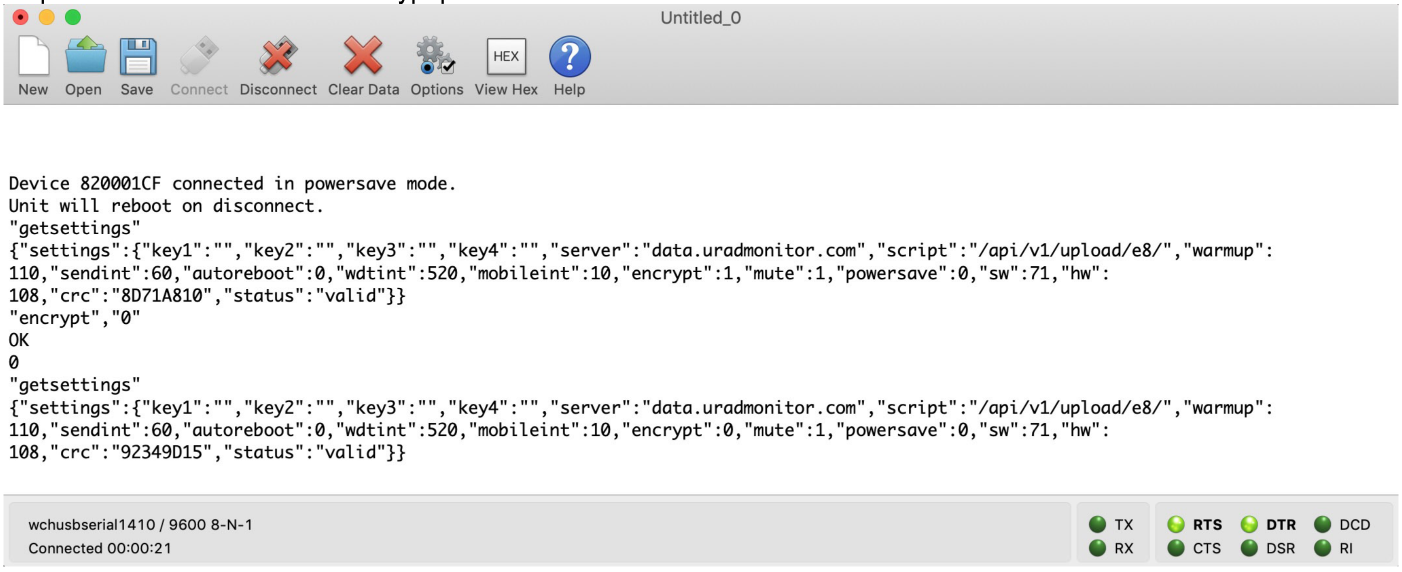
Description

uRADMonitor INDUSTRIAL is an automated, fixed monitoring station with an array of sensors that tracks a total of 12 important environmental parameters. The hardware version HW108 measures Temperature, Barometric pressure, Relative Humidity, Volatile organic compounds (VOC), Particulate matter PM1, PM2.5, PM10, Ozone O_3, Carbon Monoxide CO, Nitrogen Dioxide NO_2, Sulphur Dioxide SO_2, Noise level. The values are packed in a so called payload, a sequence of bytes that uses minimal bandwidth so it can be used across a multitude of networks including LoRaWAN. The payload is machine friendly, but not plain text. So using it outside the uRADMonitor Network and API requires knowing its structure in order to interpret it correctly.

This document presents the payload decoding mechanisms.

Payload Encryption

To protect the integrity of the uRADMonitor Network, the data is encrypted before it is transmitted from the uRADMonitor hardware device to the server. The encryption includes a checksum and a timestamp so it can assure protection against invalid data injection and also repeated injection attacks. The uRADMonitor Data server receives the encrypted payloads then decrypts them before making the data available via the uRADMonitor API.
For Direct Data Access, the payload encryption on the uRADMonitor INDUSTRIAL must first be disabled via USB commands because the decryption algorithm is not publicly available. Once the encryption is disabled, the payload can be decoded as further explained in this document, but the uRADMonitor server will no longer accept the data sent by the device.
To disable payload encryption, connect to the uRADMonitor device via USB (baurate 9600bps) and send the following command: "encrypt","0" . Then check that the change is in place with a second command: "getsettings" . The returned response must show that the encrypt parameter is now 0:



```
Device 820001CF connected in powersave mode.
Unit will reboot on disconnect.
"getsettings"
{"settings":{"key1":"","key2":"","key3":"","key4":"","server":"data.uradmonitor.com","script":"/api/v1/upload/e8/","warmup":
110,"sendint":60,"autoreboot":0,"wdtint":520,"mobileint":10,"encrypt":1,"mute":1,"powersave":0,"sw":71,"hw":
108,"crc":"8D71A810","status":"valid"}}
"encrypt","0"
OK
0
"getsettings"
{"settings":{"key1":"","key2":"","key3":"","key4":"","server":"data.uradmonitor.com","script":"/api/v1/upload/e8/","warmup":
110,"sendint":60,"autoreboot":0,"wdtint":520,"mobileint":10,"encrypt":0,"mute":1,"powersave":0,"sw":71,"hw":
108,"crc":"92349D15","status":"valid"}}
```

wchusbserial1410 / 9600 8-N-1
Connected 00:00:21

Disabling the payload encryption via USB commands

The results are immediate. The encrypted data payload on the INDUSTRIAL HW106 and firmware version SW78 or newer starts with 0x6E and has 51bytes, while the plain payload will start with the device ID and only has 50 bytes:

```
6e26cdea5b2413da650ecdcaf5027dba85f20daa15e5af1d32502e984361977ac73f616a55aff65ae598694a7582ff3a027245
```

Encrypted payload

```
1400008E064E00001C20000000000000000000000732879782031276A32A00028D2C00000D2B00000A040000000200070045
```

Non-encrypted payload

LoRaWAN Network Server data

The INDUSTRIAL HW106 with LoRaWAN connectivity will send the payloads to the Network server. If encryption is disabled as presented above, the plain payload will be accessible directly on the Network Server. The next step is to interpret the packed data and extract all measurements.

Payload structure and decoding

The payload is a packed byte sequence, optimised for low bandwidth. We will take the above plain text payload as an example:

```
1400008E064E00001C20 000000 000000 0000 0000 0732 8797 82 031276 A3 2A 00028D 2C 00000D 2B 00000A 04 000000
02 0007 00 45
```

In order, the structure is as follows:

```
ID 4B - "1400008E"
VERHW 1B - "06" (decimal 6, meaning INDUSTRIAL HW106)
VERSW 1B - "4E" (decimal 78, meaning firmware version 78)
TIMESTAMP 4B - "00001C20" (decimal 7200 sec)
LATITUDE 3B - “000000” (decimal 0, only on devices with NMEA GPS on pin RX6), DI24 encoded with 4 digits *
LONGITUDE 3B - “000000” (decimal 0, only on devices with NMEA GPS on pin RX6), DI24 encoded with 4 digits *
ALTITUDE 2B - “0000” (decimal 0, only on devices with NMEA GPS on pin RX6), DI16 encoded no digits **
SPEED 2B - “0000” (decimal 0, only on devices with NMEA GPS on pin RX6), multiplied by 10. Divide by 10.
TEMPERATURE 2B - “0732" , value 18.42°C, DI16 encoded with 2 digits **
PRESSURE 2B - "8797" offset encoded ***
HUMIDITY 1B - "82" multiplied by 2, 0.5RH resolution, decimal 130 / 2 = 65RH
VOC 3B - "031276" (decimal 201334ohms, or 201KO)
NOISE 1B - "A3" like humidity, multiplied by 2, 0.5dBA resolution, decimal 163/2 = 81dBA
SENSORGASID1 1B - "2A" (Ozone: O3=0x2A, NO2=0x2C, SO2=0x2B, CO=0x04, H2S=0x03, NH3=0x02, CL2=0x31)
SENSORGASCONC1 3B - "00028D" DI24 encoded, 3 digits *, 0.653ppm Ozone
SENSORGASID2 1B - "2C" (Nitrogen Dioxide: O3=0x2A,NO2=0x2C, SO2=0x2B, CO=0x04, H2S=0x03, NH3=0x02, CL2=0x31)
SENSORGASCONC2 3B - "00000D" DI24 encoded, 3 digits *, 0.013ppm NO2
SENSORGASID3 1B - "2B" (Sulphur Dioxide: O3=0x2A, NO2=0x2C, SO2=0x2B, CO=0x04, H2S=0x03, NH3=0x02, CL2=0x31)
SENSORGASCONC3 3B - “00000A" DI24 encoded, 3 digits *, 0.01ppm SO2
SENSORGASID4 1B - "04"  (Carbon Monoxide: O3=0x2A,NO2=0x2C, SO2=0x2B, CO=0x04, H2S=0x03, NH3=0x02, CL2=0x31)
SENSORGASCONC4 3B - "000000" DI24 encoded, 3 digits *, 0ppm CO
PM1 1B - "02" (decimal 2 µg/m³, difference from PM2.5, final value PM2.5=7 - 2= 5 µg/m³ )
PM2.5 2B - "0007" ( decimal 7 µg/m³)
PM10 1B - "00" (decimal 0 µg/m³, difference from PM2.5, final value PM2.5=7 + 2= 7 µg/m³ ))
CRC 1B - “45" ****
```

*DI24 encodes a float to a 24bit integer, with specified number of decimals. To decode, use:

```
C/C++:
double int24ToCoord(uint32_t coord, uint32_t multiplier) {
      if (coord & 0x800000)
            return (coord & 0x7FFFFF) / -(double)multiplier;
      else
            return coord / (double)multiplier;
}

PHP:
public static function uint24float($int, $multiplier) {
            $temp = hexdec($int);
            if ($temp & 0x800000) return ($temp & 0x7FFFFF) / -(double)
$multiplier;
                  else
            return $temp / (double)$multiplier;
      }
```

**DI16 encodes a float to a 16bit integer, with specified number of decimals. To decode, use:

```
PHP:
public static function uint16float($int, $multiplier) {
            $temp = hexdec($int);
            if ($temp & 0x8000) return ($temp & 0x7FFF) / -(double)
$multiplier;
                  else
            return $temp / (double)$multiplier;
      }
```

*** pressure is represented in Pascals, and the normal atmospheric values are a number close to 100000 that exceeds 2bytes to store. The normal fluctuation interval however, is a smaller value. Therefore an offset is used to decrease the number needed to store pressure in Pascals, while still being able to keep track of the ground level fluctuations correctly. An offset of 65535 is applied.

The above example translates too:
Pressure = 0x8BCF = 35791+ 65535 = 101326 Pa = 1013hPa

**** CRC is 1 byte CRC8

C/C++:

```
//CRC-8/MAXIM - based on the CRC8 formulas by Dallas/Maxim
uint8_t crc8(const uint8_t *buffer, uint8_t length) {
      uint8_t crc = 0x00;
      for (int i=0;i<length;i++) {
            crc = crc ^ *(buffer++);
            for (int j=0;j<8;j++) {
                  if (crc & 1)
                              crc = (crc >> 1) ^ 0x8C;
                  else
                              crc = crc >> 1;
            }
      }
      return crc; // not-complemented!
}
```

For a scale comparison on the numbers, you can connect via USB and send the “getdata” command to get device data and compare it to your decoding.