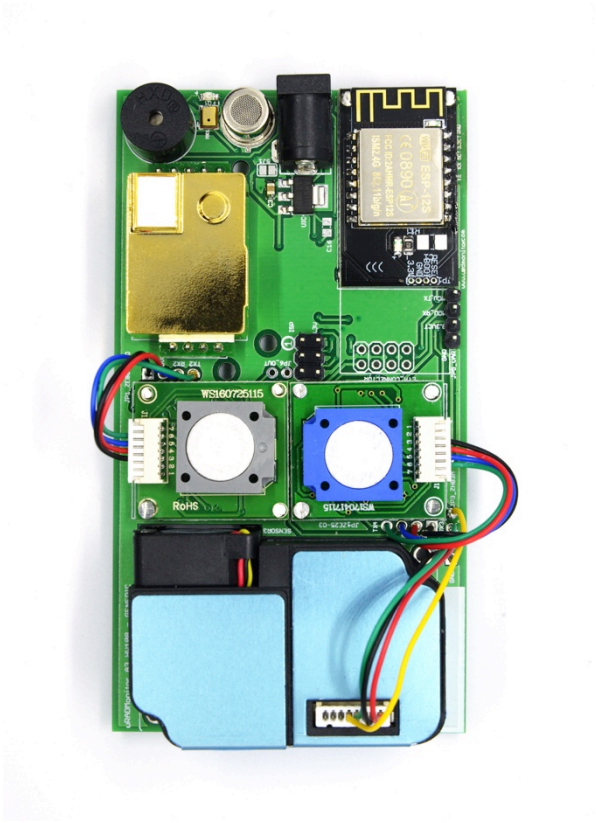


uRADMonitor HW108 motherboard

This document describes the uRADMonitor A3 data payload structure for direct data access.

Version

- The data structure corresponds to version HW108

Applications

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

Description

uRADMonitor A3 is an automated, fixed monitoring station with an array of sensors that tracks a total of 11 important environmental parameters. The hardware version HW108 measures Temperature, Barometric pressure, Relative Humidity, Volatile organic compounds (VOC), Formaldehyde, Ozone, Particulate matter PM1, PM2.5, PM10, Carbon Dioxide, 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 model A3 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 A3 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 A3 via USB (baudrate 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:

Disabling the payload encryption via USB commands

The results are immediate. The encrypted data payload on the A3 HW108 starts with 0065 and has 44bytes, while the plain payload will start with the device ID and only has 42 bytes:

```
0065286751FAA69F2C66623FC90FD8CAAFDE06A6406489FD0F9BD456B8534EEF14698AD5308286495DD9C909
```

Encrypted payload

```
820001CF006C00470000017C07428BE41A4D000195CE1621064C00A200000000001100160187F19DCDC
```

Non-encrypted payload

LoRaWAN Network Server data

The A3 HW108 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 A3 HW108 payload is a packed byte sequence optimized for low bandwidth. We will take the above plain text payload as an example:

820001CF006C00470000017C07428BE41A4D000195CE1621064C00A2000000000011001600187F19DCDC

In order, the structure is as follows:

```
ID 4B - "820001C"
VERHW 2B - "006C" (decimal 108, meaning A3 HW108)
VERSW 2B - "0047" (decimal 71)
TIMESTAMP 4B - "0000017C" (decimal 380 sec)
TEMP 2B - "0742" DI encoded *
PRESSURE 2B - "8BE4" offset encoded **
HUMI 2B - "1A4D" DI encoded *
VOC 4B - "000195CE" (decimal 103886ohms, or 103KO)
NOISE 2B - "1621" DI encoded *
CO2 2B - "064C" (decimal 1612 ppm)
CH2O 2B - "00A2" (decimal 162 ppb)
O3 2B - "0000" (decimal 0 ppb)
RESERVED 2B - "0000"
PM1 2B - "0011" (decimal 17 µg/m³)
PM2.5 2B - "0016" ( decimal 22 µg/m³)
PM10 2B - "0018" (decimal 24 µg/m³)
CRC 4B - "7F19DCDC"
```

* temperature, humidity and noise are small rational numbers (float), encoded as integers. The decoding is as follows:

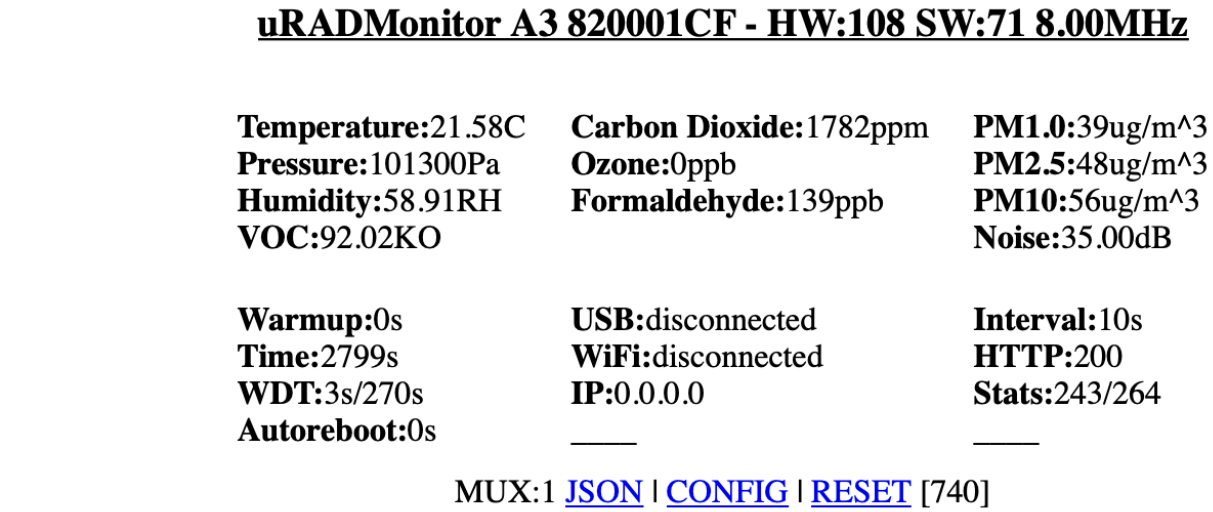
```
// an uint16_t stores 16 bits, we use first bit for sign, and following 15 bits for number (0..32767)
// result is divided by 100 for a real with 2 decimals, max is 327.00
double id(uint16_t val) {
  double res = 0;
  uint16_t mask = 1<<15;
  // check negative number
  if (val & mask) {
    val &= ~mask; // remove sign bit
    res = val;
    res = -res;
  } else {
    res = val;
  }
  res /= 100.0; // restore the 2 decimals
  return res;
}
```

Temperature, humidity and noise from the above example translate to:
Temperature = id(0x0742) = id(1858) = 18.58 °C
Humidity = id(0x1A4D) = id(6733) = 67.33 RH
Noise = id(0x1621) = id(5665) = 56.65 dB

** 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 = 0x8BE4 = 35812 + 65535 = 101347 Pa = 1013hPa

For a scale comparison on the numbers, you can load the device's internal webpage, see the image below - please note the timestamp of the screenshot is not aligned to the payload above:



uRADMonitor A3 820001CF - HW:108 SW:71 8.00MHz

Temperature:21.58C	**Carbon Dioxide:**1782ppm	**PM1.0:**39ug/m^3
Pressure:101300Pa	**Ozone:**0ppb	**PM2.5:**48ug/m^3
Humidity:58.91RH	**Formaldehyde:**139ppb	**PM10:**56ug/m^3
VOC:92.02KO		**Noise:**35.00dB
Warmup:0s	**USB:**disconnected	**Interval:**10s
Time:2799s	**WiFi:**disconnected	**HTTP:**200
WDT:3s/270s	**IP:**0.0.0.0	**Stats:**243/264
Autoreboot:0s	____	____

MUX:1 JSON | CONFIG | RESET [740]

Mini webserver exposes the measured parameters

Code wise, the parameters are obtained as follows:

```
uint8_t buf[42] = { 0x82, 0x00, 0x01, 0xCF, 0x00, 0x6C, 0x00, 0x47, 0x00, 0x00, 0x01, 0x7C, 0x07, 0x42,0x8B,
0xE4, 0x1A, 0x4D, 0x00, 0x01, 0x95, 0xCE, 0x16, 0x21, 0x06, 0x4C, 0x00, 0xA2, 0x00, 0x00, 0x00, 0x00, 0x00,
0x11, 0x00, 0x16, 0x00, 0x18, 0x7F, 0x19, 0xDC, 0xDC };
uint32_t deviceID = buf[0] << 24 | buf[1] << 16 | buf[2] << 8 | buf[3];
uint16_t verHW = buf[4] << 8 | buf[5];
uint16_t verSW = buf[6] << 8 | buf[7];
uint32_t localtime = buf[8] << 24 | buf[9] << 16 | buf[10] << 8 | buf[11];
uint16_t temperature = id(buf[12] << 8 | buf[13]);
uint16_t pressure = (buf[14] << 8 | buf[15]) + 65535;
uint16_t humidity = id(buf[16] << 8 | buf[17]);
uint32_t voc = buf[18] << 24 | buf[19] << 16 | buf[20] << 8 | buf[21];
uint16_t noise = id(buf[22] << 8 | buf[23]);
uint16_t co2 = buf[24] << 8 | buf[25];
uint16_t ch2o = buf[26] << 8 | buf[27];
uint16_t o3 = buf[28] << 8 | buf[29];
uint16_t pm1 = buf[32] << 8 | buf[33];
uint16_t pm25 = buf[34] << 8 | buf[35];
uint16_t pm10 = buf[36] << 8 | buf[37];
uint32_t crc = (buf[38] << 24 | buf[39] << 16 | buf[40] << 8 | buf[41]);
```

The last 4 bytes in the payload contain a CRC32 checksum. The CRC is computed as follows and applied on the entire sequence of bytes, except for the CRC itself (42 - 4 bytes):

```
uint32_t crc32(uint8_t *buffer, uint16_t length) {
        uint32_t crc = 0xFFFFFFFFUL; // initial seed
        uint16_t i, j;
        for (i=0; i<length; i++) {
         crc = crc ^ *(buffer++);
                for (j=0; j<8; j++) {
                        if (crc & 1)
                         crc = (crc>>1) ^ 0xEDB88320UL ;
                        else
                         crc = crc >>1 ;
                }
        }
        return crc ^ 0xFFFFFFFFUL;
}
```

Here is the CRC test result:

```
int main() {
	uint32_t crc = (buf[38] << 24 | buf[39] << 16 | buf[40] << 8 | buf[41]);

	uint32_t computed_crc = crc32(buf, 38);

	(crc == computed_crc)?printf("Match"):printf("Error");

	return 0;
}
```

This checksum is used to verify the integrity of the data.

More references

Additional resources can be found on the uRADMonitor A3 product page, online on www.uradmonitor.com/products .