Environment Sensor (LoRaWAN)

Consider using the latest firmware on your hardware

- **See available firmware downloads**

Target Measurement / Purpose

High precision measuring of **Temperature, Humidity, and Pressure**, using the Bosch BME680 environmental sensor.

Configuration

For configuration of the LoRaWAN parameters, please refer to the chapter LoRaWAN configuration in our LoRaWAN background article.

name	description	example value
`MeasureCron`	Cron expression† defining when to read.	`0 0/15 * * * *` for every 15 minutes

† See also our Introduction to Cron expressions.

Payload

Example payloads for each port:

Status Message (Port 1)

Payload: (No Example yet)

Decoded:

```
{
  "temp": 20.4,
  "vBat": 3.0,
  "version": "v0.0.1"
}
```

Data Message (Port 2)

Structure:

name	pos	len	type	description
timestamp	0	5	`uint40`	Time of measurement (see timestamps in our LoRaWAN devices)
error flag	5	1	`byte`	`1`: error, `0`: success
humidity	6	2	`uint16 BE`	Humidity in 1/10 % rH
temperature	8	2	`int16 BE`	Temperature in 1/10 °C
pressure	10	2	`uint16 BE`	Pressure in 10 Pa

Example Payload: `00386d440700015400f527d1`

Decoded:

```
{
  "error": false,
  "humidity": 34,
  "pressure": 1019.3,
  "temperature": 24.5,
  "time": 946684935000
}
```

Parser

Javascript Parser

```
function readVersion(bytes, i) {
    if (bytes.length < 3) {
        return null;
    }
    return "v" + bytes[i] + "." + bytes[i + 1] + "." + bytes[i + 2];
}

function signed(val, bits) {
    if ((val & 1 << (bits - 1)) > 0) { // value is negative (16bit 2's
complement)
        var mask = Math.pow(2, bits) - 1;
        val = (~val & mask) + 1; // invert all bits & add 1 => now
positive value
        val = val * -1;
    }
    return val;
}

function uint40_BE(bytes, idx) {
    bytes = bytes.slice(idx || 0);
    return bytes[0] << 32 |
        bytes[1] << 24 | bytes[2] << 16 | bytes[3] << 8 | bytes[4] << 0;
}

function uint16_BE(bytes, idx) {
    bytes = bytes.slice(idx || 0);
    return bytes[0] << 8 | bytes[1] << 0;
}

function int40_BE(bytes, idx) {
    return signed(uint40_BE(bytes, idx), 40);
}

function int16_BE(bytes, idx) {
    return signed(uint16_BE(bytes, idx), 16);
}
```

```
function ParseStatusMessage(data) {
    var decoded = {};
    decoded.version = readVersion(data, 0);
    decoded.vBat = uint16_BE(data, 5) / 1000.0;
    decoded.temp = int16_BE(data, 3) / 10.0; // byte 8-9 (originally in
10th degree C)
    decoded.msg = "Firmware Version: " + decoded.FirmwareVersion + "
Battery: " + decoded.Vbat + "V Temperature: " + decoded.Temp + "°C";
    return decoded;
}

function ParseDataMessage(data) {
    return {
        "time": int40_BE(data, 0) * 1000,
        "error": !!data[5],
        "humidity": uint16_BE(data, 6) / 10.0,
        "temperature": int16_BE(data, 8) / 10.0,
        "pressure": uint16_BE(data, 10) / 10
    };
}

// Decoder function for TTN
function Decoder(bytes, port) {
    // Decode an incoming message to an object of fields.
    var decoded;

    switch (port) {
        case 1:
            decoded = ParseStatusMessage(bytes);
            break;
        case 2:
            decoded = ParseDataMessage(bytes);
            break;
        default:
            decoded = {};
    }

    return decoded;
}

// Wrapper for Lobaro Platform
function Parse(input) {
    // Decode an incoming message to an object of fields.
    var b = bytes(atob(input.data));
    var decoded = Decoder(b, input.fPort);

    if (input.fPort == 2) {
        Device.setProperty("status.firmware", decoded.FirmwareVersion);
        Device.setProperty("status.voltage", decoded.Vbat);
        Device.setProperty("status.temperature", decoded.Temp);
    }

    return decoded;
}

// Wrapper for Loraserver / ChirpStack
function Decode(fPort, bytes) {
    return Decoder(bytes, fPort);
}

/*
// Wrapper for Digimondo niota.io
module.exports = function (payload, meta) {
    const port = meta.lora.fport;
    const buf = Buffer.from(payload, 'hex');

    return Decoder(buf, port);
};
*/
```