Payload Decoding Guide

Sample Decoding - JavaScript

```
// Used for decoding Synetica enLink LoRa Messages
// DN 05 Dec 2019

if (!msg.eui)
	return null;

// ------------------------------------------------------------------------------
// Ignore Port 0 Possible MAC Command
// If there is no payload, there is no need for a port, thus it equals zero
if (msg.port === 0) {
	if (msg.eui) {
		node.warn("Possible MAC Command Received from " + msg.eui);
	} else {
		node.warn("Possible MAC Command Received");
	}
	return null;
}
// Ignore zero payloads
if (msg.payload) {
	if (msg.payload.length === 0) {
		if (msg.eui) {
			node.warn("Zero-length Payload, message ignored from " + msg.eui);
		} else {
			node.warn("Zero-length Payload, message ignored");
		}
		return null;
	}
} else {
	if (msg.eui) {
		node.warn("No Payload, message ignored from " + msg.eui);
	} else {
		node.warn("No Payload, message ignored");
	}
	return null;
}
// ------------------------------------------------------------------------------
// Sensor data messages
const TYPE_TEMP = 0x01;			// 2 bytes  -3276.8°C -> 3276.7°C (-10..80)
const TYPE_RH = 0x02;			// 1 byte   0 -> 255 %RH (Actually 0..100%)
const TYPE_LUX = 0x03;			// 2 bytes  0 -> 65535 Lux
const TYPE_PRESSURE = 0x04;		// 2 bytes  0 -> 65535 mbar or hPa
const TYPE_VOC_IAQ = 0x05;		// 2 bytes  0 -> 500 IAQ Index
const TYPE_O2PERC = 0x06;		// 1 byte   0 -> 25.5%
const TYPE_CO = 0x07;			// 2 bytes  0 -> 655.35 ppm (0..100 ppm)
const TYPE_CO2 = 0x08;			// 2 bytes  0 -> 65535 ppm (0..2000 ppm)
```

```
const TYPE_OZONE = 0x09;                    // 2 bytes  0 -> 6.5535 ppm or 6553.5 ppb (0..1 ppm)
const TYPE_POLLUTANTS = 0x0A;               // 2 bytes  0 -> 6553.5 kOhm (Typically 100..1500 kOhm)
const TYPE_PM25 = 0x0B;                     // 2 bytes  0 -> 65535 ug/m3 (0..1000 ug/m3)
const TYPE_PM10 = 0x0C;                     // 2 bytes  0 -> 65535 ug/m3 (0..1000 ug/m3)
const TYPE_H2S = 0x0D;                      // 2 bytes  0 -> 655.35 ppm (0..100 ppm)
const TYPE_COUNTER = 0x0E;                  // 4 bytes  0 -> 2^32
const TYPE_MB_EXCEPTION = 0x0F;             // Type Byte + MBID + Exception Code so it's Type + 2 bytes
const TYPE_MB_INTERVAL = 0x10;              // Type Byte + MBID + F32 Value - so 6 bytes
const TYPE_MB_CUMULATIVE = 0x11;            // Type Byte + MBID + F32 Value - so 6 bytes again
const TYPE_BVOC = 0x12;                     // Float F32 ppm Breath VOC Estimate equivalent
const TYPE_PIR_COUNT = 0x13;                // 32bit counter. Num of detections
const TYPE_PIR_OCC_TIME = 0x14;             // 32bit Total Occupied Time (seconds)
const TYPE_TEMP_PROBE1 = 0x17;              // S16
const TYPE_TEMP_PROBE2 = 0x18;              // S16
const TYPE_TEMP_PROBE3 = 0x19;              // S16
const TYPE_TEMP_PROBE_IN_BAND_DURATION_S_1 = 0x1A; /* Time temperature probe 1 has spent in 'in band' zone */
const TYPE_TEMP_PROBE_IN_BAND_DURATION_S_2 = 0x1B; /* Time temperature probe 2 has spent in 'in band' zone */
const TYPE_TEMP_PROBE_IN_BAND_DURATION_S_3 = 0x1C; /* Time temperature probe 3 has spent in 'in band' zone */
const TYPE_TEMP_PROBE_IN_BAND_ALARM_COUNT_1 = 0x1D; /* Number of times in band alarm has been activated for
temperature probe 1  */
const TYPE_TEMP_PROBE_IN_BAND_ALARM_COUNT_2 = 0x1E; /* Number of times in band alarm has been activated for
temperature probe 2 */
const TYPE_TEMP_PROBE_IN_BAND_ALARM_COUNT_3 = 0x1F; /* Number of times in band alarm has been activated for
temperature probe 3 */
const TYPE_TEMP_PROBE_LOW_DURATION_S_1 = 0x20; /* Time temperature probe 1 has spent below low threshold */
const TYPE_TEMP_PROBE_LOW_DURATION_S_2 = 0x21; /* Time temperature probe 2 has spent below low threshold */
const TYPE_TEMP_PROBE_LOW_DURATION_S_3 = 0x22; /* Time temperature probe 3 has spent below low threshold */
const TYPE_TEMP_PROBE_LOW_ALARM_COUNT_1 = 0x23; /* Number of times low threshold alarm has been activated for
temperature probe 1  */
const TYPE_TEMP_PROBE_LOW_ALARM_COUNT_2 = 0x24; /* Number of times low threshold alarm has been activated for
temperature probe 2 */
const TYPE_TEMP_PROBE_LOW_ALARM_COUNT_3 = 0x25; /* Number of times low threshold alarm has been activated for
temperature probe 3 */
const TYPE_TEMP_PROBE_HIGH_DURATION_S_1 = 0x26; /* Time temperature probe 1 has spent above high threshold */
const TYPE_TEMP_PROBE_HIGH_DURATION_S_2 = 0x27; /* Time temperature probe 2 has spent above high threshold */
const TYPE_TEMP_PROBE_HIGH_DURATION_S_3 = 0x28; /* Time temperature probe 3 has spent above high threshold */
const TYPE_TEMP_PROBE_HIGH_ALARM_COUNT_1 = 0x29; /* Number of times high threshold alarm has been activated for
temperature probe 1  */
const TYPE_TEMP_PROBE_HIGH_ALARM_COUNT_2 = 0x2A; /* Number of times high threshold alarm has been activated for
temperature probe 2 */
const TYPE_TEMP_PROBE_HIGH_ALARM_COUNT_3 = 0x2B; /* Number of times high threshold alarm has been activated for
temperature probe 3 */
const TYPE_DIFF_PRESSURE = 0X2C;            // S16 -32768->32767 Pa
const TYPE_AIR_FLOW = 0X2D;                 // TBA
const TYPE_VOLTAGE = 0X2E;                  // U16 0-65.535V
const TYPE_CURRENT = 0X2F;                  // U16 0-65.535mA
const TYPE_RESISTANCE = 0X30;               // U16 0-65.535kOhm
const TYPE_LEAK_DETECT_EVT = 0x31;          // U8, 1 or 0, Leak status on resistance rope
const TYPE_VIBRATION_EVT = 0x32;            // U8, 1 or 0, vibration event detected

const TYPE_RANGE_MM = 0x33;                                 // U16 0-65,535 (0-5000mm)
const TYPE_RANGE_IN_BAND_DURATION_S = 0x34;         // U32 seconds: Time target has spent in 'in band' zone
const TYPE_RANGE_IN_BAND_ALARM_COUNT = 0x35;        // U16 count: Number of times target has activated 'in band' zone
const TYPE_RANGE_LOW_DURATION_S = 0x36;                     // U32 seconds: Time target has spent in 'low' zone
const TYPE_RANGE_LOW_ALARM_COUNT = 0x37;            // U16 count: Number of times target has activated 'low' zone
const TYPE_RANGE_HIGH_DURATION_S = 0x38;            // U32 seconds: Time target has spent in 'high' zone
const TYPE_RANGE_HIGH_ALARM_COUNT = 0x39;           // U16 count: Number of times target has activated 'high' zone

const TYPE_PRESSURE_TX = 0x3A;        // U16 Pressure Transducer (0..50,000 mbar)
const TYPE_TEMPERATURE_TX = 0x3B;     // S16 -3276.8°C -> 3276.7°C (-10..80)
```

```
const TYPE_RANGE_AMPL = 0x3E;
const TYPE_CO2E = 0x3F;              // Float F32 ppm CO2e Estimate Equivalent

// Optional KPI values that can be included in the message
const TYPE_CPU_TEMP = 0x40;        // 2 bytes  -3276.8°C -> 3276.7°C (-10..80)
const TYPE_BATT_STATUS = 0x41;     // 1 byte   0=Charging; 1~254 (1.8 - 3.3V); 255=External Power (LoRaWAN Spec)
const TYPE_BATT_VOLT = 0x42;       // 2 bytes  0 -> 3600mV (3600mV=External Power)
const TYPE_RX_RSSI = 0x43;         // 2 bytes  +-32767 RSSI
const TYPE_RX_SNR = 0x44;          // 1 byte   +-128 Signal to Noise Ratio
const TYPE_RX_COUNT = 0x45;        // 2 bytes  0 -> 65535 downlink message count
const TYPE_TX_TIME = 0x46;         // 2 bytes  0 -> 65535 ms
const TYPE_TX_POWER = 0x47;        // 1 byte   +-128 dBm
const TYPE_TX_COUNT = 0x48;        // 2 bytes  0 -> 65535 uplink message count
const TYPE_POWER_UP_COUNT = 0x49;  // 2 bytes  0 -> 65535 counts
const TYPE_USB_IN_COUNT = 0x4A;    // 2 bytes  0 -> 65535 counts
const TYPE_LOGIN_OK_COUNT = 0x4B;  // 2 bytes  0 -> 65535 counts
const TYPE_LOGIN_FAIL_COUNT = 0x4C; // 2 bytes  0 -> 65535 counts
const TYPE_FAN_RUN_TIME = 0x4D;    // 4 bytes 0 -> 2^32s = 136 years

const TYPE_SOUND_MIN = 0x50;       // Float F32 dB(A)
const TYPE_SOUND_AVG = 0x51;       // Float F32 dB(A)
const TYPE_SOUND_MAX = 0x52;       // Float F32 dB(A)

// ------------------------------------------------------------------------------
// Convert binary value bit to Signed 16 bit
function S16(bin) {
        var num = bin & 0xFFFF;
        if (0x8000 & num)
                num = -(0x010000 - num);
        return num;
}
// Convert binary value bit to Signed 8 bit
function S8(bin) {
        var num = bin & 0xFF;
        if (0x80 & num)
                num = -(0x0100 - num);
        return num;
}
// Convert 4 IEEE754 bytes
function fromF32(byte0, byte1, byte2, byte3) {
        var bits = (byte0 << 24) | (byte1 << 16) | (byte2 << 8) | (byte3);
        var sign = ((bits >>> 31) === 0) ? 1.0 : -1.0;
        var e = ((bits >>> 23) & 0xff);
        var m = (e === 0) ? (bits & 0x7fffff) << 1 : (bits & 0x7fffff) | 0x800000;
        var f = sign * m * Math.pow(2, e - 150);
        return f;
}
// Function to decode enLink Messages
function DecodePayload(data) {
        var obj = {};
        obj.short_eui = msg.eui.slice(-8);
        var msg_ok = false;
        for (i = 0; i < data.length; i++) {
                console.log(data[i]);
                switch (data[i]) {
                // Parse Sensor Message Parts
                case TYPE_TEMP: // Temperature deg C
                        obj.temperature = (S16((data[i + 1] << 8) | (data[i + 2]))) / 10;
                        i += 2;
```

```
            msg_ok = true;
            break;
case TYPE_RH: // Humidity %rH
            obj.humidity = (data[i + 1]);
            i += 1;
            msg_ok = true;
            break;
case TYPE_LUX: // Light Level lux
            obj.lux = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
case TYPE_PRESSURE: // Barometric Pressure
            obj.pressure_mbar = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
case TYPE_VOC_IAQ: // Indoor Air Quality (0-500)
            obj.iaq = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
case TYPE_O2PERC: // O2 percentage
            obj.o2perc = (data[i + 1]) / 10;
            i += 1;
            msg_ok = true;
            break;
case TYPE_CO: // Carbon Monoxide
            obj.co_ppm = ((data[i + 1] << 8) | (data[i + 2])) / 100;
            i += 2;
            msg_ok = true;
            break;
case TYPE_CO2: // Carbon Dioxide
            obj.co2_ppm = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
case TYPE_OZONE: // Ozone ppm and ppb
            obj.ozone_ppm = ((data[i + 1] << 8) | (data[i + 2])) / 10000;
            obj.ozone_ppb = ((data[i + 1] << 8) | (data[i + 2])) / 10;
            i += 2;
            msg_ok = true;
            break;
case TYPE_POLLUTANTS: // Pollutants kOhm
            obj.pollutants_kohm = ((data[i + 1] << 8) | (data[i + 2])) / 10;
            i += 2;
            msg_ok = true;
            break;
case TYPE_PM25: // Particulates @2.5
            obj.pm25 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
case TYPE_PM10: // Particulates @10
            obj.pm10 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
case TYPE_H2S: // Hydrogen Sulphide
            obj.h2s_ppm = ((data[i + 1] << 8) | (data[i + 2])) / 100;
```

```
                i += 2;
                msg_ok = true;
                break;

        case TYPE_COUNTER:
                if (obj.counter) {
                        obj.counter.push(
                           [ data[i + 1], ((data[i + 2] << 24) | (data[i + 3] << 16) | (data[i + 4] << 8) | (data[i + 5])) ]);
                } else {
                        obj.counter = [
                                [ data[i + 1], ((data[i + 2] << 24) | (data[i + 3] << 16) | (data[i + 4] << 8) | (data[i + 5])) ]
                        ];
                }
                i += 5;
                msg_ok = true;
                break;
        case TYPE_MB_EXCEPTION: // Modbus Error Code
                if (obj.mb_ex) {
                        obj.mb_ex.push([ data[i + 1], data[i + 2] ]);
                } else {
                        obj.mb_ex = [ [ data[i + 1], data[i + 2] ] ];
                }
                i += 2;
                msg_ok = true;
                break;
        case TYPE_MB_INTERVAL: // Modbus Interval Read
                if (obj.mb_int_val) {
                        obj.mb_int_val.push([ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i +
5]).toFixed(2) ]);
                } else {
                        obj.mb_int_val = [ [ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i +
5]).toFixed(2) ] ];
                }
                i += 5;
                msg_ok = true;
                break;
        case TYPE_MB_CUMULATIVE: // Modbus Cumulative Read
                if (obj.mb_cum_val) {
                        obj.mb_cum_val.push([ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i +
5]).toFixed(2) ]);
                } else {
                        obj.mb_cum_val = [ [ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i +
5]).toFixed(2) ] ];
                }
                i += 5;
                msg_ok = true;
                break;

        case TYPE_BVOC: // Breath VOC Estimate equivalent
                obj.bvoc = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(3);
                i += 4;
                msg_ok = true;
                break;

        case TYPE_PIR_COUNT:
                obj.pir_detection_count = ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
                i += 4;
                msg_ok = true;
                break;
        case TYPE_PIR_OCC_TIME: // Occupied time in seconds
```

```
            obj.pir_occ_time_s = ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE1:
            obj.temp_probe_1 = S16((data[i + 1] << 8 | data[i + 2])) / 10;
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE2:
            obj.temp_probe_2 = S16((data[i + 1] << 8 | data[i + 2])) / 10;
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE3:
            obj.temp_probe_3 = S16((data[i + 1] << 8 | data[i + 2])) / 10;
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_IN_BAND_DURATION_S_1:
            /* Cumulative detection time u32 */
            obj.temp_probe_in_band_duration_s_1 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_IN_BAND_DURATION_S_2:
            /* Cumulative detection time u32 */
            obj.temp_probe_in_band_duration_s_2 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_IN_BAND_DURATION_S_3:
            /* Cumulative detection time u32 */
            obj.temp_probe_in_band_duration_s_3 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_IN_BAND_ALARM_COUNT_1:
            /* In band alarm events u16 */
            obj.temp_probe_in_band_alarm_count_1 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_IN_BAND_ALARM_COUNT_2:
            /* In band alarm events u16 */
            obj.temp_probe_in_band_alarm_count_2 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_IN_BAND_ALARM_COUNT_3:
            /* In band alarm events u16 */
            obj.temp_probe_in_band_alarm_count_3 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_LOW_DURATION_S_1:
            /* Cumulative detection time u32 */
```

```
            obj.temp_probe_low_duration_s_1 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_LOW_DURATION_S_2:
            /* Cumulative detection time u32 */
            obj.temp_probe_low_duration_s_2 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_LOW_DURATION_S_3:
            /* Cumulative detection time u32 */
            obj.temp_probe_low_duration_s_3 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_LOW_ALARM_COUNT_1:
            /* Low alarm events u16 */
            obj.temp_probe_low_alarm_count_1 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_LOW_ALARM_COUNT_2:
            /* Low alarm events u16 */
            obj.temp_probe_low_alarm_count_2 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_LOW_ALARM_COUNT_3:
            /* Low alarm events u16 */
            obj.temp_probe_low_alarm_count_3 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_HIGH_DURATION_S_1:
            /* Cumulative detection time u32 */
            obj.temp_probe_high_duration_s_1 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_HIGH_DURATION_S_2:
            /* Cumulative detection time u32 */
            obj.temp_probe_high_duration_s_2 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_HIGH_DURATION_S_3:
            /* Cumulative detection time u32 */
            obj.temp_probe_high_duration_s_3 =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_HIGH_ALARM_COUNT_1:
            /* High alarm events u16 */
```

```
            obj.temp_probe_high_alarm_count_1 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_HIGH_ALARM_COUNT_2:
            /* High alarm events u16 */
            obj.temp_probe_high_alarm_count_2 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_TEMP_PROBE_HIGH_ALARM_COUNT_3:
            /* High alarm events u16 */
            obj.temp_probe_high_alarm_count_3 = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;

        case TYPE_DIFF_PRESSURE: // 2 bytes S16, +/- 5000
            obj.diff_pressure = S16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case TYPE_VOLTAGE: // 2 bytes U16, 0 to 10 V
            obj.voltage = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_CURRENT: // 2 bytes U16, 0 to 20 mA
            obj.current = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_RESISTANCE: // 2 bytes U16, 0 to 10 kOhm
            obj.resistance = (data[i + 1] << 8) | (data[i + 2]);
            i += 2;
            msg_ok = true;
            break;
        case TYPE_LEAK_DETECT_EVT: // 1 byte U8, Leak status changed
            obj.leak_detect_event = ((data[i + 1] << 8) | (data[i + 2])) ? true : false;
            i += 2;
            msg_ok = true;
            break;
        case TYPE_VIBRATION_EVT: // 1 byte U8, 1 or 0, vibration event detected
            obj.vibration_event = ((data[i + 1] << 8) | (data[i + 2])) ? true : false;
            i += 2;
            msg_ok = true;
            break;

        case TYPE_RANGE_MM:
            obj.range_mm = (data[i + 1] << 8 | data[i + 2]);
            i += 2;
            msg_ok = true;
            break;

        case TYPE_RANGE_IN_BAND_DURATION_S:
            /* Cumulative detection time u32 */
            obj.range_in_band_duration_s =
              ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
```

```
                    break;
                case TYPE_RANGE_IN_BAND_ALARM_COUNT:
                    /* In band alarm events u16 */
                    obj.range_in_band_alarm_count = (data[i + 1] << 8) | (data[i + 2]);
                    i += 2;
                    msg_ok = true;
                    break;
                case TYPE_RANGE_LOW_DURATION_S:
                    /* Cumulative detection time u32 */
                    obj.range_low_duration_s =
                      ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
                    i += 4;
                    msg_ok = true;
                    break;
                case TYPE_RANGE_LOW_ALARM_COUNT:
                    /* In band alarm events u16 */
                    obj.range_low_alarm_count = (data[i + 1] << 8) | (data[i + 2]);
                    i += 2;
                    msg_ok = true;
                    break;
                case TYPE_RANGE_HIGH_DURATION_S:
                    /* Cumulative detection time u32 */
                    obj.range_high_duration_s =
                      ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
                    i += 4;
                    msg_ok = true;
                    break;
                case TYPE_RANGE_HIGH_ALARM_COUNT:
                    /* In band alarm events u16 */
                    obj.range_high_alarm_count = (data[i + 1] << 8) | (data[i + 2]);
                    i += 2;
                    msg_ok = true;
                    break;

// Pressure Transducer
case TYPE_PRESSURE_TX:
  // u16
                    obj.pressure_tx_mbar = (data[i + 1] << 8 | data[i + 2]);
                    i += 2;
                    msg_ok = true;
                    break;
                case TYPE_TEMPERATURE_TX:
  //s16 in deci-celcius
  obj.temperature_tx_degc = (S16((data[i + 1] << 8) | (data[i + 2]))) / 10;
                    i += 2;
                    msg_ok = true;
                    break;

                case TYPE_RANGE_AMPL:
                    obj.range_ampl = (data[i + 1] << 8 | data[i + 2]) / 10;
                    i += 2;
                    msg_ok = true;
                    break;

                case TYPE_CO2E: // CO2e Estimate Equivalent
                    obj.co2e_ppm = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
                    i += 4;
                    msg_ok = true;
                    break;
```

```
case TYPE_SOUND_MIN:
        obj.sound_min_dba = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

case TYPE_SOUND_AVG:
        obj.sound_avg_dba = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

case TYPE_SOUND_MAX:
        obj.sound_max_dba = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;


// Optional KPIs
case TYPE_CPU_TEMP:
        // Scaled
        obj.cpu_temp = data[i + 1] + (Math.round(data[i + 2] * 100 / 256) / 100);
        i += 2;
        msg_ok = true;
        break;
case TYPE_BATT_STATUS:
        obj.batt_status = data[i + 1];
        i += 1;
        msg_ok = true;
        break;
case TYPE_BATT_VOLT:
        obj.batt_volt = ((data[i + 1] << 8) | (data[i + 2])) / 1000;
        i += 2;
        msg_ok = true;
        break;
case TYPE_RX_RSSI:
        obj.rx_rssi = S16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
case TYPE_RX_SNR:
        obj.rx_snr = S8(data[i + 1]);
        i += 1;
        msg_ok = true;
        break;
case TYPE_RX_COUNT:
        obj.rx_count = (data[i + 1] << 8) | (data[i + 2]);
        i += 2;
        msg_ok = true;
        break;
case TYPE_TX_TIME:
        obj.tx_time_ms = (data[i + 1] << 8) | (data[i + 2]);
        i += 2;
        msg_ok = true;
        break;
case TYPE_TX_POWER:
        obj.tx_power_dbm = S8(data[i + 1]);
        i += 1;
        msg_ok = true;
```

```
                        break;
                case TYPE_TX_COUNT:
                        obj.tx_count = (data[i + 1] << 8) | (data[i + 2]);
                        i += 2;
                        msg_ok = true;
                        break;
                case TYPE_POWER_UP_COUNT:
                        obj.power_up_count = (data[i + 1] << 8) | (data[i + 2]);
                        i += 2;
                        msg_ok = true;
                        break;
                case TYPE_USB_IN_COUNT:
                        obj.usb_in_count = (data[i + 1] << 8) | (data[i + 2]);
                        i += 2;
                        msg_ok = true;
                        break;
                case TYPE_LOGIN_OK_COUNT:
                        obj.login_ok_count = (data[i + 1] << 8) | (data[i + 2]);
                        i += 2;
                        msg_ok = true;
                        break;
                case TYPE_LOGIN_FAIL_COUNT:
                        obj.login_fail_count = (data[i + 1] << 8) | (data[i + 2]);
                        i += 2;
                        msg_ok = true;
                        break;
                case TYPE_FAN_RUN_TIME:
                        obj.fan_run_time_s = ((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
                        i += 4;
                        msg_ok = true;
                        break;

                default: // something is wrong with data
                        i = data.length;
                        msg_ok = true;
                        break;
                }
        }
        if (msg_ok) {
                return obj;
        } else {
                return null;
        }
}

var res = DecodePayload(msg.payload);
if (res !== null) {
        var json = JSON.stringify(res, null, 4);
        msg.payload = json;
        return msg;
} else {
        return null;
}
```

Sample Payloads

enLink Zone:

```
0100e5023b0300000403e805005e0803d21300001cf6140001aa62420e10
0100e2023a03000b0403e80500190803a913000000001400000000420e10
0100e7023803000e0403e805005e08039f420e10
0100e6023903000e0403e805006508037c420e10
```

enLink Air:

```
0100ee023d0300000403e905006c06ce0d00000700000a1e910b00030c0004401bf64900004d000f9053
0100ee023d0300000403e905006606ce0d00000700000a1e910b00040c0005401b9e4900004d000f90d1
```

enLink Modbus:

```
101041e2926e11114247dea01112436e0a60101341e2926e11144247dbe01115436e0a60101641e2926e11174247dd40
1118436e18f0101941e2926e111a4247dd40111b436e18f0101c41e2926e111d4247dd40111e436dbcc0101f41e2926e
100041e293121101424802c01102436dfc60100341e2931211044248016011054 36dfc60100641e2931211074247fe80
```

Results:

```
04/10/2019, 11:53:32node: 8897badc.ace73msg.payload : string[208]
"{↲ "temperature": 22.9,↲ "humidity": 59,↲ "lux": 0,↲ "pressure_mbar": 1000,↲ "iaq": 94,↲ "co2_ppm": 978,↲
"pir_detection_count": 7414,↲ "pir_occ_time_s": 109154,↲ "batt_volt": 3.6↲}"
04/10/2019, 11:53:32node: 1012d32e.de251dmsg.payload : string[60]
"0100e5023b0300000403e805005e0803d21300001cf6140001aa62420e10"

04/10/2019, 11:53:37node: 8897badc.ace73msg.payload : string[201]
"{↲ "temperature": 22.6,↲ "humidity": 58,↲ "lux": 11,↲ "pressure_mbar": 1000,↲ "iaq": 25,↲ "co2_ppm": 937,↲
"pir_detection_count": 0,↲ "pir_occ_time_s": 0,↲ "batt_volt": 3.6↲}"
04/10/2019, 11:53:38node: 1012d32e.de251dmsg.payload : string[60]
"0100e2023a03000b0403e80500190803a913000000001400000000420e10"

04/10/2019, 11:53:39node: 8897badc.ace73msg.payload : string[146]
"{↲ "temperature": 23.1,↲ "humidity": 56,↲ "lux": 14,↲ "pressure_mbar": 1000,↲ "iaq": 94,↲ "co2_ppm": 927,↲
"batt_volt": 3.6↲}"
04/10/2019, 11:53:39node: 1012d32e.de251dmsg.payload : string[40]
"0100e7023803000e0403e805005e08039f420e10"

04/10/2019, 11:53:40node: 8897badc.ace73msg.payload : string[145]
"{↲ "temperature": 23,↲ "humidity": 57,↲ "lux": 14,↲ "pressure_mbar": 1000,↲ "iaq": 101,↲ "co2_ppm": 892,↲
"batt_volt": 3.6↲}"
04/10/2019, 11:53:40node: 1012d32e.de251dmsg.payload : string[40]
"0100e6023903000e0403e805006508037c420e10"

04/10/2019, 11:53:41node: 8897badc.ace73msg.payload : string[298]
"{↲ "temperature": 23.8,↲ "humidity": 61,↲ "lux": 0,↲ "pressure_mbar": 1001,↲ "iaq": 108,↲ "o2perc": 20.6,↲
"h2s_ppm": 0,↲ "co_ppm": 0,↲ "pollutants_kohm": 782.5,↲ "pm25": 3,↲ "pm10": 4,↲ "cpu_temp": 27.96,↲
"power_up_count": 0,↲ "fan_run_time_s": 1019987↲}"
04/10/2019, 11:53:41node: 1012d32e.de251dmsg.payload : string[84]
"0100ee023d0300000403e905006c06ce0d00000700000a1e910b00030c0004401bf64900004d000f9053"

04/10/2019, 11:53:42node: 8897badc.ace73msg.payload : string[298]
"{↲ "temperature": 23.8,↲ "humidity": 61,↲ "lux": 0,↲ "pressure_mbar": 1001,↲ "iaq": 102,↲ "o2perc": 20.6,↲
"h2s_ppm": 0,↲ "co_ppm": 0,↲ "pollutants_kohm": 782.5,↲ "pm25": 4,↲ "pm10": 5,↲ "cpu_temp": 27.62,↲
"power_up_count": 0,↲ "fan_run_time_s": 1020113↲}"
04/10/2019, 11:53:42node: 1012d32e.de251dmsg.payload : string[84]
"0100ee023d0300000403e905006606ce0d00000700000a1e910b00040c0005401b9e4900004d000f90d1"

04/10/2019, 11:53:43node: 8897badc.ace73msg.payload : string[582]
"{↲ "mb_int_val": [↲ [↲ 16,↲ 28.32149887084961↲ ],↲ [↲ 19,↲ 28.32149887084961↲ ],↲ [↲ 22,↲
28.32149887084961↲ ]↲ ],↲ "mb_cum_val": [↲ [↲ 17,↲ 49.9674072265625↲ ],↲ [↲ 18,↲ 238.04052734375↲ ],↲ [↲
20,↲ 49.9647216796875↲ ],↲ [↲ 21,↲ 238.04052734375↲ ],↲ [↲ 23,↲ 49.966064453125↲ ]↲ ]↲}"
04/10/2019, 11:53:43node: 1012d32e.de251dmsg.payload : string[96]
"101041e2926e11114247dea01112436e0a60101341e2926e11144247dbe01115436e0a60101641e2926e11174247dd40"

04/10/2019, 11:53:44node: 8897badc.ace73msg.payload : string[581]
```

"{↵ "mb_cum_val": [↵ [↵ 24,↵ 238.097412109375↵],↵ [↵ 26,↵ 49.966064453125↵],↵ [↵ 27,↵ 238.097412109375↵],↵ [↵ 29,↵ 49.966064453125↵],↵ [↵ 30,↵ 237.7373046875↵]↵],↵ "mb_int_val": [↵ [↵ 25,↵ 28.32149887084961↵],↵ [↵ 28,↵ 28.32149887084961↵],↵ [↵ 31,↵ 28.32149887084961↵]↵]↵}"
04/10/2019, 11:53:44node: 1012d32e.de251dmsg.payload : string[96]
"1118436e18f0101941e2926e111a4247dd40111b436e18f0101c41e2926e111d4247dd40111e436dbcc0101f41e2926e"

04/10/2019, 11:53:45node: 8897badc.ace73msg.payload : string[572]
"{↵ "mb_int_val": [↵ [↵ 0,↵ 28.32181167602539↵],↵ [↵ 3,↵ 28.32181167602539↵],↵ [↵ 6,↵ 28.32181167602539↵]↵],↵ "mb_cum_val": [↵ [↵ 1,↵ 50.002685546875↵],↵ [↵ 2,↵ 237.98583984375↵],↵ [↵ 4,↵ 50.0013427734375↵],↵ [↵ 5,↵ 237.98583984375↵],↵ [↵ 7,↵ 49.99853515625↵]↵]↵}"
04/10/2019, 11:53:45node: 1012d32e.de251dmsg.payload : string[96]
"100041e2931211014248O2c01102436dfc60100341e2931211044248016011054360fc60100641e2931211074247fe80"

Technical Support

For technical assistance, please visit the downloads section of our web site at www.synetica.net or email us at support@synetica.net