IoP-MI8-L
Configuration 002:
8x Dendrometer or 4x LAT-B3

ECOMATIK GmbH
Muenchner Str. 23
D-85221 Dachau/Germany
Tel.: +49 8131 260 738
Fax: +49 8131 260 736
e-mail: info@ecomatik.de
website: www.ecomatik.de

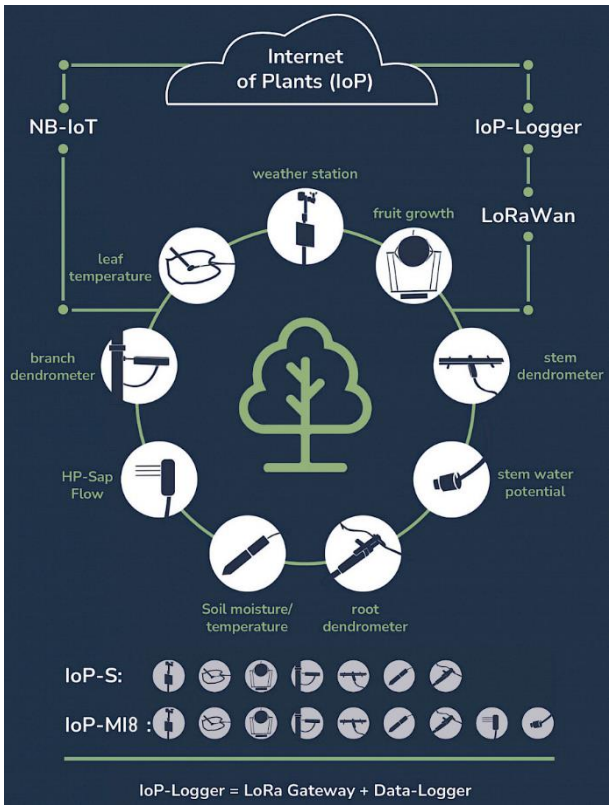
Ecomatik MultiNode IoP-MI8 – Intelligent Plant Data Acquisition with Modern Wireless Technology

Thank you for choosing the Ecomatik MultiNode IoP-MI8!

The IoP-MI8 from Ecomatik is a powerful, flexible, energy-efficient, battery powered measurement solution for the continuous acquisition of plant physiological and environmental data. The system combines precise sensor technology with modern wireless communication for cable-free data transmission – ideal for research, agriculture, and environmental monitoring.

The IoP-MI8 is available in two versions:

- IoP-MI8-**L** – with LoRaWAN module (Low Power, Long Range) for independent LoRa infrastructures
- IoP-MI8-**N** – with NB-IoT module (Narrowband IoT) for direct communication via mobile networks



Both versions are based on the versatile Ecomatik Multi-Interface (MI), which offers the following connection options:

- 8x analog inputs (8x single-ended or 4x differential)
- 1x SDI-12 port
- 1x I²C interface

This variety allows for the simultaneous operation of multiple sensors with a single device, such as e.g.:

- 1× Heat-Pulse Sap Flow Sensor (e.g., N3D1)
- 2× Dendrometers (fruit, branch, stem or root)
- 1× Leaf & Air Temperature Sensor (LAT-B3)
- 1× Stem Water Potential Sensor (FloraPulse)
- 1× Soil Sensor (soil moisture & temperature)
- 1× Air Sensor (air humidity & temperature)

Each IoP-MI8 is fully pre-configured and tailored to the customer's specific requirements.

Getting started is done in just a few simple steps, as described in this quick-start guide for your customized configuration.

Please read this manual carefully before installing and commissioning the device. It also serves as a reference in case of questions during setup or operation.

Note: To operate the LoRa version IoP-MI8-**L**, a LoRa gateway and a LoRa stack server are additionally required to receive and process the transmitted data. The best option here is our IoP-Logger, a LoRa gateway with integrated stack server and logging function.

1. Quick start instructions

a. 📦 Delivery State
- IoP-MI8 is deactivated upon delivery. If MI8 is ordered without IoP-BAT, jumper inside of tranmission node OPEN. If ordered together with IoP-BAT, jumper inside of tranmission node CLOSED and ready, but large 12.8 LiFePO4 battery in IoP-BAT is not connected.

b. 🪡 Mounting
- Use the included tension strap to attach the device to a sturdy tree, stake, or mast.

c. 🌿 Sensor Setup
- Install all sensors on the plant or measurement site.

d. 🔌 Wiring
- Connect sensor cables to the Multi-Interface following the wiring diagram below.

e. ⚙️ Activation (! Only after all sensors are installed and wired properly !)
- If IoP-MI8 was ordered **without** IoP-BAT: Activate device (Jumper inside of tranmission node CLOSED, see photos below)
- If IoP-MI8 was ordered **with** IoP-BAT: connect 12.8 V LiFePO4 battery inside of the IoP-BAT box: red wire => battery(+) and black wire => battery (-)

f. ✅ Verification
- Look for the LED signal after activation.
- Check data reception on the server to confirm proper transmission.

⚠️ **IMPORTANT NOTE:**

!!! **For power-intensive sensors**—such as heat-pulse sap flow sensors or when connecting multiple SDI-12 sensors—an external battery box **(IoP-BATT) is required**.

!!! **Always install and wire all sensors before activation connecting the battery box.**

!!! **Never power heat-pulse sensors unless they are properly installed in a stem or embedded in a heat-absorbing material.** Firing a heat pulse into the air will immediately burn the heater element and irreparably damage the sensor.

!!! Please refer to the **IoP-BATT Manual** for detailed installation and safety instructions.

2. Scope of delivery

IoP-MI8 without IoP-BAT

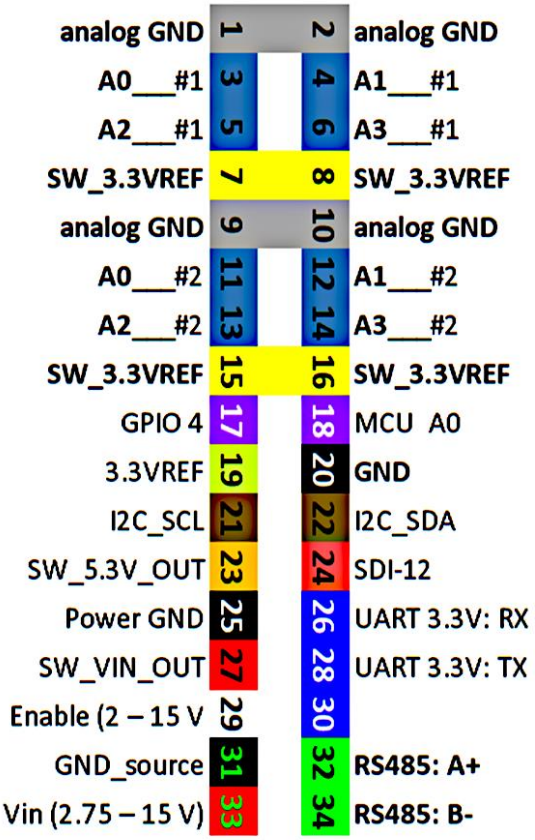
- 1x IoP-MI8 MutliNode
- 1x Antenna
- 1x tension strap
- 2x wood screws

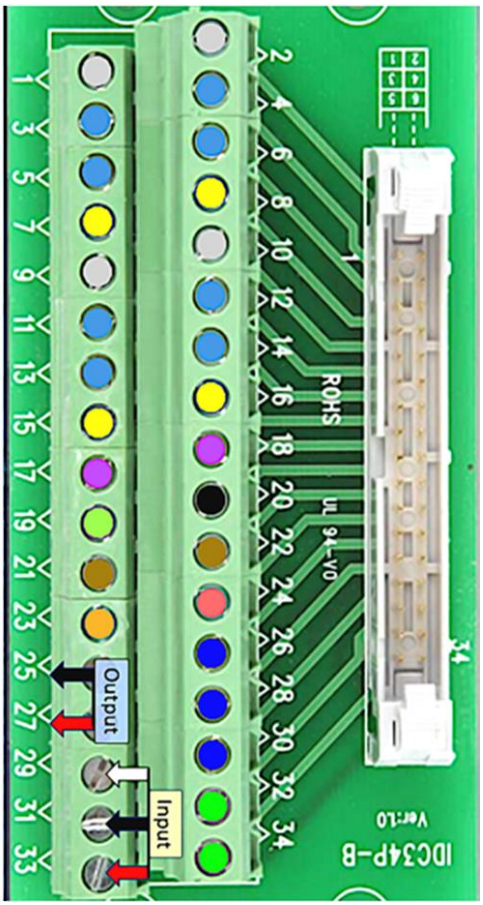
IoP-MI8 with IoP-BAT

- 1x Box equipped with
  - 3.3V power module
  - 12.8 LiFePO4 30 Ah battery
  - Mounting parts for box

3. Channel Description, Wiring & Activation:

3.1. Multi-Interface ports overview:





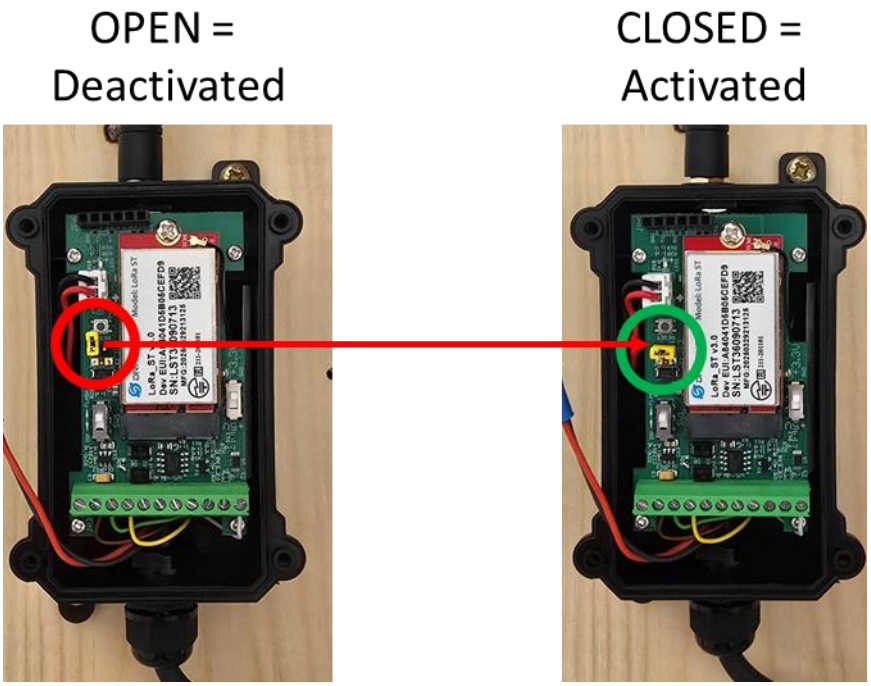
3.2. Sensor wiring (configuration 002, option 1, 8x dendrometer):

Default: Option 1	Sensor-side		Multi-Interface MI8 side		
Sensor option 1 (default)	wire colour	Function of sensor wire	(MI8) Channel Nr.	Channel description	Comment
#1 Dendrometer	white	analog GND	1	analog GND	
	yellow	signal out	3	A0 #1	
	brown	excitation voltage input	7	SW_3.3Vref	
#2 Dendrometer	white	analog GND	2	analog GND	
	yellow	signal out	4	A1 #1	
	brown	excitation voltage input	8	SW_3.3Vref	
#3 Dendrometer	white	analog GND	1	analog GND	
	yellow	signal out	5	A2 #1	
	brown	excitation voltage input	7	SW_3.3Vref	
#4 Dendrometer	white	analog GND	2	analog GND	
	yellow	signal out	6	A3 #1	
	brown	excitation voltage input	8	SW_3.3Vref	
#5 Dendrometer	white	analog GND	9	analog GND	
	yellow	signal out	11	A0 #2	
	brown	excitation voltage input	15	SW_3.3Vref	
#6 Dendrometer	white	analog GND	10	analog GND	
	yellow	signal out	12	A1 #2	
	brown	excitation voltage input	16	SW_3.3Vref	
#7 Dendrometer	white	analog GND	9	analog GND	
	yellow	signal out	13	A2 #2	
	brown	excitation voltage input	15	SW_3.3Vref	
#8 Dendrometer	white	analog GND	10	analog GND	
	yellow	signal out	14	A3 #2	
	brown	excitation voltage input	16	SW_3.3Vref	
I2C T/RH Air Sensor	black (! Correct !)	GND	20	GND	
	yellow	SCL (I2C)	21	SCL (I2C)	
	green	SDA (I2C)	22	SDA (I2C)	
	red	3.3V sensor power supply	19	3.3V out, power supply for sensors	
SDI-12: 1x SMT100	white	GND	25	High Power Output GND	Other configs for SDI-12 sensors are possible
	green	signal out (SDI-12)	24	SDI-12 input	
	brown	5V sensor power supply	23	SW_5V_out	

3.3. Sensor wiring (configuration 002, option 2, 4x LAT-B3):

Option 2	Sensor-Seite		Multi-Interface (MI8)	Multi-Interface MI8 Seite	
Alternative option 2 (on request)	wire colour	Function of sensor wire	Channel Nr.	Channel description	Comment
LAT-B3	grey	analog GND	1	analog GND	
	yellow	signal out (Tair)	3	A0 #1	
	brown	excitation voltage input	7	SW_3.3Vref	
	green	signal out (Tleaf)	4	A1 #1	
	white	analog GND	1	analog GND	
LAT-B3	grey	analog GND	2	analog GND	
	yellow	signal out (Tair)	5	A2 #1	
	brown	excitation voltage input	8	SW_3.3Vref	
	green	signal out (Tleaf)	6	A3 #1	
	white	analog GND	2	analog GND	
LAT-B3	grey	analog GND	9	analog GND	
	yellow	signal out (Tair)	11	A0 #2	
	brown	excitation voltage input	15	SW_3.3Vref	
	green	signal out (Tleaf)	12	A1 #2	
	white	analog GND	9	analog GND	
LAT-B3	grey	analog GND	10	analog GND	
	yellow	signal out (Tair)	13	A2 #2	
	brown	excitation voltage input	16	SW_3.3Vref	
	green	signal out (Tleaf)	14	A3 #2	
	white	analog GND	10	analog GND	
I2C T/RH Air Sensor	black (! Correct !)	GND	20	GND	
	yellow	SCL (I2C)	21	SCL (I2C)	
	green	SDA (I2C)	22	SDA (I2C)	
	red	3.3V sensor power supply	19	3.3V out, power supply for sensors	
SDI-12: 1x SMT100	white	GND	25	High Power Output GND	Other configs for SDI-12 sensors are possible
	green	signal out (SDI-12)	24	SDI-12 input	
	brown	5V sensor power supply +VCC	23	SW_5V_out, power supply for sensors	

3.4. Activation of the device:



4. IoP-MI8 Payload structure:

The general payload of the IoP-MI8 consists of two main parts:

1. General System Section
   This part is generated by the transmitter module (e.g., NB-IoT or LoRaWAN) and includes general system information such as device identification, firmware version, battery voltage, signal quality, and timestamp.
2. Sensor Data Section (from the Multi-Interface)
   This part is provided by the connected Multi-Interface (MI) and contains the actual measurement data. Each measurement value is encoded as a 3-byte triplet. The number and order of these values are variable and depend on the specific configuration of the MI.
3.

Payload from Dragino			In MI data each value is packed in 3 HEX-bytes							...
0	1	2	3	4	5	6	7	8	9	10	11	...
Battery		Payload Version	Value 1 from MI			Value 2 from MI			Value 3 from MI			Value n from MI

Important Note for LoRa version:
In the pre-configured IoP-MI8-L, the payload length is at 51 bytes. In the EU868 region, this ensures unrestricted use of all available transmission data rates starting from DR0.

By default, an uplink interval of 15 minutes is configured (corresponding to the command AT+TDC=900000), which is recommendable for dendrometer measurements.

Please be aware of any applicable Fair Use Policy (e.g., in the case of TTN), which may still limit the air-time of LoRa end devices.

4.1. Payload Decoding for IoP-MI8-L with configuration 002

Payload structure

Byte #	MI Index	Description / Signal	Decoding Method
0 – 1	Dragino	Supply voltage of Dragino transmission node (between 2.8 and 3.6 V if operated on small internal battery, stable at 3.0 V if operated via IoP-BAT)	BATmV = (bytes[0] << 8) | bytes[1]) & 0x7FFF
2 - 2	Dragino	Payload version	
3 – 5	0	Fault current (1234 = OK; 222 = short on 3.3VREF, 333 = short on SW_3.3VREF, 444 = short on SW_5V)	1 & 2
6 – 8	1	Vin MI8 [V]	1 & 2
9 – 11	7	A0_#1: analog SE [ratio in % of SW_3.3Vref]	1 & 2
12 – 14	9	A1_#1: analog SE [ratio in % of SW_3.3Vref]	1 & 2
15 – 17	10	A2_#1: analog SE [ratio in % of SW_3.3Vref]	1 & 2
18 – 20	35	A3_#1: analog SE [ratio in % of SW_3.3Vref]	1 & 2
21 – 23	36	A0_#2: analog SE [ratio in % of SW_3.3Vref]	1 & 2
24 – 26	37	A1_#2: analog SE [ratio in % of SW_3.3Vref]	1 & 2
27 – 29	35	A2_#2: analog SE [ratio in % of SW_3.3Vref]	1 & 2
30 – 32	36	A3_#2: analog SE [ratio in % of SW_3.3Vref]	1 & 2
33 – 35	35	SDI12: SMT100 Volumetric Water Content [%]	1 & 2
36 – 38	36	SDI12: SMT100 Temperature [°C]	1 & 2
39 – 41	11	I2C: AirTemp [°C]	1 & 2
42 – 44	12	I2C: AirHum [%]	1 & 2

Each Value of the MI is coded in a Byte Triplet which can be decoded into an unsigned integer value:

(1) Integer Value = Byte1 + (Byte2 × 2^8) + (Byte3 × 2^16)

JavaScript code snippet:

```
// Combine three bytes to form a 24-bit integer.
// Order: least significant byte first (bytes[index]), then next byte, then highest byte.
// This forms a raw integer value representing the sensor measurement.
const raw_integer = (bytes[index + 2] << 16) | (bytes[index + 1] << 8) | bytes[index];
```

The MI8 is pre-configured to accommodate the value ranges of the connected sensors.
The default range of -100 to +1577.7216 applies to most sensor values (except e.g. Teros 11 or Teros 21). To convert an integer value to a floating-point value, use the following formula:

(2) Decoded Floating-Point Value = (Integer Value / 10 000) – 100

JavaScript code snippet:

```
// Convert default range value (-100 to +1577.7216) from raw integer to floating point:
// Formula: (raw / 10000) - 100
// Apply a rounding to ensure numerical stability and fixed decimals.
const fp_value = Math.round((((raw / 10000) - 100) * 100000 + Number.EPSILON)) / 100000;
```

For large values, the MI8 is pre-configured to encode each byte triplet for a range from -100,000 to +67,772.16 (e.g., in the case of Teros 11 or Teros 21). To convert these large integer values to floating-point values, use the following formula:

(3) Decoded Floating-Point Value = (Integer Value / 100) – 100 000

JavaScript code snippet:

```
// Convert large range value (-100000 to +67772.16) from raw integer to floating point:
// Formula: (raw / 10000) - 100
// Apply a rounding to ensure numerical stability and fixed decimals.
const fp_value = Math.round((((raw / 100) - 100000) * 100 + Number.EPSILON)) / 100;
```

Test Payload for decoder debugging

0d2a01608dcbf706107f4412b05314c07a15d0a1127f3012b0f314c07a16d0a111c07f147f3013000F12d0f917

Payload Part	Byte-Position	Hex-value	Description
Header 1	0–1	0d2a	Node Battery voltage
Header 2	2	1	z. B. Payload-Version
Triplet 1	3–5	608dcb	Fault current code
Triplet 2	6–8	f70610	MI supply voltage
Triplet 3	9–11	7f4412	A0 #1
Triplet 4	12–14	b05314	A1 #1
Triplet 5	15–17	c07a15	A2 #1
Triplet 6	18–20	d0a112	A3 #1
Triplet 7	21–23	7f3012	A0 #2
Triplet 8	24–26	b0f314	A1 #2
Triplet 9	27–29	c07a16	A2 #2
Triplet 10	30–32	d0a111	A3 #2
Triplet 11	33–35	c07f14	SMT100 VWC %
Triplet 12	36–38	7f3013	SMT100 Temp °C
Triplet 13	39–41	000f12	AirTemp °C
Triplet 14	42–44	d0f917	AirHum %

TTN decoder:

A PDF version of this guide for copy-paste the decoder code can be found on the IoP product page on our website on the tab ‘Documents’:
https://ecomatik.de/en/products/data-capture-and-measurement-systems/iop-systems-internet-of-plants/

Decoder Example output (with temperature calculation for LAT-B3 disabled ‘0’ see description in code) for sample payload:

0d2a01608dcbf706107f4412b05314c07a15d0a1127f3012b0f314c07a16d0a111c07f147f3013000F12d0f917


```
{
 "MI_value_01": 1234,
 "MI_value_02": 5.0359,
 "MI_value_03": 19.7183,
 "MI_value_04": 33.2144,
 "MI_value_05": 40.768,
 "MI_value_06": 22.1072,
 "MI_value_07": 19.2063,
 "MI_value_08": 37.3104,
 "MI_value_09": 47.3216,
 "MI_value_10": 15.5536,
 "MI_value_11": 34.3424,
 "MI_value_12": 25.7599,
 "MI_value_13": 18.3488,
 "MI_value_14": 57.128,
 "MI_value_15": -100,
 "MI_value_16": -100,
 "RS485_BL_battery_V": 3.37,
 "RS485_BL_payload_version": 1
}
```

```
// TTN-Javascript-Decoder for IoP-MI8-L with Config_002_ver_001
//

// Global toggle: set to true ('1)' to enable calculation of thermistor temperatures
// and their temperature differences (deltaT). Set to false ('0') to skip these calculations
// and only output raw ratio values.


const CALCULATE_LAT_B3_TEMPERATURES_AND_DELTAS = 0; // default is '0' for raw ratios only

/**
 * Main decoder function for uplink payloads.
 *
 * @param {object} input - The input object containing a 'bytes' array representing the payload.
 * @returns {object} - An object containing decoded data fields.
 */
function decodeUplink(input) {
  const bytes = input.bytes;
  const data = {};

  // --------------------------
  // Constants and coefficients
  // --------------------------

  // Reference resistor value in Ohms used in NTC calculation
  const R_REF = 20000;

  // Steinhart-Hart coefficients for NTC thermistor temperature calculation
  // Given in scientific notation for precision
  const a = 8.4906823733471E-04;
  const b = 2.6092929546095E-04;
  const c = 1.3016564067000E-07;

  // --------------------------
  // Battery voltage decoding
  // --------------------------

  // Battery voltage is stored in bytes 0 and 1 (big endian)
  // Mask with 0x7FFF to ignore the highest bit if used for something else
  // Divide by 1000 to convert millivolts to volts
  const batteryRaw = ((bytes[0] << 8) | bytes[1]) & 0x7FFF;
  data.RS485_BL_battery_V = batteryRaw / 1000;

  // --------------------------
  // Payload version decoding
  // --------------------------

  // Payload version is stored in byte 2
  data.RS485_BL_payload_version = bytes[2];

  // --------------------------
  // Decode 16 sensor triplets (3 bytes each) starting from byte 3
  // --------------------------

  // We'll store all decoded values here first
  const values = [];

  for (let i = 0; i < 16; i++) {
    const index = 3 + i * 3;

    // Combine three bytes to form a 24-bit integer.
    // Order: least significant byte first (bytes[index]), then next byte, then highest byte.
    // This forms a raw integer value representing the sensor measurement.
    const raw = (bytes[index + 2] << 16) | (bytes[index + 1] << 8) | bytes[index];

    // Convert raw integer to a ratio value:
    // Formula: (raw / 10000) - 100
    // Apply a rounding trick to ensure numerical stability and fixed decimals.
    const ratio = Math.round((((raw / 10000) - 100 + Number.EPSILON) * 100000)) / 100000;

    // For thermistor measurements (values 03 to 10), convert ratio to temperature in °C if enabled
    if (CALCULATE_LAT_B3_TEMPERATURES_AND_DELTAS && i >= 2 && i <= 9) {
      values.push(calculateTemperature(ratio, R_REF, a, b, c));
    } else {
```

```
      // For other values, keep the raw ratio
      values.push(ratio);
    }
  }

  // Store all raw or converted values as MI_value_01 through MI_value_16
  for (let i = 0; i < 16; i++) {
    const key = `MI_value_${(i + 1).toString().padStart(2, '0')}`;
    data[key] = values[i];
  }

  // --------------------------
  // Calculate paired temperatures and their differences (delta T)
  // Only for thermistor pairs (value_03 & value_04, value_05 & value_06, etc.)
  // --------------------------

  if (CALCULATE_LAT_B3_TEMPERATURES_AND_DELTAS) {
    // Loop over the 4 pairs: (03,04), (05,06), (07,08), (09,10)

    // Important channel assignment:
    // In each pair:
    // - The first channel is the air temperature sensor (Tair)
    // - The second channel is the leaf temperature sensor (Tleaf)
    // These assignments are fixed by channel pairing and NOT by numerical comparison of values.

    // Channel pairs:
    // Pair #1: value_03 (A0_#1, Tair), value_04 (A1_#1, Tleaf)
    // Pair #2: value_05 (A2_#1, Tair), value_06 (A3_#1, Tleaf)
    // Pair #3: value_07 (A0_#2, Tair), value_08 (A1_#2, Tleaf)
    // Pair #4: value_09 (A2_#2, Tair), value_10 (A3_#2, Tleaf)

    for (let pairIndex = 0; pairIndex < 4; pairIndex++) {
      const airIdx = 3 + pairIndex * 2;    // 1-based index for Tair (e.g. 3,5,7,9)
      const leafIdx = airIdx + 1;          // 1-based index for Tleaf (e.g. 4,6,8,10)

      const tAir = values[airIdx - 1];
      const tLeaf = values[leafIdx - 1];

      const prefix = `LAT_B3_0${pairIndex + 1}__`;

      data[`${prefix}Tleaf_degC`] = tLeaf;
      data[`${prefix}Tair_degC`] = tAir;

      // Delta: Tleaf - Tair
      data[`${prefix}dT_degC`] = (typeof tLeaf === 'number' && typeof tAir === 'number')
        ? Math.round((tLeaf - tAir) * 100) / 100
        : null;
    }
  }

  // Return the decoded data object
  return { data };
}

/**
 * Converts a sensor ratio to temperature in °C using the Steinhart-Hart equation.
 *
 * @param {number} ratio - The raw sensor ratio value (expected between 0 and 1).
 * @param {number} R_REF - Reference resistor value in Ohms.
 * @param {number} a - Steinhart-Hart coefficient a.
 * @param {number} b - Steinhart-Hart coefficient b.
 * @param {number} c - Steinhart-Hart coefficient c.
 * @returns {number|null} Temperature in degrees Celsius rounded to 2 decimals,
 *                        or null if ratio is invalid.
 */
function calculateTemperature(ratio, R_REF, a, b, c) {
  //scale ratio from 0 - 100% to 0 - 1
  ratio = ratio/100.0;

  // Ensure ratio is in a valid range for calculation
  if (ratio > 0 && ratio < 1) {
    // Calculate the NTC resistance based on the ratio and reference resistor
    const rNtc = (1 / ratio - 1) * R_REF;

    // Calculate natural logarithm of the resistance
    const lnR = Math.log(rNtc);
```

```
    // Apply Steinhart-Hart equation to convert resistance to temperature in Kelvin
    const temperatureK = 1 / (a + b * lnR + c * Math.pow(lnR, 3));

    // Convert Kelvin to Celsius
    const temperatureC = temperatureK - 273.15;

    // Round to two decimals and return
    return Math.round(temperatureC * 100) / 100;
  } else {
    // Invalid ratio values return null
    return null;
  }
}

/*
================================================================================
DOCUMENTATION
================================================================================

Decoder Overview:
-----------------
This decoder function interprets a LoRaWAN payload structured as follows:

- Bytes 0–1: Battery voltage (big endian)
- Byte 2: Payload version
- Bytes 3–50: 16 sensor triplets (3 bytes each), representing various sensor readings.
  Among these, values 03 through 10 (triplets 3 to 10) correspond to thermistor sensor ratios.

Key Features:
-------------
- Converts battery voltage raw value to volts.
- Extracts payload version.
- Parses 16 sensor triplets.
- Converts thermistor sensor ratios (values 03–10) into temperatures (°C) using the Steinhart-Hart
equation.
- Calculates temperature differences (delta T) between paired sensors:
  - Pairs: (value_03 & value_04), (value_05 & value_06), (value_07 & value_08), (value_09 & value_10)
- Assigns temperature values based on fixed channel pairing (not numerical comparison).
- Delta temperature is calculated as: Tleaf - Tair.
- Outputs values with consistent key names.

Output Keys:
------------
- `MI_battery_V`: Battery voltage in volts (float).
- `MI_payload_version`: Payload version (integer).
- `MI_value_01` ... `MI_value_16`: Raw sensor ratios or calculated temperatures.
- `LAT_B3_0n__Tleaf_degC`, `LAT_B3_0n__Tair_degC`, `LAT_B3_0n__dT_degC`:
  Temperatures and delta T for thermistor pairs n=1 to 4.

Thermistor Channel Pairing and Naming:
--------------------------------------

The thermistor values from the payload correspond to specific sensor channels
and are grouped in fixed pairs representing physical sensor pairs.

Each pair consists of two channels:

- Pair #1:
  - A0_#1 corresponds to value_03 (always the "air temperature" sensor)
  - A1_#1 corresponds to value_04 (always the "leaf temperature" sensor)

- Pair #2:
  - A2_#1 corresponds to value_05 (air temperature)
  - A3_#1 corresponds to value_06 (leaf temperature)

- Pair #3:
  - A0_#2 corresponds to value_07 (air temperature)
  - A1_#2 corresponds to value_08 (leaf temperature)

- Pair #4:
  - A2_#2 corresponds to value_09 (air temperature)
  - A3_#2 corresponds to value_10 (leaf temperature)

Important Notes:
----------------
```

```
- The assignment of Tair and Tleaf is fixed by these channel mappings,
  **not by comparing which value is numerically higher**.

- Delta temperature (dT) is always calculated as:
  dT = Tleaf - Tair

- The decoder outputs temperature values using the naming convention:
  LAT_B3_0n__Tleaf_degC, LAT_B3_0n__Tair_degC, LAT_B3_0n__dT_degC
  where n = 1 to 4 for the 4 channel pairs.

Usage:
------
- To disable temperature and delta calculations and output raw ratios only,
  set `CALCULATE_LAT_B3_TEMPERATURES_AND_DELTAS = false`.

- To enable full temperature decoding and delta calculation,
  set `CALCULATE_LAT_B3_TEMPERATURES_AND_DELTAS = true`.

*/
```