Tector Sensor Payload Decoding Guide

Woody v2 & Flatty — LoRaWAN Uplink and Downlink Format

Applies to	Tector Woody v2 and Tector Flatty sensors
Document version	1.1
Date	2026-07-10
Contact	Tector support — support@tector.com

---

1. Introduction

This document describes how to decode the data transmitted by **Tector Woody v2** and **Tector Flatty** sensors. Both devices use the **same payload format**, so a single decoder handles both.

The sensors are LoRaWAN devices. Each uplink message carries a binary payload (the LoRaWAN *FRMPayload*) containing one or more of the following:

- **Heartbeat** — device uptime, battery voltage, radio settings
- **Wood moisture (WMC)** — electrical resistance of the monitored material, from the built-in electrodes or an external probe
- **Environment** — ambient temperature and relative humidity, from the built-in sensor or an external probe
- **Announcement** — hardware and firmware version, sent alone in the first uplink after a device reset

Depending on your LoRaWAN network server, the payload is delivered to you either as **raw bytes** or encoded as a **Base64 string** (for example, the `frm_payload` field on The Things Stack) or a **hex string**. All examples in this document start from the Base64 form.

External-probe readings (`external_wmc`, `external_environment_data`) are available on devices running firmware version 5.0.0 or later with an external probe connected.

2. Payload format overview

The payload is a Protocol Buffers (**proto3**) message. Decoding takes three steps:

1. **Base64-decode** the payload string into raw bytes (skip this step if your network server already gives you raw bytes).
2. **Parse** the bytes as the `Uplink` protobuf message defined in section 3.
3. **Scale** the raw integer values into physical units (section 4) and check for special status values (section 5).

Because the format is standard Protocol Buffers, you can generate a decoder for virtually any programming language from the schema below — see section 7.

3. Uplink message definition

Save the following as `tector_uplink.proto`:

```
syntax = "proto3";

package uplink;

message Heartbeat {
    uint32 device_uptime = 1;
    uint32 battery_voltage = 2;
    uint32 mcu_temperature = 3;
    uint32 tx_power = 4;
    uint32 heartbeat_interval = 5;
}

message firmware_version {
    uint32 major = 1;
    uint32 minor = 2;
    uint32 patch = 3;
}

message Announcement {
    uint32 hardware_version = 1;
    firmware_version fw_version = 2;

    enum ResetReason {
        RESET_CAUSE_DEFAULT = 0;
        RESET_CAUSE_LOW_POWER_RESET = 1;
        RESET_CAUSE_WINDOW_WATCHDOG_RESET = 2;
        RESET_CAUSE_INDEPENDENT_WATCHDOG_RESET = 3;
        RESET_CAUSE_SOFTWARE_RESET = 4;
        RESET_CAUSE_EXTERNAL_RESET_PIN_RESET = 5;
        RESET_CAUSE_BROWNOUT_RESET = 6;
        RESET_CAUSE_UNKNOWN = 7;
    }
    ResetReason reset_reason = 3;

    enum ActiveRegion {
        ACTIVE_REGION_DEFAULT = 0;
        ACTIVE_REGION_EU = 1;
```

```
        ACTIVE_REGION_US = 2;
        ACTIVE_REGION_AU = 3;
        ACTIVE_REGION_AS = 4;
    }
    ActiveRegion active_region = 4;
}

message WMC {
    uint32 resistance_kohms = 1;
}

message EnvironmentData {
    int32 ambient_temperature = 1;
    uint32 ambient_humidity = 2;
}

message Uplink {
    Heartbeat heart_beat = 1;
    WMC wmc = 2;
    EnvironmentData environment_data = 3;
    Announcement announcement = 4;
    EnvironmentData external_environment_data = 5;
    WMC external_wmc = 6;
}
```

> **proto3 note — missing fields mean zero.** Proto3 does not transmit fields whose value is `0`. If a numeric field is absent from a decoded message, its value is `0`. For example, an `fw_version` containing only `major: 5` means firmware **5.0.0**, and a Heartbeat without `tx_power` means `tx_power = 0`.

What a given uplink contains

Every top-level field of `Uplink` is optional; a given uplink contains only the sections the device chose to send:

- The **very first uplink after the device (re)starts** contains only the `announcement` — no other sections are included in that uplink.
- A **normal periodic uplink** contains `heart_beat`, one WMC section, and one environment section (the internal or external variants, depending on configuration).
- When the radio link is weak, the device **omits sections to shrink the payload** and improve the odds of a successful transmission:

Signal quality (LoRa spreading factor)	Sections included
Good–moderate (SF7–SF10)	Heartbeat + WMC + environment
Poor (SF11)	WMC + environment (`heart_beat` omitted)
Very poor (SF12)	WMC only (`heart_beat` and environment omitted)

Your decoder should therefore treat every section as optional and handle uplinks where only some sections are present.

4. Field reference and unit scaling

Heartbeat (`heart_beat`)

Field	Raw type	Unit / conversion
`device_uptime`	uint32	Seconds since the device last started
`battery_voltage`	uint32	Battery voltage in **millivolts** (e.g. `3595` = 3.595 V)
`mcu_temperature`	uint32	Internal microcontroller temperature in °C (diagnostic; only present on some firmware versions)
`tx_power`	uint32	LoRa transmit power setting
`heartbeat_interval`	uint32	Configured transmission interval in **minutes**

Wood moisture (`wmc` / `external_wmc`)

Field	Raw type	Unit / conversion
`resistance_kohms`	uint32	Electrical resistance in **kΩ**. Divide by 1 000 for MΩ: `resistance_MΩ = resistance_kohms / 1000`

`wmc` is the reading from the device's built-in electrodes; `external_wmc` is the reading from an external moisture probe.

Before converting, check for status values — see section 5. An empty `wmc` / `external_wmc` message (or `resistance_kohms = 0`) means no valid reading was taken.

Environment (`environment_data` / `external_environment_data`)

Field	Raw type	Unit / conversion
`ambient_temperature`	int32 (signed)	Tenths of °C. Divide by 10: `raw 250` = **25.0 °C**. Negative temperatures are supported; standard protobuf `int32` decoding handles them.
`ambient_humidity`	uint32	Tenths of %RH. Divide by 10: `raw 311` = **31.1 %RH**

`environment_data` is from the built-in sensor; `external_environment_data` is from an external probe.

Announcement (`announcement`)

Field	Raw type	Meaning
`hardware_version`	uint32	Hardware revision number
`fw_version`	message	Firmware version as `major.minor.patch` (absent parts are `0`)
`reset_reason`	enum	Why the device restarted — see table below
`active_region`	enum	Configured LoRaWAN region — see table below

ResetReason

Value	Name	Meaning
0	`RESET_CAUSE_DEFAULT`	Normal start (default value)
1	`RESET_CAUSE_LOW_POWER_RESET`	Low-power reset
2	`RESET_CAUSE_WINDOW_WATCHDOG_RESET`	Window watchdog reset
3	`RESET_CAUSE_INDEPENDENT_WATCHDOG_RESET`	Independent watchdog reset
4	`RESET_CAUSE_SOFTWARE_RESET`	Software-initiated reset
5	`RESET_CAUSE_EXTERNAL_RESET_PIN_RESET`	External reset pin
6	`RESET_CAUSE_BROWNOUT_RESET`	Brownout (supply voltage dip)
7	`RESET_CAUSE_UNKNOWN`	Unknown cause

ActiveRegion

Value	Name	LoRaWAN region
0	`ACTIVE_REGION_DEFAULT`	Not reported
1	`ACTIVE_REGION_EU`	EU868
2	`ACTIVE_REGION_US`	US915
3	`ACTIVE_REGION_AU`	AU915
4	`ACTIVE_REGION_AS`	AS923

5. Special resistance status values

Three reserved values of `resistance_kohms` are **status codes, not measurements**. Check for them **before** dividing by 1 000:

Raw `resistance_kohms`	Meaning	Interpretation
`999999`	Resistance above measurable range	Material is very **dry**
`999998`	Measurement timed out	No reading — treat as missing data
`999997`	Resistance below measurable range	Material is very **wet**

A reading of `0` (or an entirely empty WMC message) also means no valid measurement.

If your application converts resistance to a moisture value, map these codes explicitly (e.g. "very dry", "no data", "very wet") instead of feeding them into the conversion.

6. Worked example

Payload as received (Base64):

```
CggI2zYQixwYFBIAGgYI+gEQtwIiBggLEgIIAQ==
```

Base64-decoded to 28 bytes (hex):

```
0a 08 08 db 36 10 8b 1c 18 14 12 00 1a 06 08 fa 01 10 b7 02 22 06 08 0b 12 02 08 01
```

Byte-by-byte protobuf breakdown:

```
0a 08                   Uplink field 1 (heart_beat), 8 bytes:
   08 db 36               device_uptime      = 7003      (varint)
   10 8b 1c               battery_voltage    = 3595      (varint)
   18 14                  mcu_temperature    = 20        (varint)
12 00                   Uplink field 2 (wmc), 0 bytes → empty (no reading)
```

```
1a 06                   Uplink field 3 (environment_data), 6 bytes:
   08 fa 01               ambient_temperature = 250      (varint)
   10 b7 02               ambient_humidity    = 311      (varint)
22 06                   Uplink field 4 (announcement), 6 bytes:
   08 0b                  hardware_version   = 11        (varint)
   12 02                  fw_version, 2 bytes:
      08 01                 major            = 1
```

Applying the unit scaling from section 4:

Field	Raw value	Physical value
Device uptime	7003	7 003 s (~1 h 57 min)
Battery voltage	3595	3.595 V
MCU temperature	20	20 °C
TX power	*(absent)*	0
Heartbeat interval	*(absent)*	0 (not reported)
Wood moisture	*(empty)*	No valid reading
Ambient temperature	250	25.0 °C
Ambient humidity	311	31.1 %RH
Hardware version	11	rev. 11
Firmware version	major 1	1.0.0

> **Note**: this example payload was recorded on an older firmware version that combined the announcement with the measurement sections. Current firmware sends the announcement alone in the first uplink of a session (see section 3).

Second example — external probe (firmware ≥ 5.0.0)

```
CgsIybUEEMMcIAIoPCoGCOsBENIFMgIIGA==
```

decodes to:

Field	Raw value	Physical value
Device uptime	72393	72 393 s (~20 h)
Battery voltage	3651	3.651 V
TX power	2	2
Heartbeat interval	60	60 min
External wood moisture	24 kΩ	0.024 MΩ
External ambient temperature	235	23.5 °C
External ambient humidity	722	72.2 %RH

Note that this uplink contains `external_wmc` (field 6) and `external_environment_data` (field 5) instead of the internal `wmc` and `environment_data` sections.

7. Decoding it yourself

Option A — `protoc` (any language)

With the protobuf compiler installed and the schema from section 3 saved as `tector_uplink.proto`, you can decode a payload directly from the command line:

```
echo "CggI2zYQixwYFBIAGgYI+gEQtwIiBggLEgIIAQ==" | base64 -d > payload.bin
protoc --decode=uplink.Uplink tector_uplink.proto < payload.bin
```

Output:

```
heart_beat {
  device_uptime: 7003
  battery_voltage: 3595
  mcu_temperature: 20
}
wmc {
}
environment_data {
  ambient_temperature: 250
  ambient_humidity: 311
}
announcement {
  hardware_version: 11
  fw_version {
    major: 1
```

```
  }
}
```

You can also generate native decoder bindings for your language of choice, for example:

```
protoc --python_out=. tector_uplink.proto   # Python
protoc --java_out=.   tector_uplink.proto   # Java
protoc --cpp_out=.    tector_uplink.proto   # C++
```

Option B — standalone Python script

The script below decodes a Base64 payload and applies all unit scaling and status-value handling from sections 4 and 5. It requires Python 3.10+ and two packages:

```
pip install protobuf grpcio-tools
python -m grpc_tools.protoc -I. --python_out=. tector_uplink.proto
```

Then save and run this script:

```
"""Decode a Tector Woody v2 / Flatty uplink payload.

Usage: python decode_tector_payload.py <base64-payload>
"""

import base64
import json
import sys

from google.protobuf.json_format import MessageToDict

from tector_uplink_pb2 import Uplink

RESISTANCE_STATUS_CODES = {
    999999: "resistance above range (very dry)",
    999998: "measurement timeout (no data)",
    999997: "resistance below range (very wet)",
}


def decode_payload(payload_b64: str) -> dict:
    uplink = Uplink()
    uplink.ParseFromString(base64.b64decode(payload_b64))
    data = MessageToDict(uplink, preserving_proto_field_name=True)
```

```
    # Temperature and humidity are transmitted as tenths of a unit
    for key in ("environment_data", "external_environment_data"):
        if key in data:
            for field in ("ambient_temperature", "ambient_humidity"):
                if field in data[key]:
                    data[key][field] /= 10.0

    # Resistance is transmitted in kOhms; reserved values are status codes
    for key in ("wmc", "external_wmc"):
        if key in data:
            raw = data[key].pop("resistance_kohms", 0)
            if raw in RESISTANCE_STATUS_CODES:
                data[key]["status"] = RESISTANCE_STATUS_CODES[raw]
            elif raw == 0:
                data[key]["status"] = "no valid reading"
            else:
                data[key]["resistance_mohms"] = raw / 1000.0

    return data


if __name__ == "__main__":
    print(json.dumps(decode_payload(sys.argv[1]), indent=2))
```

Example:

```
$ python decode_tector_payload.py "CggI2zYQixwYFBIAGgYI+gEQtwIiBggLEgIIAQ=="
{
  "heart_beat": {
    "device_uptime": 7003,
    "battery_voltage": 3595,
    "mcu_temperature": 20
  },
  "wmc": {
    "status": "no valid reading"
  },
  "environment_data": {
    "ambient_temperature": 25.0,
    "ambient_humidity": 31.1
  },
  "announcement": {
    "hardware_version": 11,
    "fw_version": {
      "major": 1
    }
```

```
  }
}
```

8. Downlink messages

Devices can be configured over LoRaWAN downlink. The downlink payload is also a proto3 message; save the following as `tector_downlink.proto`:

```
syntax = "proto3";

package downlink;

enum deviceCommand {
    DEVICE_COMMAND_NONE = 0;
    DEVICE_COMMAND_REBOOT = 1;
    DEVICE_COMMAND_TOGGLE_EXTERNAL_WMC = 2;
}

message deviceConfig {
    uint32 heartbeat_interval = 1;
}

message Downlink {
    deviceCommand command = 1;
    deviceConfig config = 2;
}
```

Field	Meaning
`command`	`DEVICE_COMMAND_NONE` (0) = no action, `DEVICE_COMMAND_REBOOT` (1) = restart the device, `DEVICE_COMMAND_TOGGLE_EXTERNAL_WMC` (2) = toggle between internal and external moisture measurement
`config.heartbeat_interval`	New transmission interval in **minutes**

Encode the message with protobuf, then Base64-encode the bytes if your network server expects Base64 (as The Things Stack does for scheduled downlinks).

Examples (verified encodings):

Downlink	Bytes (hex)	Base64
Set heartbeat interval to 15 min	`12 02 08 0f`	`EgIIDw==`
Reboot + set interval to 240 min	`08 01 12 03 08 f0 01`	`CAESAwjwAQ==`

Command-line example using `protoc`:

```
echo 'config { heartbeat_interval: 15 }' | \
  protoc --encode=downlink.Downlink tector_downlink.proto | base64
# EgIIDw==
```

If anything in this document is unclear or you need help decoding a specific payload, contact Tector support at support@tector.com and include the raw payload string (Base64 or hex) together with the device serial number.