NEON Temperature Transmitter

Communication Protocol v4

This document applies to products with protocol v4.

- For **DS-TT-02-00** from production batch **AA**

Contents

1 Introduction

1.1 Encoding & Decoding

The messages are sent over LoRa in a binary bytestring. A LoRa network server can decode raw bytestrings coming from the LoRa devices into JSON objects. It can also encode JSON objects back into bytestrings that form the payload of downlink messages.

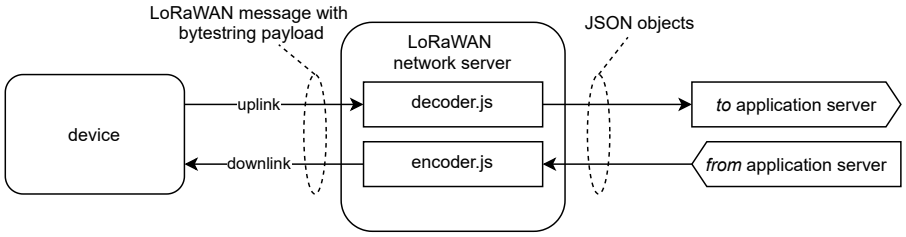

Figure 1: Decoding/encoding

1.2 Document Content

The LoRaWAN specification provides Port Field (FPort) to differentiate between various types of messages. This document is organized based on the utilized FPort where each section describes all the messages associated to a given FPort. Sections 2, 3, 4, 5, and 6 which correspond to Fports 1, 2, 3, 5 and 6, respectively, only include one uplink message type per FPort. However, Section 7 (Fport 7) covers different types of configuration messages.

The following table provides an Overview of all the messages and also provides information on the fPort used per message.

FPort	Section	Message Name	Up/Down	Purpose
1	2	Boot	up	Informs on device reboot
2	3	Device status	up	Informs on device health, battery info, counter for statistics, etc. It is sent on button press, or on a timer.
3	4	Sensor event	up	Informs on the temperature. It is sent on event detection, button press, or on a timer.
5	5	Activated	up	Indicates that a device is activated and operational.
6	6	Deactivated	up	Indicates that a device is deactivated.
7	7	Base configuration	down/up[1]	Configures the device with radio setting.
7	7	Sensor configuration	down/up	Configure the sensor specific settings.
7	7	Sensor conditions configuration	down/up	Configures the event conditions.

Table 1: Message Overview

[1] A downlink request and an uplink answer

1.3 Important Remarks

- The protocol version for the current document is 4.
- All variables are little endian.
- To participate in a LoRaWAN network, all devices use *Over-The-Air Activation* (OTAA) rather than *Activation by Persolalization* (ABP).

In the following sections, each message is described and how it is encoded in the bytestring. The Decoder / Encoder is available on the product support page. If using this Decoder / Encoder, the bytestring information below can be ignored.

1.4 Button press

The user can trigger device uplink messages by pressing the button while the device is active. By default the device will send both device status (see Section 3) and sensor event (see Section 4) messages.

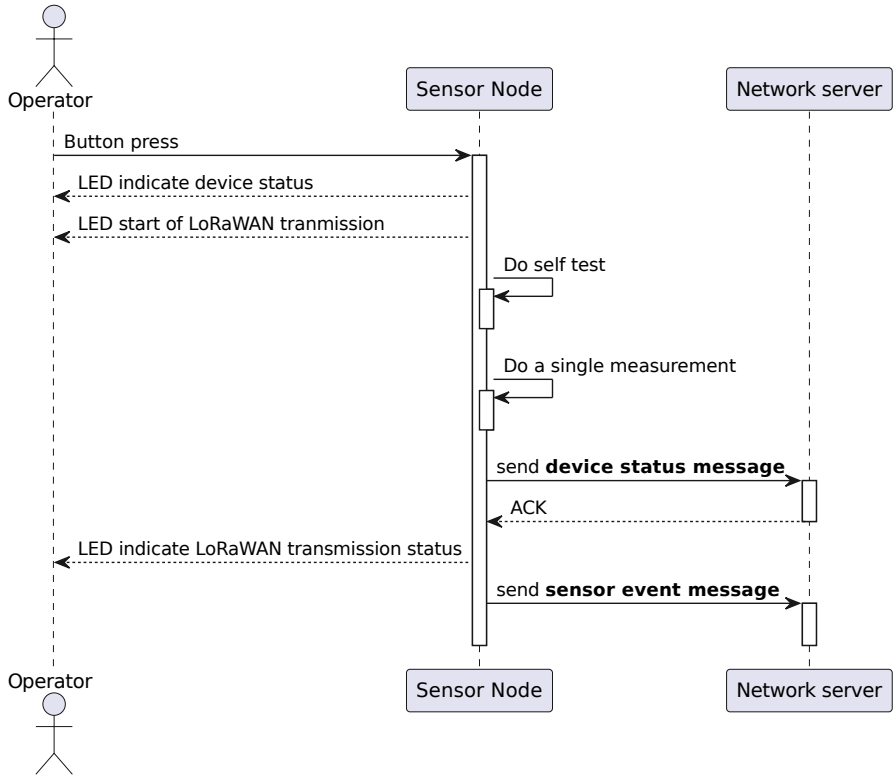

Figure 2: Button Press Scheme

2 Boot Information

The ***Boot Message*** is sent using FPort 1 when the device is used for the first time or when a reboot occurs. Reboots might occur intended in case of activating a newly received configuration. Or the device might reboot due to a system error as a matter to recovery (e.g. continuous communication failures, unforeseen situations, and others). The boot message contains information about why it has (re)booted. Typically this information can be ignored and is only used for solving problems in the field. During normal operation and with sufficient network quality reboots seldomly occur, other than activating a newly received configuration.

2.0.1 JSON Structure

```
{
    "boot": {
        "protocol_version": <json string>,
        "reboot_reason": {
            "major": <json string>,
            "minor": <json string>
        }
    }
}
```

Listing 1: Boot JSON Structure

2.0.2 Binary encoding and description

Description	Binary encoding	Byte index
`protocol_version` The version of the protocol.	uint8-bits[4..7]	0
`reboot_reason` Provides information on transmitter's reboot reason.	`reboot_reason` See section 2.0.3	1

Table 2: Boot message description

2.0.3 Reboot reason

This field provides information about device's reboot reason. The reason are divided into two parts, major and minor. Minor reason provides details on the major reason.

Description	Binary encoding	Byte index

`reboot_reason`
 	
`reboot_reason.major_reason`
There are 7 possible values of major reasons:
• "none"
• "config update"
• "firmware update"
• "button reset"
• "power"
• "communication failure"
• "system failure"
	uint8
	
	`none = 0`
	`config_update = 1`
	`firmware_update = 2`
	`button_reset = 3`
	`power = 4`
	`comm_failure = 5`
	`system_failure = 6`
		0
		
		bits[0..3]

Description	Binary encoding	Byte index
`reboot_reason.minor_reason` Minor reasons will be described in Table 4.		bits[4..7]

Table 3: Reboot reason binary encoding

Reboot reason description
This section provide detailed description of reboot reason.

Major	Minor	Description
`"none"`	-	The particular device has no reason to send a boot message. The counterpart (transmitter or sensor) may trigger the boot message.
`"config update"`	-	Boot message is sent due to successful configuration update.
`"firmware update"`	 `"success"` (0) `"rejected"` (1) `"error"` (2) `"in progress"` (3)	Boot message is triggered by firmware update. Firmware update is successful. Update image is rejected. Error occured during firmware update. Firmware update is in progress.
`"button reset"`	-	Transmitter rebooted due to manual button reset.
`"power"`	 `"black out"`[2] (0) `"brown out"`[3] (1) `"power safe state"` (2)	Device rebooted due to power issues. Device detects low power issue. Device detecs brown out reset. Device recovered from power safe state[4].
`"communication failure"`	-	On persistent LoRaWAN communication failure, the device will reboot to try to recover communication.
`"system failure"`	-	Reboot in order to try to recover from any unforeseen mistake in either hardware or firmware.

Table 4: Reboot reason description

[2] Completely intruption of system power
[3] Temporary drop in system voltage
[4] On too many reboots due to power issues in a short period, the device will halt for 1 hour to avoid bootloops.

3 Device Status

Periodically a ***Device status message*** is sent using FPort 2. Timing is independent from the temperature measure timer. The message contains information for maintenance, device health and other analysis. ***Device status message*** is also sent on short button press.

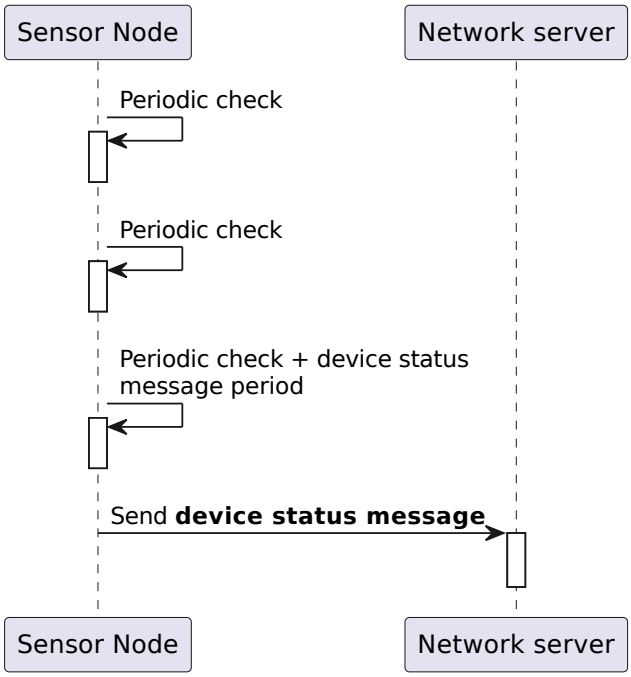

Figure 3: Device Status Scheme

3.0.1 JSON Structure

```
{
    "device_status": {
        "protocol_version": <json number>,
        "battery_voltage": <json number>,
        "temperature": <json number>,
        "lora_tx_counter": <json number>,
        "avg_rssi": <json string>,
        "bist": <json string>,
        "event_counter": <json number>,
        "sensor_type": <json string>
    }
}
```

Listing 2: Device Status JSON Structure

3.0.2 Binary encoding and description

Description	Binary encoding	Byte index
`protocol_version` The version of the protocol.	uint8-bits[4..7]	0
`battery_voltage` Voltage ratio, meant as input for battery charge estimation in the backend. To feed the model on the backend, the voltage measured on the low load is sent. Range: 2.000 V .. 4.000 V	uint8 $\frac{2}{255}$ V per LSB The battery voltage is encoded where 0 is the minimum of 2.000 V and 255 is the maximum of 4 V.	1
`temperature` Base temperature in units of 1 °C. It is reported in the average temperature since the last device status message. The number of temperature measurements can be configured. Range: –50 °C .. 125 °C (–58 °F .. 257 °F)	int8 1 °C per LSB ([°F] = $\frac{9}{5}$ [°C] + 32)	2
`lora_tx_counter` Number of LoRa transmissions since the last device status message. It is reset after the device status message is sent. If the number of LoRa transmissions becomes greater than 65535 this field is set to 65535.	uint16	3,4

Description	Binary encoding	Byte index
`avg_rssi` The average RSSI of received messages since the last device status message. This configuration is applied in the base device. Range: • "0 .. –79" • "–80 .. –99" • "–100 .. –129" • "<-129"	uint8 "0 .. –79" = 0 "–80 .. –99" = 1 "–100 .. –129" = 2 "<-129" = 3	5
`bist` A bitmask with the result of the build in self test. At boot and also before sending a status message the device performs a self test in order to verify the working of essential components. This can be used for analysis when device is not working properly. byte value: • 0x7F (127) : Self-tests of base are OK, it is activated and the paired sensor is connected • 0x1F (31) : Failure in the measurement circuit • 0x3F (63) : No sensor is connected • 0x6F (111) : Base is deactivated • Other values : Base self test failed bit value: • 0: test failed • 1: test succeeded bit 0: battery measurement bit 1: base flash memory bit 2: lora module bit 3: configuration (provisioning) bit 4: activation bit 5: sensor measurement circuit bit 6: sensor probe connection bit 7: reserved	uint8	6

Description	Binary encoding	Byte index
`event_counter` Counter for number of events. It counts the event messages triggered by condition changes. The periodic and button press event messages are not counted in here. This can be used to detect missed condition changes. It is reset on boot and after device status message transmission. If the number of event messages triggered by condition chagne between two consecutive device status messages exceeds 255 , this value remains 255. Range: 0 .. 255 After 255 it will wrap to 1. Zero value is reserved for after boot as well as when there has been no event message triggered by condition change between the device status message.	uint8	7

`sensor_type`
• "PT100"
• "J"
• "K"
• "T"
• "N"
• "E"
• "B"
• "R"
• "S"
	uint8
	
	`PT100 = 0`
	`J = 1`
	`K = 2`
	`T = 3`
	`N = 4`
	`E = 5`
	`B = 6`
	`R = 7`
	`S = 8`
		8

Table 5: Device Status message description

4 Sensor Event

The temperature is measured periodically. When a condition change is detected a ***Sensor event message*** is sent using FPort 3. ***Sensor event message*** can also be triggered by timer, or on short button press. **Sensor Event message** can be sent in *normal* or *extended mode*. The message is sent in *extended mode* if `selection` is set to `extended` where `selection` is determined based on the switch mask provided by the **Sensor Configuration message** described in section 7.2.2.

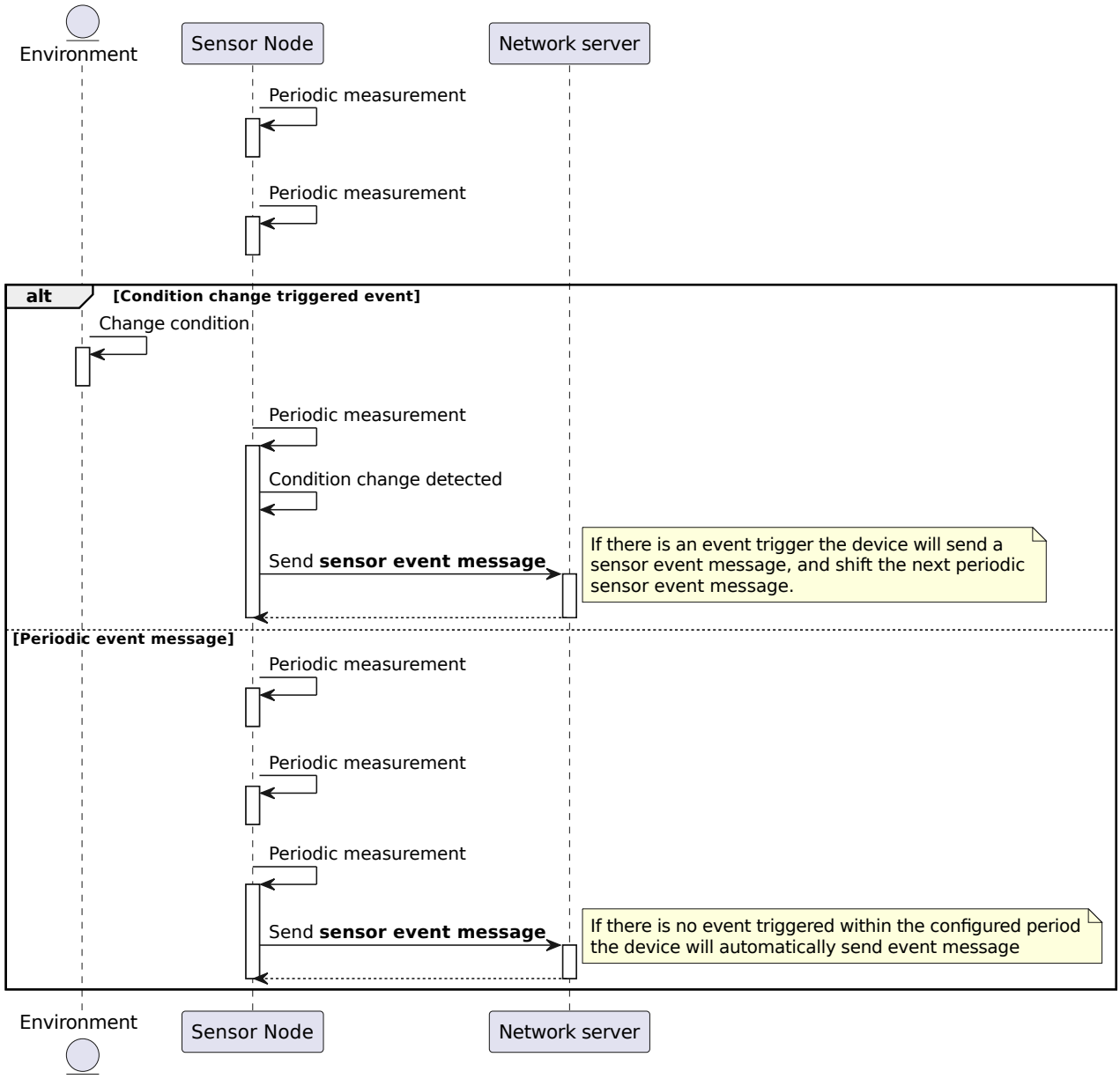

Figure 4: Event Message Scheme

4.0.1 JSON Structure

Normal Mode:

```
{
    "sensor_event": {
        "protocol_version": <json string>,
        "selection": <json string>,
        "condition_0": <json number>,
        "condition_1": <json number>,
        "condition_2": <json number>,
        "condition_3": <json number>,
        "trigger": <json string>,
        "temperature": {
            "min": <optional json number>,
            "max": <optional json number>,
            "avg": <optional json number>,
            "status": <json string>
        }
    }
}
```

Listing 3: Normal Sensor Event JSON structure

4.0.2 Binary encoding and description

Event Common:

Description	Binary encoding	Byte index
`protocol_version` The version of the protocol.	uint8-bits[4..7]	0

`selection`
 	
Indicates if the message is sent in *extended mode* or in *normal mode*. In *normal mode*, either min, max, or, avg is reported. String with the possible values:
• "extended"
• "min_only"
• "max_only"
• "avg_only"
	uint8
	
	`extended = 0`
	`min_only = 1`
	`max_only = 2`
	`avg_only = 3`
		1

Description	Binary encoding	Byte index

`condition_n`
The current state of each condition.
• True condition is represented by 1
• False condition is represented by 0
 	
`trigger`
String with the possible values:
• "condition change"
• "periodic"
• "button press"
	uint8
	
	bit0 = condition_0
	bit1 = condition_1
	bit2 = condition_2
	bit3 = condition_3
	bits[6..7]
	
	`condition_change = 0`
	`periodic = 1`
	`button_press = 2`
		2

Measurement values as described in table 7 and table 8 for *normal mode* and *extended mode*, respectively.		

Table 6: Sensor Event message description common

Event Normal:

Description	Binary encoding	Byte index
`temperature` The connected sensor temperature reported in the min, max, or avg (determined by `selection`) temperature since the last event message. Range: –270.0 ℃ .. 1850.0 ℃ (–454.0 ℉ .. 3362.0 ℉)	int16 0.1 ℃ per LSB ([℉] = $\frac{9}{5}$ [°C] + 32) If no error, then `status` will be set to `OK`. In case of an error the `status` will indicate an *Error Code* (See Table 9) and the `min`, `max`, and `avg` are removed from the JSON.	3,4

Table 7: Sensor Event Binary Encoding

Event Extended:

Description	Binary encoding	Byte index
`temperature` The connected sensor temperature reported in the min, max and avg temperature since the last event message. Range: –270.0 °C .. 1850.0 °C (–454.0 °F .. 3362.0 °F)	int16[3] 0.1 °C per LSB ([°F] = $\frac{9}{5}$ [°C] + 32) If no error, then `status` will be set to `OK`. In case of an error the `status` will indicate an *Error Code* (See Table 9) and the `min`, `max`, and `avg` are removed from the JSON.	3..8

Table 8: Sensor Event Binary Encoding Extended

Error Codes

Error	Description	Binary encoding
`Hardware Error`	There is a problem with the hardware.	0x7FFF
`PT100 Upper Bound Error`	The temperature sensor reading is higher than expected.	0x7FFE
`PT100 Lower Bound Error`	The temperature sensor reading is lower than expected.	0x7FFD

Table 9: Error Codes Binary Encoding

5 Activation Process

Activation of the Temperature Transmitter is initiated by the operator. After activation an ***Activation Message*** is sent using FPort 5. It is an indication that the device is operational. The full sequence is depicted below.

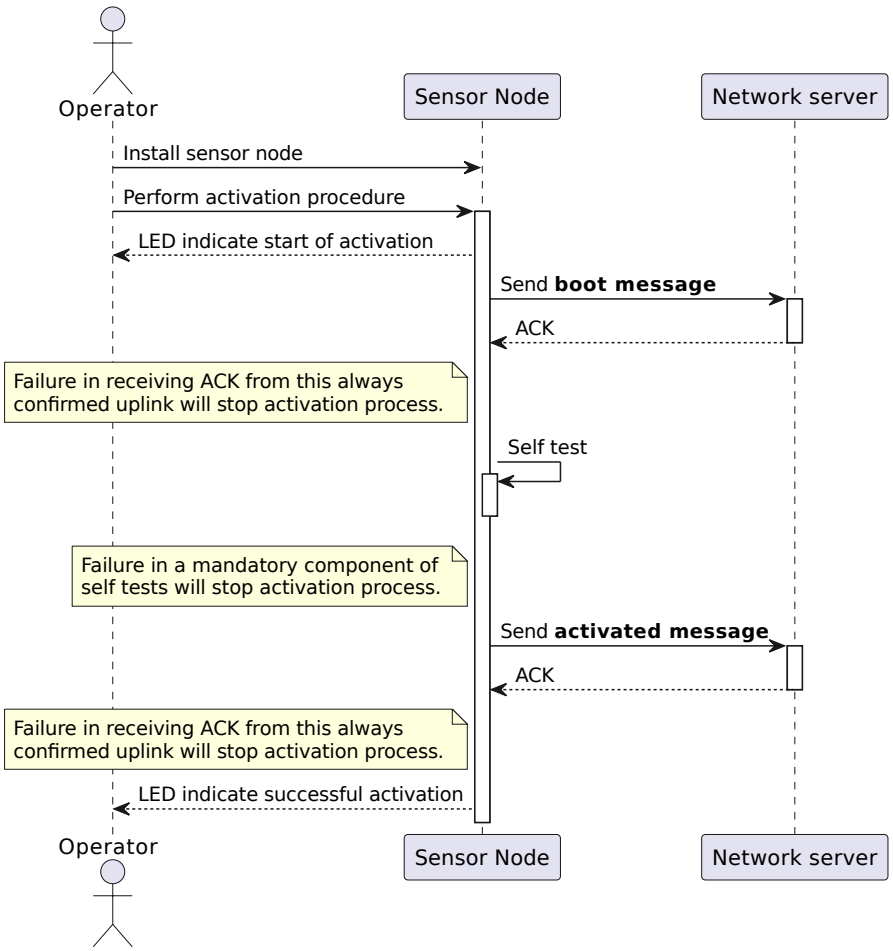

Figure 5: Activation Sequence Scheme

In case of an error during activation, the procedure is aborted. The error situations can be as followed:

1. No LoRa confirmation message from network server is received: no LoRa coverage
2. Failure in one of the activation actions

5.0.1 JSON Structure

```
{
    "activated": {
        "protocol_version": <json number>,
        "device_type": <json string>
    }
}
```

Listing 4: Activation JSON Structure

5.0.2 Binary encoding and description.

Description	Binary encoding	Byte index
`protocol_version` The version of the protocol.	uint8-bits[4..7]	0
`device_type` The device type as known in the NEON family. For this device, Temperature Transmitter the value is: • "tt"	uint8 `tt = 4`	1

Table 10: Activated Message Binary Encoding

6 Deactivation Process

The operator can deactivate by a long press of the button. This will result in a ***Deactivation Message*** sent using FPort 6. It is an indication that the device is not operational.

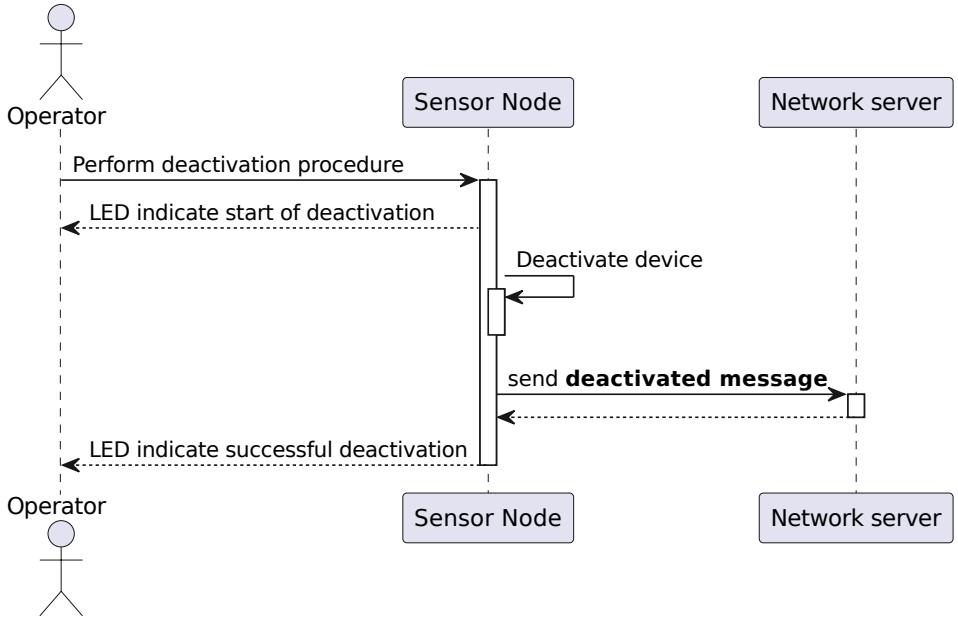

Figure 6: Deactivation Message Scheme

6.0.1 JSON Structure

```
{
    "deactivated": {
        "protocol_version": <json number>,
        "reason": <json string>
    }
}
```

Listing 5: Deactivation JSON Structure

6.0.2 Binary encoding and description

Description	Binary encoding	Byte index
`protocol_version` The version of the protocol.	uint8-bits[4..7]	0
`reason` The reason why the sensor is deactivated. String with the possible values: • "user_triggered" A user performed the deactivation process	uint8 `user_triggered = 0`	1
`reserved` Allways 0.	uint8	2

Table 11: Deactivated Message Binary Encoding

7 Configuration Mechanism

7.1 Configuration Messages

The device can be configured with a configuration downlink. Different configurations are supported by the device to configure the non sensor related behavior (See Table 14), sensor related behavior (see Table 15), and to set the conditions which trigger the event message (See Table 16). All configuration messages are sent using FPort 7.

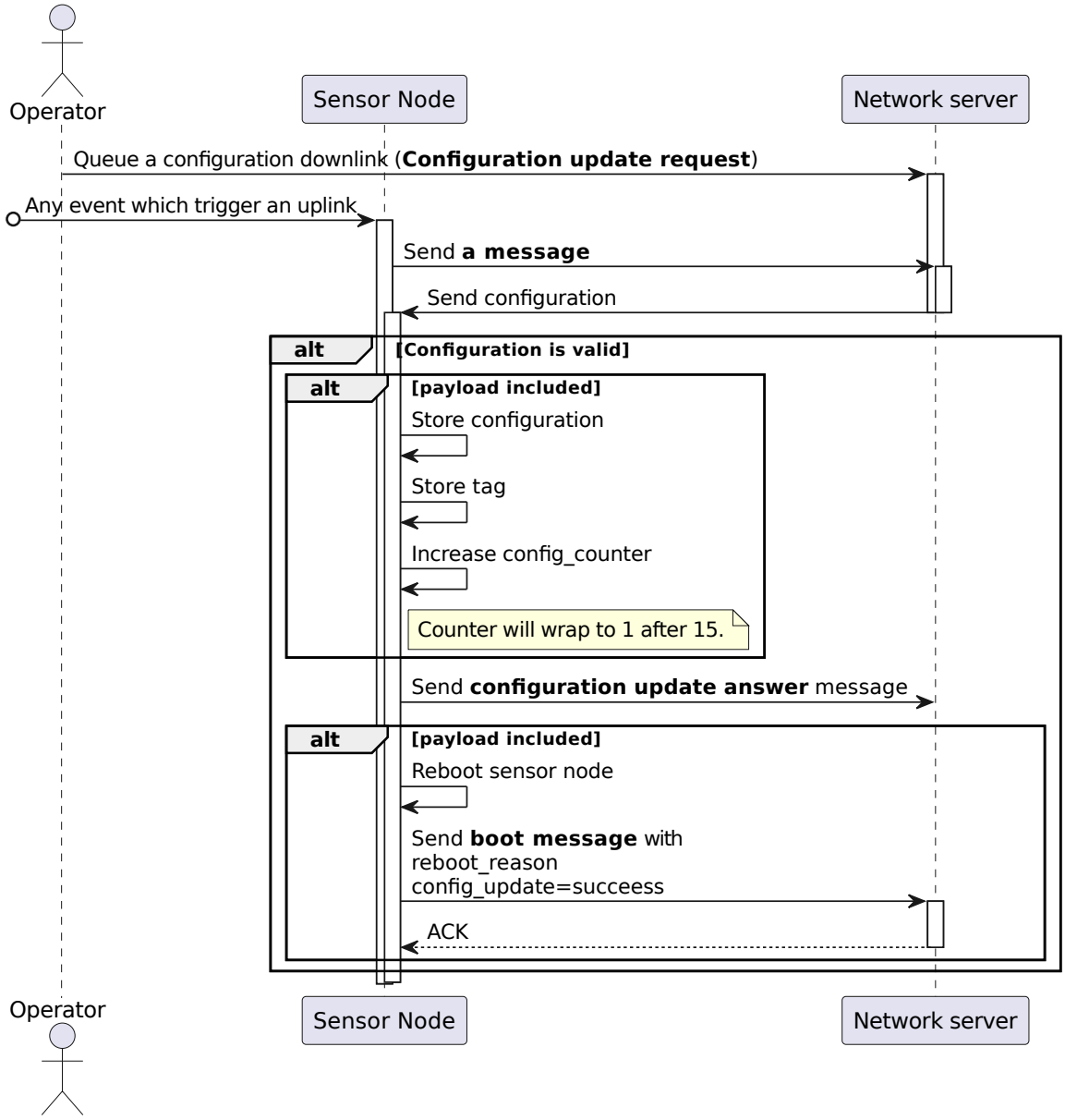

Figure 7: Single Configuration Update Scheme

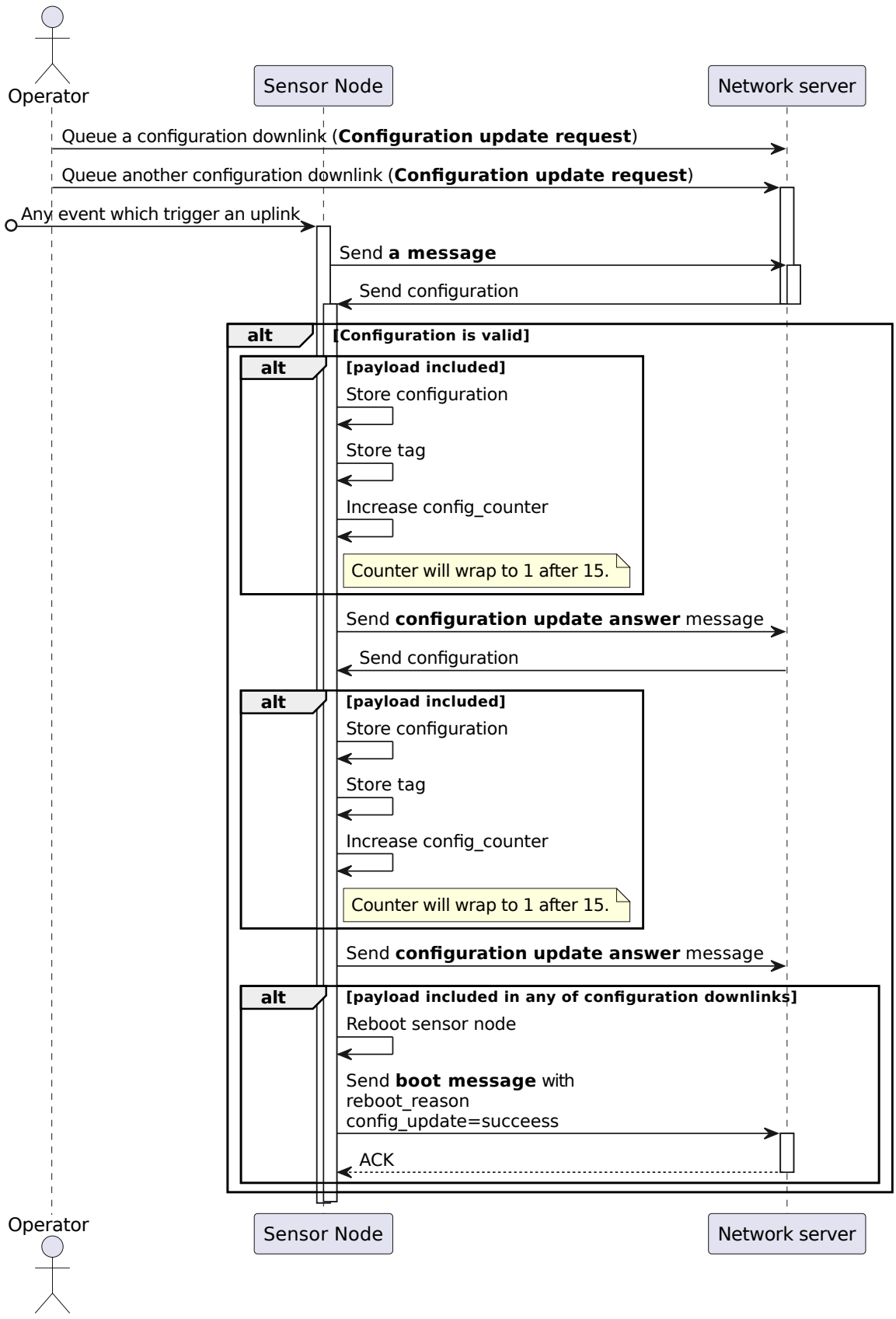

Figure 8: Multiple Configuration Update Scheme

7.1.1 Configuration Update Request

A **Configuration Update Request** is sent by the user to:

- Change a configuration of the device:

  In this case, the **Configuration Update Request** includes a *payload* as well as an associated *tag*. The ***tag*** is assigned to each configuration by the user to discriminate between the configurations. This `tag` can be e.g., a hash or an ID. Moreover, the device assigns a ***counter*** to each configuration type. The `counter` keeps on increasing every time a configuration is updated. The `tag` and the `counter` will be saved on the device alongside the configuration itself and can be requested by the user as explained in the following.

- Get the `counter` and `tag` of the current configuration on the device (See Table 13) without changing the configuration:

  In this case, the *payload* and *tag* are not required.

After a valid **Configuration Update Request** with payload, the device reboots. However, if multiple **Configuration Update Request** messages are sent consecutively, the device only reboots once for all of them.

JSON Structures

```
{
    "config_update_req": {
        "config_type" <json string>,
        "protocol_version": <json number>,
        "tag": <optional json number>,
        "payload": <optional json object>
    }
}
```

Listing 6: Configuration Update Request JSON Structure

Binary encoding and description

Description	Default value	Binary encoding	Byte index

`config_type`
The type of the configuration.
 	
• "base"
• "sensor"
• "sensor_data"
• "sensor_conditions"
 	
Not all the configuration types are supported by the current device. Check the rest of the current section to see the supported configuration types.
 	
`protocol_version`
The version of the protocol.
	-
	
	-
		bits[0..3]
		
		`base = 0`
		`sensor = 3`
		`sensor_data = 4`
		`sensor_conditions = 5`
		
		bits[4..7]
			0

`tag` **Note**: Optional, not used in cases of only requesting the tag and counter. A 4 bytes value that can be assigned by the user to identify the configuration.		uint8[4]	1..4

Description	Default value	Binary encoding	Byte index
`payload` **Note**: Optional, not used in cases of only requesting the tag and counter. A configuration payload which can be any length as long as it fits the LoRa payload. Configuration payload is `config_type` specific. See the Section 7.2 for more information.			

Table 12: Configuration update request message description

7.1.2 Configuration Update Answer

A **Configuration Update Answer** is always sent by the device in response to **Configuration Update Request**.

- If the **Configuration Update Request** included a valid `payload`, **Configuration Update Answer** includes the `tag` and `counter` of the new received configuration.
- If the **Configuration Update Request** does not include the `pyaload`, **Configuration Update Answer** includes the `tag` and `counter` of the current configuration of the device associated to the `config_type`.

JSON Structure

```
{
    "config_update_ans": {
        "config_type": <json string>,
        "protocol_version": <json number>,
        "tag": <json number>,
        "counter": <json number>
    }
}
```

Listing 7: Configuration Update Answer JSON Structure

Binary encoding and description

Description	Default value	Binary encoding	Byte index

`config_type`
The type of the configuration as described in Table. 12.
 	
`protocol_version`
The version of the protocol.
	-
	
	-
		bits[0..3]
		
		bits[4..7]
			0

`tag` The tag of the current configuration. It is a 4 bytes value that is assigned by user to identify the configuration.		uint8[4]	1..4
`counter` A value which keep on increasing everytime a configuration is updated. After 15, the value is wrapped to 1. 0 is reserved for value right after production. Note that `counter` does not increase if the payload of the configuration update request is empty. Each configuration type has its own counter.		uint8	5

Table 13: Configuration update answer message description

7.2 Configuration Types

7.2.1 Base Configuration

In the base configuration, the non sensor related behavior can be configured. Changing these parameters will have an effect on battery life and quality of service.

Payload JSON Structure

```
{
    "payload": {
        "switch_mask": {
            "enable_confirmed_event_message": <json string>,
            "enable_confirmed_data_message": <json string>,
            "allow_deactivation": <json string>
        },
        "periodic_message_random_delay_seconds": <json number>,
        "status_message_interval": <json string>
    }
}
```

Listing 8: Base Config Payload JSON Structure

Binary encoding and description

Description	Default value	Binary encoding	Byte index
`config_type`, `protocol_version`, and `tag` as described in Table. 12 with `configuration_type = "base"`.			0..4

`switch_mask`
Booleans to turn features on or off.
 	
`enable_confirmed_event_message`
Enable confirmed sensor event message
 	
`enable_confirmed_data_message`
Enable confirmed data message
 	
`allow_deactivation`
Allow local deactivation procedure to be executed for either disabling the sensor or replacing it with a new sensor.
 	
`reserved`
	
	false
	
	false
	
	true
	
	0
	
	Binary (hex): 0x04
		uint8 (bitmask:)
		
		bit[0]
		
		bit[1]
		
		bit[2]
		
		bits[3..7] reserved
			5

		uint8	6

Description	Default value	Binary encoding	Byte index
`periodic_message_random_delay_seconds` To avoid clustering and collisions of uplink transmissions of multiple devices a random delay is added to periodic messages (device status message and timer triggered event message). Range: 0 s .. 31 s	30	bits[0..4]	

`status_message_interval`
Interval between two periodic device status messages, which can be picked from one of the following options.
 	
Available options:
• "2 minutes"
• "15 minutes"
• "1 hour"
• "4 hours"
• "12 hours"
• "1 day"
• "2 days"
• "5 days"
	"1 day"
	
	Binary (hex): 0xBE
		bits[5..7]
		
		`2_minutes = 0`
		`15_minutes = 1`
		`1_hour = 2`
		`4_hours = 3`
		`12_hours = 4`
		`1_day = 5`
		`2_days = 6`
		`5_days = 7`	

Table 14: Base Configuration message description

7.2.2 Sensor Configuration

Payload JSON Structure

```
{
    "payload": {
        "device_type": <json string>,
        "sensor_type": <json string>,
        "switch_mask": {
            "selection": <json string>
        },
        "measurement_interval_minutes": <json number>,
        "periodic_event_message_interval": <json number>
    }
}
```

Listing 9: Sensor Config Payload JSON Structure

Binary encoding and description

Description	Default value	Binary encoding	Byte index
`config_type`, `protocol_version`, and `tag` as described in Table. 12 with `configuration_type = "sensor"`.			0..4
`device_type` The device type as known in the NEON family. For this device the value is: "tt" (4)	"tt" Binary (hex): 0x06	uint8 "tt" = 4	5

`sensor_type`
Sensor type selection:
• "PT100"
• "J"
• "K"
• "T"
• "N"
• "E"
• "B"
• "R"
• "S"
	"K"
	
	Binary (hex): 0x02
		uint8
		
		`PT100 = 0`
		`J = 1`
		`K = 2`
		`T = 3`
		`N = 4`
		`E = 5`
		`B = 6`
		`E = 7`
		`B = 8`
			6

Description	Default value	Binary encoding	Byte index

`switch_mask`
 	
`switch_mask.selection`
Indicates if the event message is sent in *extended mode* or in *normal mode*. In *normal mode*, either min, max, or, avg is reported. String with the possible values:
• "extended"
• "min_only"
• "max_only"
• "avg_only"
 	
`reserved`
	
	avg_only
	
	Binary (hex):
	0x03
		uint8
		
		bits[0..1]
		
		`extended = 0`
		`min_only = 1`
		`max_only = 2`
		`avg_only = 3`
		
		bits[2..7]
			7

`measurement_interval_minutes` Interval in minutes, at which the temperature sensor is read. Changing this value has an effect on responsiveness and battery life. Range: 1 min .. 240 min (4 hours)	5 Binary (hex): 0x05	uint8 1 minute per LSB	8
`periodic_event_message_interval` Interval in the number of measurements at which the sensor event messages are periodically sent. The periodic counter is reset on every event message. Range: 0 measurements .. 10 080 measurements Example setups of periodic event message interval: • once per 1 hours (default): set this parameter to 12 (default) and `measurement_interval_minutes` to 5 min • once per 8 hours: set this parameter to 32 and `measurement_interval_minutes` to 15 min • Setting `periodic_event_message_interval` to zero disables the periodic sensor event message.	12 Binary (hex): 0x0C 0x00	uint16 1 measurement per LSB	9..10

Table 15: Sensor Configuration message description

7.2.3 Sensor Conditions Configuration

As stated in Section 4, an event message is triggered on timer, button press, or on *condition change*. This configuration sets the *conditions* which can trigger the event message.

JSON Structure

```
{
    "payload":
    {
        "device_type": <json string>,
        "event_conditions": [
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "threshold_temperature": <json number>
            },
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "threshold_temperature": <json number>
            },
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "threshold_temperature": <json number>
            },
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "threshold_temperature": <json number>
            }
        ]
    }
}
```

Listing 10: Sensor Condition Paylaod Configuration JSON structure

Binary Encoding and Description

Description	Default value	Binary encoding	Byte index
`config_type`, `protocol_version`, and `tag` as described in Table. 12 with `configuration_type = "sensor_conditions"`.			0..4
`device_type` The device type as known in the NEON family. For this device the value is: • "tt"	"tt" Binary (hex): 0x04	uint8 `tt = 4`	5

Description	Default value	Binary encoding	Byte index
`event_conditions` Configuration of four independent event_conditions. See Temperature Event Configuration Binary Encoding	[{"mode":"off"}, {"mode":"off"}, {"mode":"off"}, {"mode":"off"}] Binary (hex): 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	event[4]	6..17

Table 16: Sensor Conditions Configuration Binary Encoding

Binary encoding and description Temperature event config

Description	Default value	Binary encoding	Byte index

`mode`
The operational mode of this event:
• "off"
• "above"
• "below"
• "increasing"
• "decreasing"
 	
off mode: Disables condition detection. If `mode` is set to `off`, then `measurement_window` would be considered to be 0. If `measurement_window` is equal to 0, condition detection is disabled regardless of the selected `mode`.
 	
above mode: If the maximum temperature along the **measurement_window** is above the **temperature_threshold** then the condition is **true**. A transition of the condition from false to true or true to false will trigger an event message.
 	
below mode: If the minimum temperature along the **measurement_window** is below the **temperature_threshold** then the condition is **true**. A transition of the condition from false to true or true to false will trigger an event message.
 	
increasing mode: The condition is **true** when the current temperature is at least **temperature_threshold** higher than the minimum temperature in the **measurement_window**. A transition of the condition from false to true or true to false will trigger an event message.
*Note that in **increasing** mode the accepted configuration for **threshold_temperature** is only positive.*
 	
decreasing mode: The condition is true when the current temperature is at least **temperature_threshold** lower than the maximum temperature in the **measurement_window**. A transition of the condition from false to true or true to false will trigger an event message.
*Note that in **decreasing** mode the accepted configuration for **threshold_temperature** is only negative.*
	0x00
	
	"off"
		uint8
		
		bits[0..1]
		
		`above = 0`
		`below = 1`
		`increasing = 2`
		`decreasing = 3`
			0

Description	Default value	Binary encoding	Byte index
`measurements_window` The maximum number of measurements to observe delta temperature to trigger an event. For mode "off" the default value is 0. For all other modes: Range: 1 measurements .. 63 measurements	0 Binary (hex): 0x00	bits[2..7]	
`threshold_temperature` The threshold temperature in units of 0.1 °C. In mode *increasing* and mode *decreasing*: Only *positive*, Range 0.0 .. 2120.0 °C (32.0 °F .. 3848.0 °F) In mode *above* and mode *below*: Range –270.0 .. 1850.0 °C (–454.0 °F .. 3362.0 °F)	0 Binary (hex): 0x00 0x00	int16 0.1 °C per LSB ([°F] = $\frac{9}{5}$ [°C] + 32)	1..2

Table 17: Temperature Event Configuration Binary Encoding

Revision History

Revision	Date	Author(s)	Description
A1	07-12-2022	NY, TB, ML	Created protocol V4 based on protocol V3. - Devided sensor configuration into sensor configuration and sensor condition configuration - Devided base configuration into base configuration and region configuration (Extra functionalities) - Added extended and debug mode - Allocated 1 byte for `reboot_reason` - Utilized different fports for different type of messages - Restructured the communication document based on the utilized fports - New configuration container, which allows to check the current stored config tag
A2	03-01-2023	NY	Added US units