NEON Ratiometric Transmitter

Communication Protocol v1

This document applies to the following range of products with protocol v1:

- For **DS-RT-02-xx** 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.

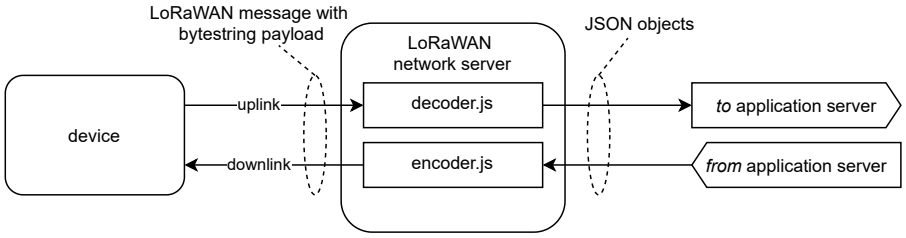

Figure 1: Decoding/encoding

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, 6, and 8 which correspond to Fports 1, 2, 3, 5, 6, and 8, 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 sensor measurement. 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 basic device settings.
7	7	Sensor configuration	down/up	Configure the sensor specific settings.
7	7	Sensor conditions configuration	down/up	Configures the event conditions.
7	7	User calibration configuration	down/up	Calibrates the measurements.
8	8	uncalibrated measurement	up	Informs of last sensor measurement before user calibration being applied.

Table 1: Message Overview

[1] A downlink request and an uplink answer

1.3 Important Remarks

- The protocol version for the current document is 1.
- 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).
- The units are in Metric system. However, USCS (United States Customary System) units are written within brackets (()).
- The Decoder / Encoder is available on the product support page.
- The USCS Decoder / Encoder is also available on the product support page and has a '-us' postfix.

In the following sections, each message is described and how it is encoded in the bytestring. If using one of the provided Decoders / Encoders, the bytestring information below can be ignored.

1.4 Specific Binary Encoding Formats

Apart from the common binary encoding formats, *e.g.,* uint8, another encoding format, *float32*, is used as indicated in the following table:

Binary encoding	Description	Size (Bytes)
float32	Single-precission IEEE Standard 754 floating-point reference: https://en.wikipedia.org/wiki/IEEE_754	4

Table 2: Custom Binary Encoding

1.5 Measurement Unit

This document applies to a range of products as depicted in the table below. The sensor measurement unit varies according to the given product as follows:

Product	Metric Unit	USCS unit
pressure sensor (PS)	1 bar	14.5038 psi

Table 3: Sensors Units

1.6 Button press

The user can trigger device uplink messages by pressing the button while the device is active. By default the device will send device status (see Section 3), sensor event (see Section 4), and uncalibrated measurement (see Section 8) messages.

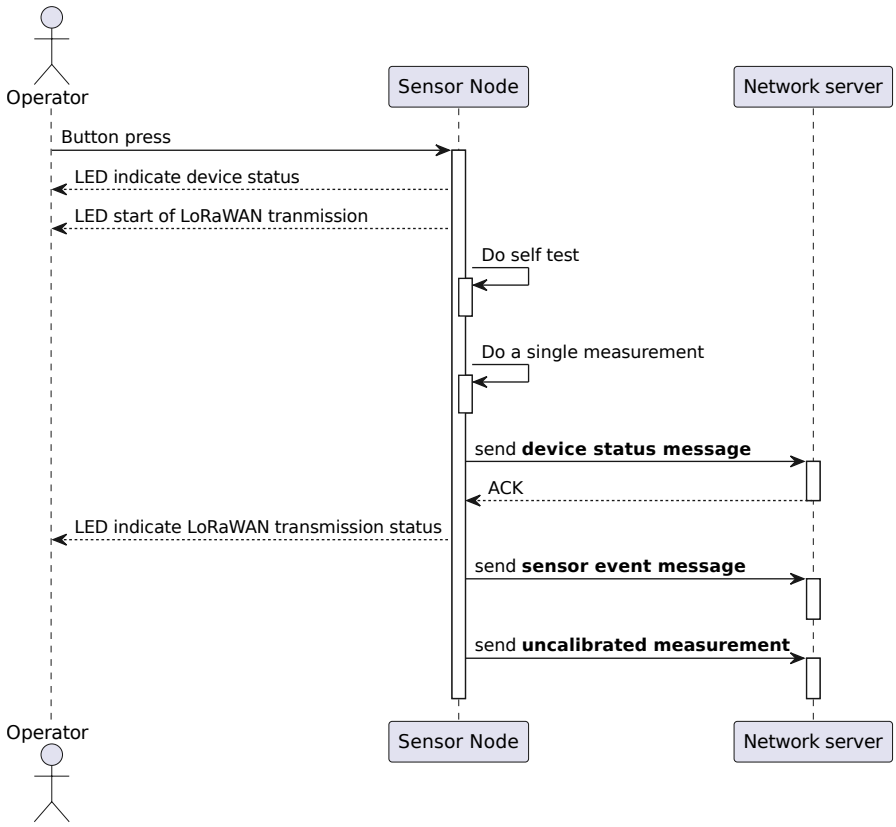

Figure 2: Button Press Scheme

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.

2.1 JSON Structure

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

Listing 1: Boot JSON Structure

2.2 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 2.3	1

Table 4: Boot message description

2.3 Reboot reason

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

Description	Binary encoding	Byte index
`reboot_reason`	uint8	0

Description	Binary encoding	Byte index

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

`reboot_reason.minor_reason` Minor reasons will be described in Table 6.		bits[4..7]

Table 5: 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 6: 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 measurement timer. The message contains information for maintenance, device health and other analysis. ***Device status message*** is also sent on short button press.

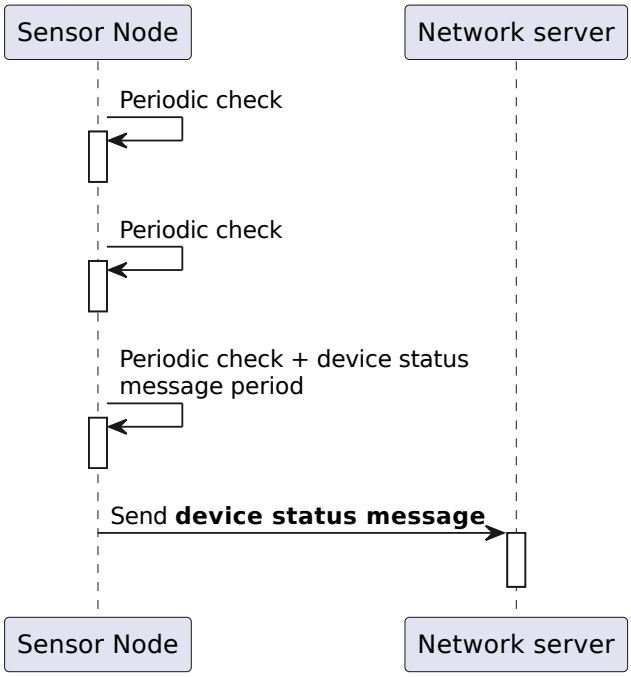

Figure 3: Device Status Scheme

3.1 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>
    }
}
```

Listing 2: Device Status JSON Structure

3.2 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 4.000 V.	1
`temperature` 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. 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. bit value: • 0: test failed • 1: test succeeded byte value: • 0x03FF (1023) : Self-tests of the device are OK, it is activated and the sensor is connected • 0x039F (927) : Failure in the measurement circuit • 0x03BF (959) : Sensor probe is not connected • 0x03EF (1007) : Device is deactivated • Other values : Self test failed	uint16 bit 0: battery measurement bit 1: flash memory bit 2: lora module bit 3: provisioning bit 4: activation bit 5: sensor measurement circuit bit 6: sensor probe connection bit 7: factory calibration bit 8: conversion factor bit 9: region	6,7
`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 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	8

Table 7: Device Status message description

4 Sensor Event

The sensor measurement is done 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* (Default mode) 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.

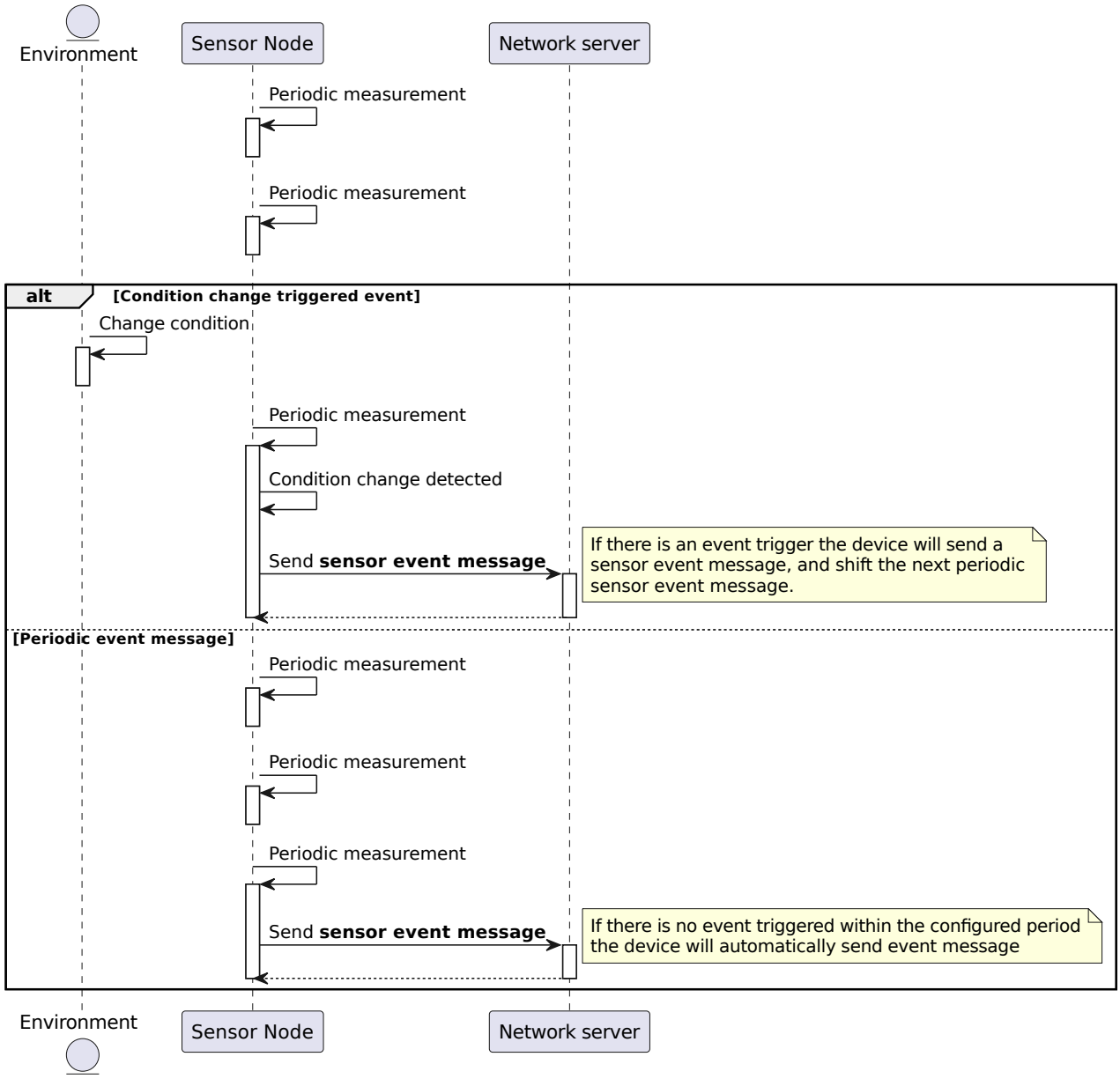

Figure 4: Event Message Scheme

4.1 JSON Structure

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

Listing 3: Normal Sensor Event JSON structure

4.2 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"
• "frequent"
	uint8
	
	bit0 = condition_0
	bit1 = condition_1
	bit2 = condition_2
	bit3 = condition_3
	
	bits[6..7]
	
	`condition_change = 0`
	`periodic = 1`
	`button_press = 2`
	`frequent = 3`
		2

Measurement values as described in table 9 and table 10 for *normal mode* and *extended mode*, respectively.		

Table 8: Sensor Event Message Description Common

Event Normal:

Description	Binary encoding	Byte index

`measurement.status`
If no error occurred during the measurements `measurement.status` will report `OK`. In case an error occurred `measurement.status` will report one of the errors listed in Table 11.
 	
`measurement.min` / `max` / `avg`
Either the one of the three: minimum, maximum or average of the measured values since last event message. Depending on the `selection`. The other two are not set.
 	
If `measurement.status` is not `OK`, then this field reports `null`.
	Errors are encoded in the measurement field as `uint32` and listed in Table 11, which correspond with `float32::NaN`.
	
	float32 (See Table 2)
	For unit, see Table 3.
		3..6

Table 9: Sensor Event Binary Encoding

Event Extended:

Description	Binary encoding	Byte index
`measurement.status` If no error occurred during the measurements `measurement.status` will report `OK`. In case an error occurred `measurement.status` will report one of the errors listed in Table 11.	Errors are encoded in the each measurement field as `uint32` and listed in Table 11, which correspond with `float32::NaN`. In case of an error each field will have the same error code.	
`measurement.min` The minimum of the measured values since last event message. If `measurement.status` is not `OK`, then this field reports `null`.	float32 (See Table 2) For unit, see Table 3.	3..6
`measurement.max` The maximum of the measured values since last event message. If `measurement.status` is not `OK`, then this field reports `null`.	float32 (See Table 2) For unit, see Table 3.	7..10
`measurement.avg` The average of the measured values since last event message. If `measurement.status` is not `OK`, then this field reports `null`.	float32 (See Table 2) For unit, see Table 3.	11..14

Table 10: Sensor Event Binary Encoding Extended

Error Codes

Error	Description	Binary encoding
`Hardware Error`	There is a problem with the hardware.	0xFF800001
`Upper Bound Error`	The sensor reading is higher than expected.	0xFF800002
`Lower Bound Error`	The sensor reading is lower than expected.	0xFF800003
`Factory Calibration Error`	Factory calibration is missing.	0xFF800004
`Conversion Factor Error`	Conversion factor is missing.	0xFF800005

Table 11: Error Codes Binary Encoding

5 Activation Process

Activation of the device 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.

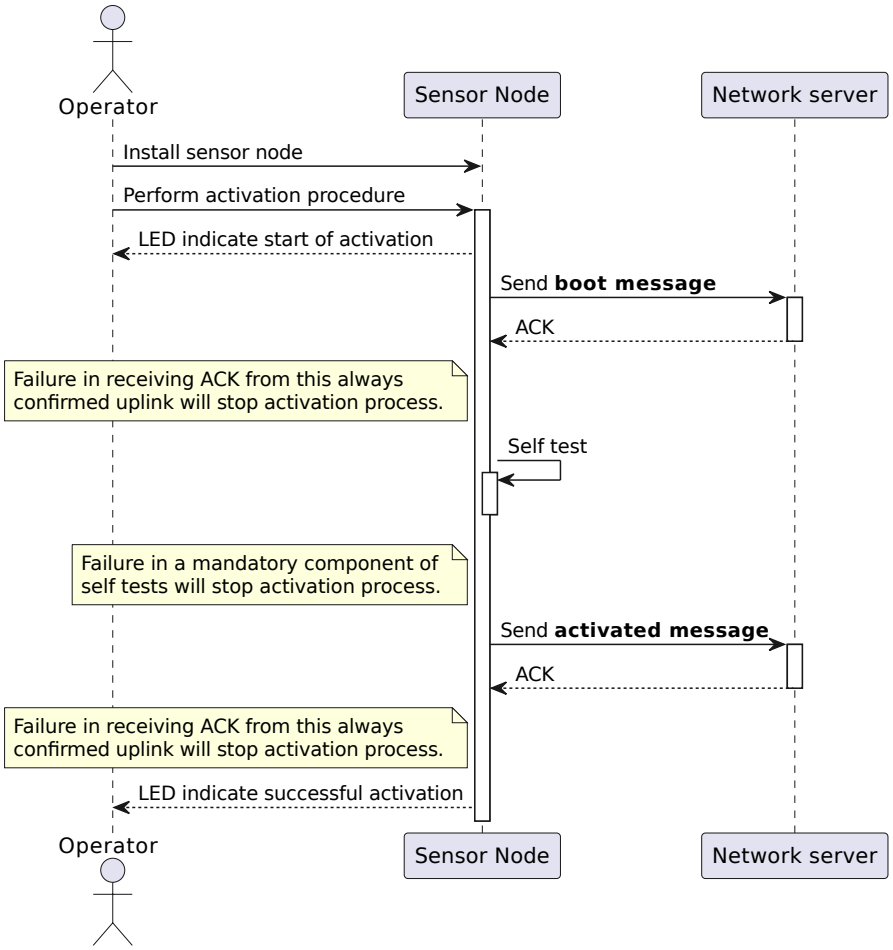

Figure 5: Activation Sequence Scheme

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

5.1 JSON Structure

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

Listing 4: Activation JSON Structure

5.2 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, the value is: • "rt"	uint8 `rt = 8`	1

Table 12: 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.

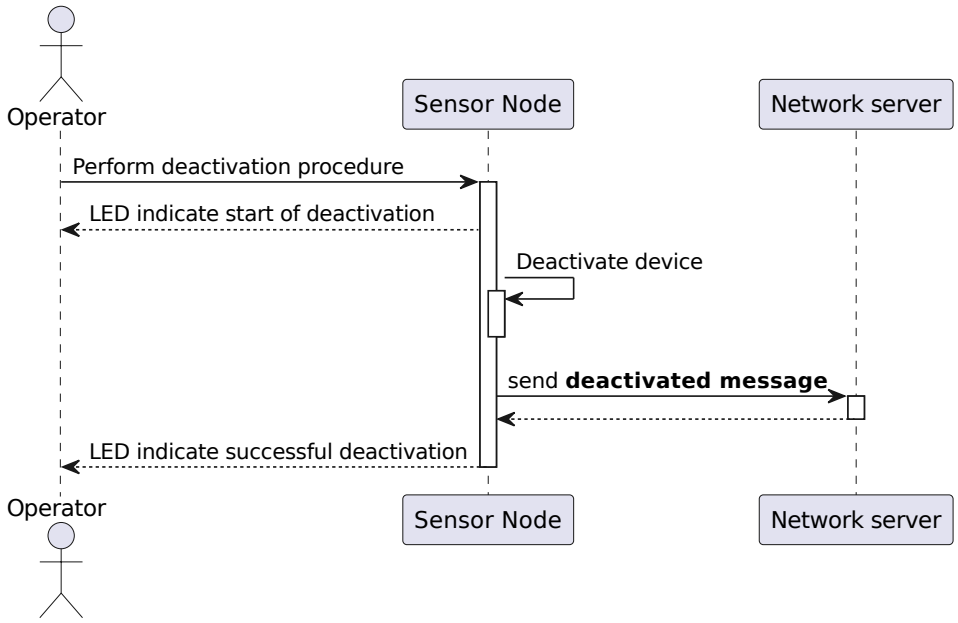

Figure 6: Deactivation Message Scheme

6.1 JSON Structure

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

Listing 5: Deactivation JSON Structure

6.2 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 13: 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. Configuration downlinks can be used to configure the non sensor related behavior (See Table 17), or sensor related behavior (see Table 18). Furthermore, they can be used to set the conditions which trigger the event message (See Table 19), or to calibrate the device (see Table 21). All configuration messages are sent using FPort 7.

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. 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 section.

After a valid ***Configuration update request*** with payload, the device reboots (see Figure 7). However, if multiple ***Configuration update request*** messages are queued before the next uplink message, the device only reboots once for all of them (see Figure 8). The later enables the user to batch update the configurations.

Typically, each configuration type has default settings provided by the manufacturer. Default settings are presented in *Default value* column of the associated table. They are given a tag by the manufacturer. Default configurations facilitate using the device as no configuration update is mandatory, enabling the user to utilize the device without configuration update. Changing default configurations can affect the battery life and/or quality of service.

Consider the following facts about a `tag`:

- A `tag` can be e.g., a hash or an ID.
- It is highly recommended to use distinct tags for each configuration. It means that two different configurations should not share the same tag. Please note that it is solely the user's responsibility to adhere to this recommendation, and the device does not provide any protection for non-compliance.
- The tags within the range of `0xFFFF0000` to `0xFFFFFFFF` are reserved by the manufacturer. Although the user has the option to select a tag within this range, it is strongly advised against doing so.

JSON Structures

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

Listing 6: Configuration Update Request JSON Structure

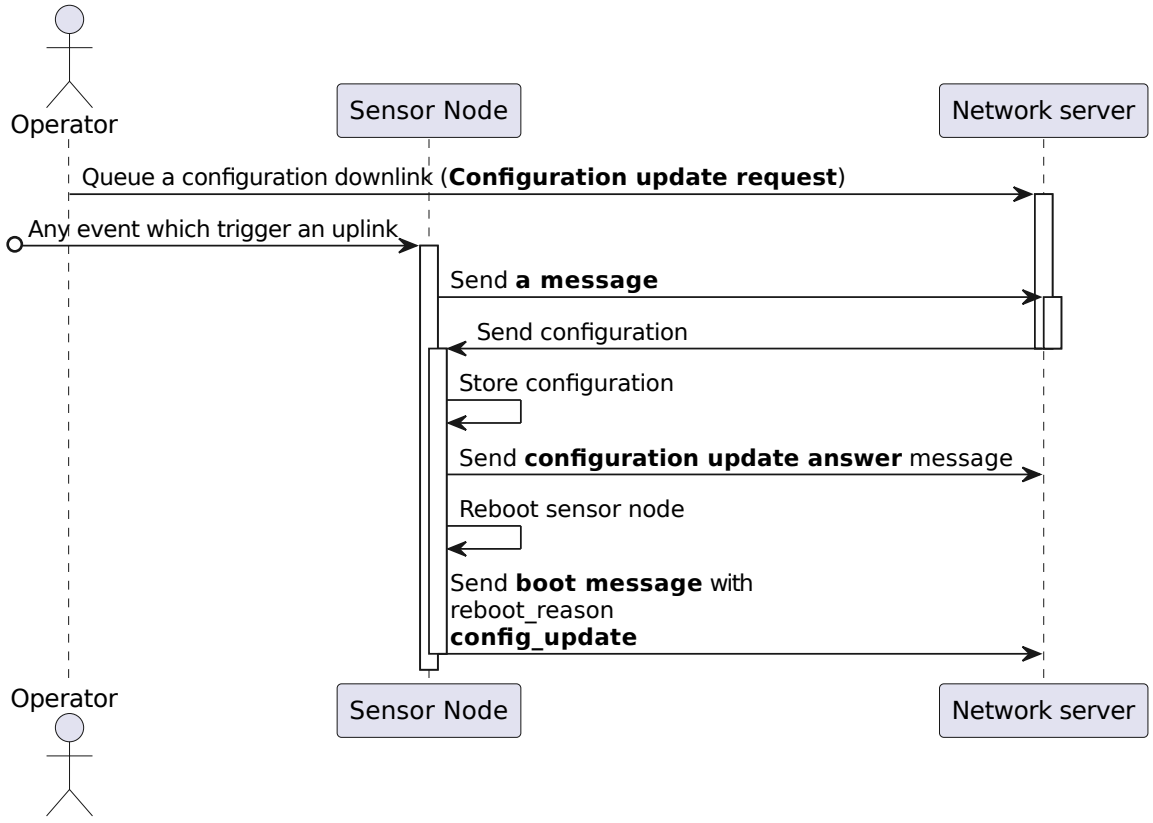

Figure 7: Single Configuration Update Scheme

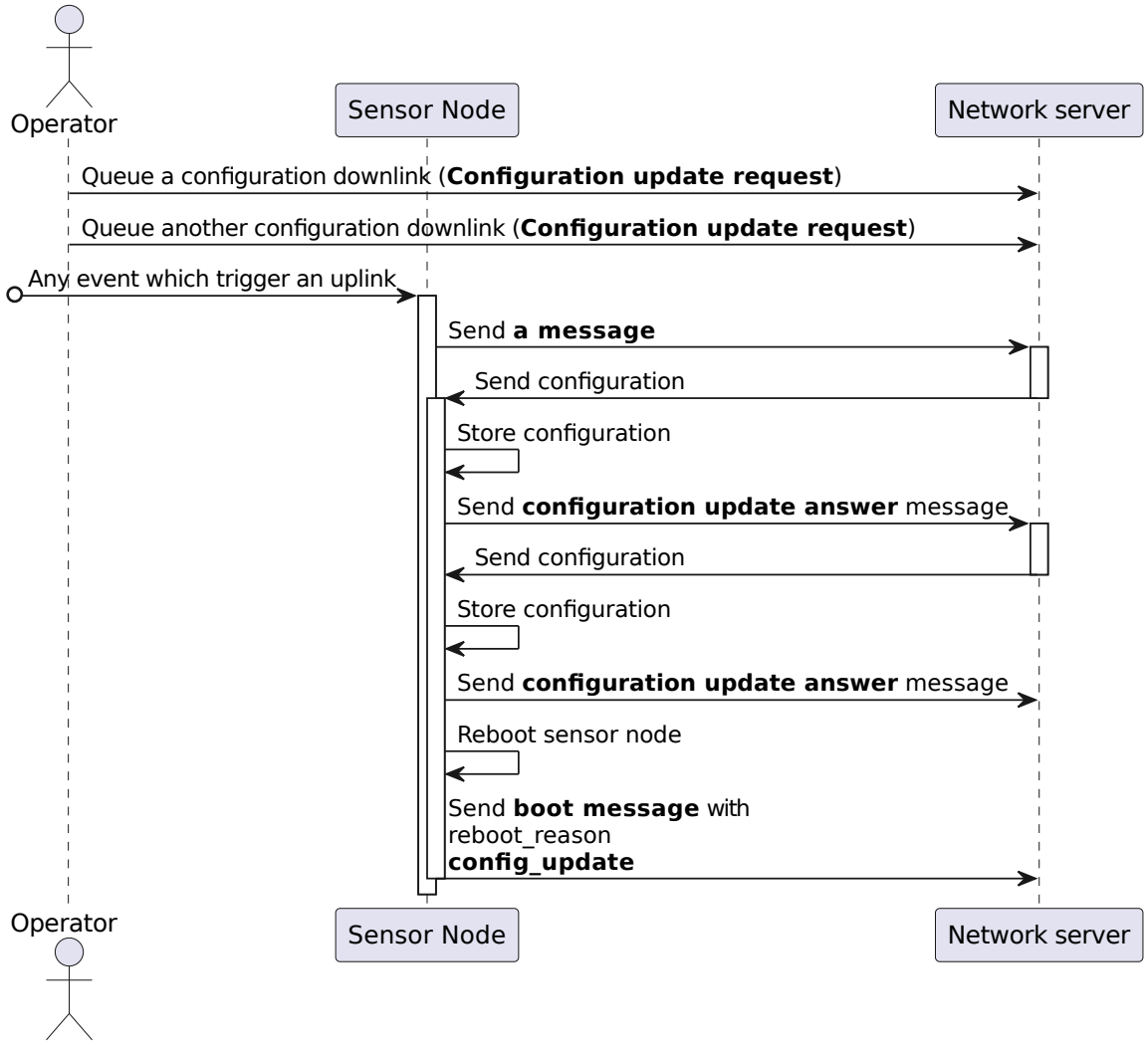

Figure 8: Multiple Configuration Update Scheme

Binary encoding and description

Description	Default value	Binary encoding	Byte index

`config_type`
The type of the configuration.
 	
• "base"
• "sensor"
• "sensor_data"
• "sensor_conditions"
• "user_calibration"
 	
Not all the configuration types are supported by the current device. Check the rest of the current section to see the supported configuration types.
	-	bits[0..3]
		
		`base = 0`
		`sensor = 3`
		`sensor_data = 4`
		`sensor_conditions = 5`
		`user_calibration = 6`
			0

`protocol_version` The version of the protocol.	-	bits[4..7]	
`tag` A 4 bytes value that can be assigned by the user to identify the configuration.		uint8[4]	1..4
`payload` 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 14: Configuration update request message description

7.1.2 Configuration Check Request

A ***Configuration update request*** is sent by the user to Get the `counter` and `tag` of the current configuration on the device (See Table 16) without changing the configuration. This helps to identify which configuration is already used by the device.

JSON Structures

```
{
    "config_update_req": {
        "config_type": <json string>,
        "protocol_version": <json number>
    }
}
```

Listing 7: Configuration Check 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"
• "user_calibration"
 	
Not all the configuration types are supported by the current device. Check the rest of the current section to see the supported configuration types.
	-	bits[0..3]
		
		`base = 0`
		`sensor = 3`
		`sensor_data = 4`
		`sensor_conditions = 5`
		`user_calibration = 6`
			0

`protocol_version` The version of the protocol.	-	bits[4..7]	

Table 15: Configuration check request message description

7.1.3 Configuration Update Answer

A **Configuration update answer** is always sent by the device in response to **Configuration update request** either when the user updates a configuration or checks a configuration.

- 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*** did not include the `payload`, ***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 string>,
        "counter": <json number>
    }
}
```

Listing 8: 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. 14.	-	bits[0..3]	0
`protocol_version` The version of the protocol.	-	bits[4..7]	
`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 16: 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 9: 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. 14 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 17: Base Configuration message description

7.2.2 Sensor Configuration

Payload JSON Structure

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

Listing 10: 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 14 with `configuration_type = "sensor"`.			0..4
`device_type` The device type as known in the NEON family. For this device the value is: "rt" (8)	"rt" Binary (hex): 0x08	uint8 "rt" = 8	5

`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]
			6

`measurement_interval_minutes` Interval in minutes, at which the 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	7

Description	Default value	Binary encoding	Byte index
`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	8..9

Table 18: Sensor Configuration Message Description

7.2.3 Sensor Conditions Configuration

As stated in Section 4, a sensor 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>,
                "measurement_threshold": <json number>,
                "frequent_event": <json boolean>
            },
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "measurement_threshold": <json number>,
                "frequent_event": <json boolean>
            },
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "measurement_threshold": <json number>,
                "frequent_event": <json boolean>
            },
            {
                "mode": <json string>,
                "measurement_window": <json number>,
                "measurement_threshold": <json number>,
                "frequent_event": <json boolean>
            }
        ]
    }
}
```

Listing 11: 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 14 with `configuration_type = "sensor_conditions"`.			0..4
`device_type` The device type as known in the NEON family. For this device the value is: • "rt"	"rt" Binary (hex): 0x08	uint8 `rt = 8`	5

Description	Default value	Binary encoding	Byte index
`event_conditions.frequent_event` It takes either `true` or `false` values: • true: For a given event_condition, if frequent_event is set to `true` and condition is true, one ***Sensor event message*** is triggered per measurement. • false: Device will trigger ***Sensor event messages*** as usual.	false Binary (hex): 0x00	uint8 bit 0: first event condition bit 1: second event condition bit 2: third event condition bit 3: fourth event condition bits 4..7: reserved	6
`event_conditions` Configuration of four independent event_conditions. See Table 20	[{"mode":"off"}, {"mode":"off"}, {"mode":"off"}, {"mode":"off"}] Binary (hex): 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	event[4]	7..26

Table 19: Sensor Conditions Configuration Binary Encoding

Binary Encoding and Description Event Condition 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 measurement along the **measurement_window** is above or equal to the **measurement_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 measurement along the **measurement_window** is below or equal to the **measurement_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 measurement is at least **measurement_threshold** higher than the minimum measurement 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 **measurement_threshold** is only positive.*
 	
decreasing mode: The condition is true when the current measurement is at least **measurement_threshold** lower than the maximum measurement 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 **measurement_threshold** is only positive.*
	0x00
	
	"off"
		uint8
		
		bits[0..1]
		
		`above = 0`
		`below = 1`
		`increasing = 2`
		`decreasing = 3`
			0

Description	Default value	Binary encoding	Byte index
`measurement_window` The maximum number of measurements to observe delta measurement to trigger an event. For mode "off" the default value is 0. For all other modes: Range: 1 measurement .. 63 measurements When the `frequent_event` is set to `True`, it is advisable to set the `measurement_window` parameter to 1 or a small value. This is because setting the `frequent_event` to `True` results in at least as many ***sensor event messages*** as the `measurement_window` once a condition is `True`.	0 Binary (hex): 0x00	bits[2..7]	
`measurement_threshold` The measurement threshold. In mode *increasing* and mode *decreasing*: Only *positive*	0 Binary (hex): 0x00 0x00 0x00 0x00	float32 For unit, see Table 3.	1..4

Table 20: Measurement Event Configuration Binary Encoding

7.2.4 User Calibration Configuration

The user can calibrate the measurements using a ***User calibration configuration*** message. A valid ***User calibration configuration*** message will overwrite an existing user calibration.

The measurements which are reported in ***sensor event message*** are calibrated using the parameters from the user calibration using the following linear equation.

$$M_{cal} = M + \overbrace{b \cdot M + a}^{calibration\ polynomial} \tag{1}$$

with,

- M_{cal} as the calibrated measurement value which is reported to the server with ***sensor event message***.
- M as the measurement before it is calibrated using user calibration.
- b as the slope coefficient of the linear equation.
- a as the constant term of the linear equation.

User calibration is optional. For default user calibration configuration, $a = 0$ and $b = 0$ which corresponds to $M_{cal} = M$.

To facilitate calibration polynomial calculation by user, the ***uncalibrated measurement message*** (See Sec. 8) includes not calibrated measurements.

JSON Structure

```
{
    "payload":
    {
        "device_type": <json string>,
        "coefficients": [
            {
                "a": <json number>,
                "b": <json number>
            }
        ]
    }
}
```

Listing 12: User Calibration Configuration Payload JSON structure

Binary Encoding and Description

Description	Default value	Binary encoding	Byte index
`config_type`, `protocol_version`, and `tag` as described in Table 14 with `configuration_type = "user_calibration"`.			0..4
`device_type` The device type as known in the NEON family. For this device the value is: • "rt"	"rt" Binary (hex): 0x08	uint8 `rt = 8`	5
`coefficients.a` Constant term of the calibration linear equation. Range: • $-3.40 \times 10^{38} < a \leq 3.40 \times 10^{38}$ • `NaN` is an invalid value.	0.0 Binary (hex): 0x00 0x00 0x00 0x00	float32	6 .. 9
`coefficients.b` Slope coefficient of the calibration linear equation. Range: • $-3.40 \times 10^{38} < b \leq 3.40 \times 10^{38}$ • `NaN` is an invalid value.	0.0 Binary (hex): 0x00 0x00 0x00 0x00	float32	10 .. 13
`reserved` These bytes must be zero.	0.0	float32[4]	14..29

Table 21: User Calibration Configuration Binary Encoding

8 Uncalibrated Measurement

As depicted in Figure 2, on button press, the device does a single sensor measurement and sends an ***Uncalibrated measurement message*** using FPort 8. The triggered ***Uncalibrated measurement message*** will include the given single measurement. The reported value corresponds to the measurement before user calibration. This information can be used to calculate the user calibration polynomial (See Section 7.2.4).

8.1 JSON Structure

```
{
    "uncalibrated_measurement": {
        "protocol_version": <json string>,
        "measurement": {
            "uncalibrated_measurement": <json number or null>,
            "status": <json string>
        }
    }
}
```

Listing 13: Uncalibrated Measurement JSON structure

8.2 Binary encoding and description

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

`measurement.status`
If no error occurred during the measurement `measurement.status` will report `OK`. In case an error occurred `measurement.status` will report one of the errors listed in Table 11.
 	
`measurement.uncalibrated_measurement`
The sensor measurement before being calibrated by user calibration.
 	
If `measurement.status` is not `OK`, then this field reports `null`.
	Errors are encoded in the `measurement.uncalibrated_measurement` field as `uint32` and listed in Table 11, which correspond with `float32::NaN`.
	float32 (See Table 2)
	For unit, see Table 3.
		1..4

Table 22: Uncalibrated Measurement Message Description

Revision History

Revision	Date	Author(s)	Description
A1	28-03-2023	NY	Created protocol V1.
A2	03-05-2023	NY	- Updated bist - Made minor fixes