Subject	LoRa WAN Communication Protocol VS-QT	Rev	2.2
		By	RK/ML/ZZ/MS/TB
Product	NEON Quarter Turn Valve Sensor	Date	07-10-2020

Version	Date	Author	Changes
2.0	9-4-2020	ZZ / ML	Split combined protocol in VS-QT, VS-MT and TS specific protocol documents. Separate downlink to protocol configuration and application configuration
2.1	5-6-2020	JP / ZZ / ML	1. Develop decoder / encoder according to: Split combined protocol in VS-QT, VS-MT and TS specific protocol documents. Separate downlink to protocol configuration and application configuration 2. Update message scheme diagrams to match new protocol configuration and application configuration 3. Rounded the range of status_message_confirmed_interval 4. Removed 500 kHz channel from lorawan_fsb_mask 5. Added disclaimer about lorawan_fsb_mask
2.2	07-10-2020	TB	1. Added 'Binary (hex)' for default values 2. Added CRC example

General

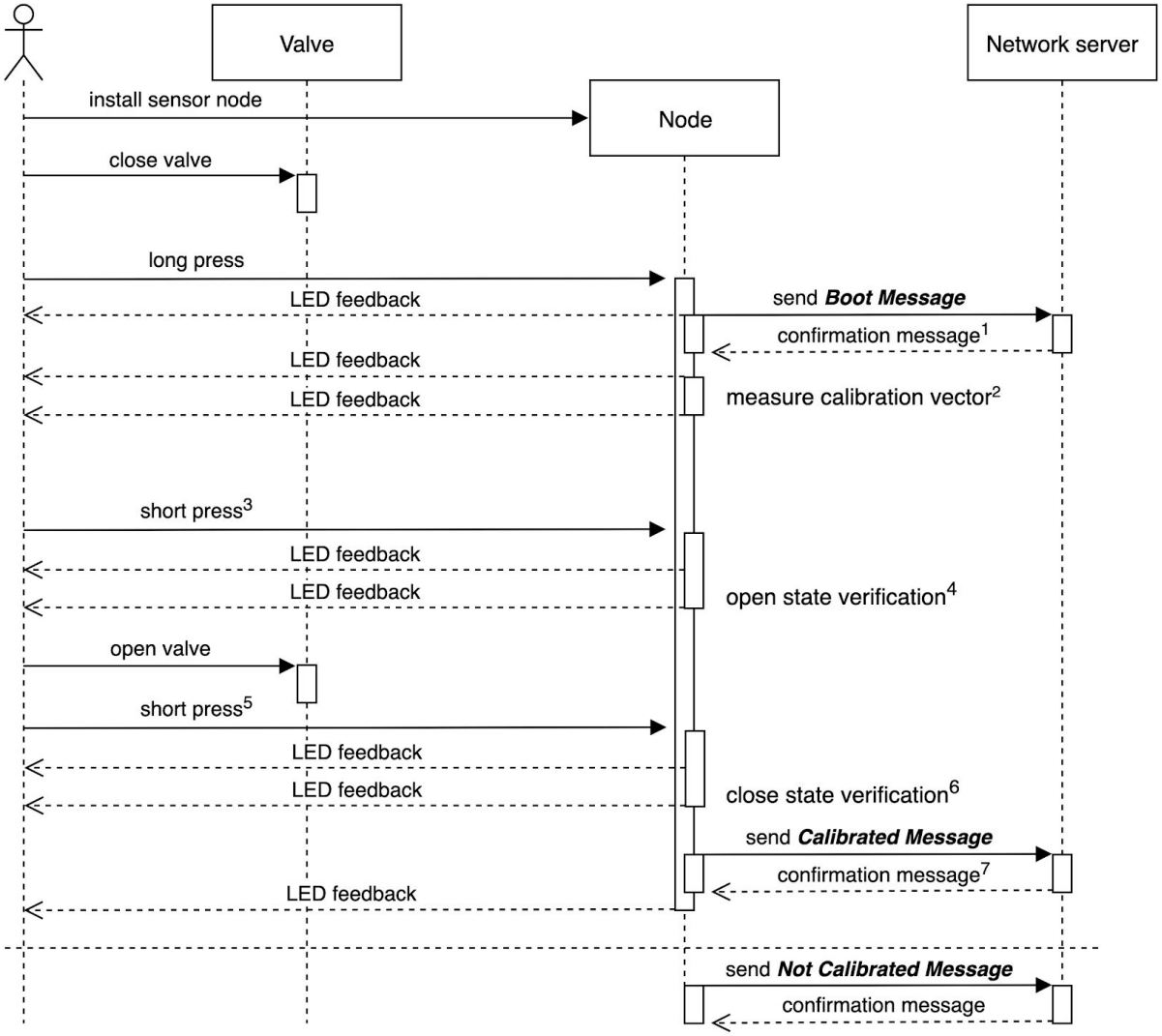
All variables are little endian.

Message schemes

Calibration

Calibration of the valve sensor is initiated by the operator. The valve sensor must be installed and the valve must be closed. First step of the calibration is the verification of LoRa coverage using a ***Boot Message***. The calibration is determined in closed position. To complete the calibration procedure the open and close state detection are verified. On

successful open and close detections the calibration is sent with a ***Calibration Message***. The full sequence is depicted below.

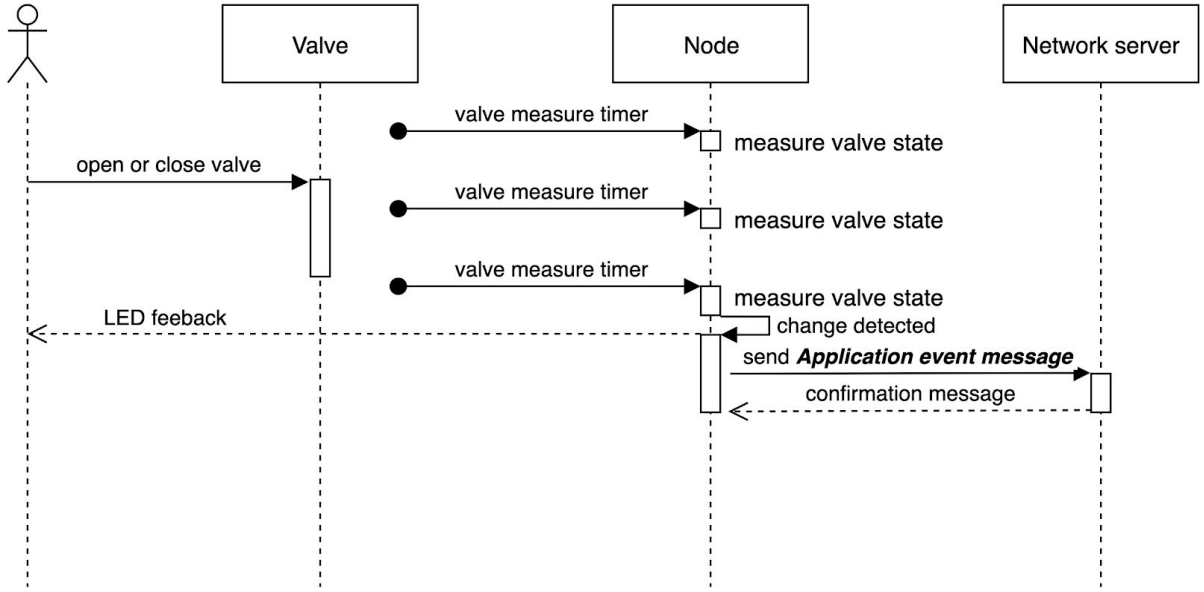


In case of an error during calibration the procedure is aborted. In case of an error that is not communication related it will notify the failure with a ***Not Calibrated Message*** with corresponding error code. The error situations can be as followed (see numbers in diagram):

1. No LoRa confirmation message from network server is received: no LoRa coverage
2. Timeout on determining calibration vector: too much magnetic noise
3. Timeout on button press: button press not detected within timeout
4. Failed to verify open state
    a. Timeout on measurement: too much magnetic noise
    b. Measured closed state: incorrected installation
5. Timeout on button press: button press not detected within timeout
6. Failed to verify closed state
    a. Timeout on measurement: too much magnetic noise
    b. Measured open state: incorrected installation

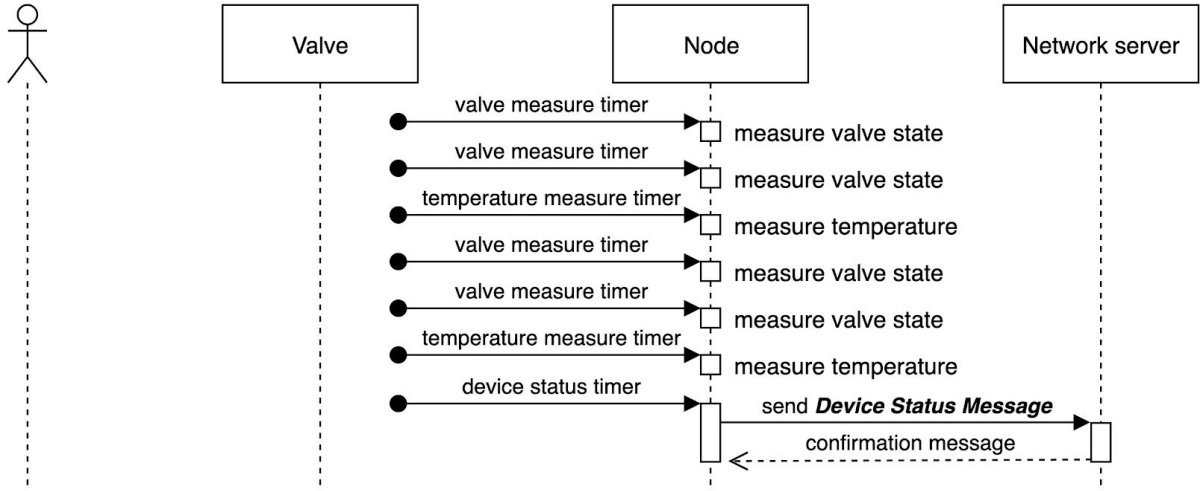
Application Event (valve state change)

The valve state is measured periodically. When the measurement is stable (stopped moving, no noise) the valve state is determined. On a state change an ***Application event message*** is sent.



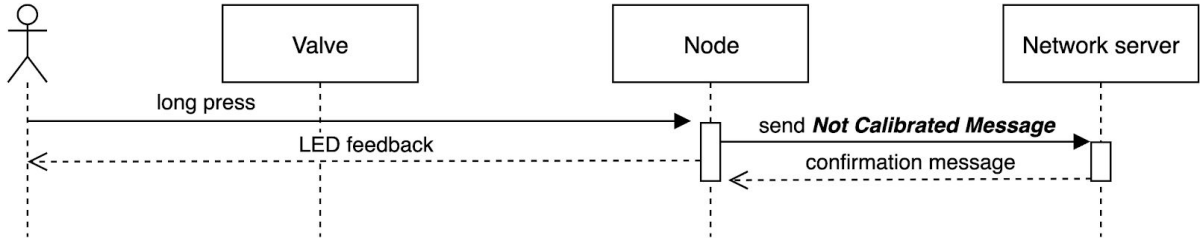
Device Status

Periodically a ***Device status message*** is sent. Timing is independent from the temperature measure timer and the valve measure timer (valve only). The message contains average, min and max values which are reset on send.



De-calibration

The operator can de-calibrate by a long press of the magnet key. This will result in a ***Not Calibrated Message***.

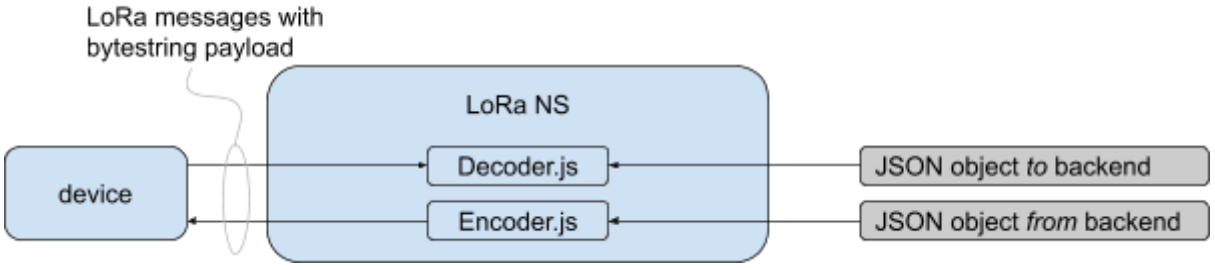


Message overview

ID	Name	Up/down	Purpose
0	Boot	up	Inform on device reboot
1	Calibrated	up	Indicates that a device is successfully calibrated and operational.
2	Not calibrated	up	Indicates that a device is not calibrated (anymore). Either due to failure during the calibration process or deactivated by the magnet key.
3	Application event	up	Inform on the valve state. It is sent on state transitions (open-close, close-open) or on a timer.
4	Device status	up	Informs on device health, battery info, counter for statistics etc. It is sent periodically.
5	Device configuration	down	Configure the device with radio setting
6	Application configuration	down	Configure the application specific settings

Decoding/Encoding

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.



Below each message is described and how it is encoded in the bytestring. A new decoder / encoder will follow according to this document. If using this Decoder / Encoder the bytestring information below can be ignored.

Message header

The header is at the head of all messages. It indicates the protocol version and message type.

JSON structure

```
"header": {
      "message_type": <json string>,
      "protocol_version": <json number>
}
```

Binary encoding and description

Description	**Binary encoding**	**Byte index**
`message_type` Type of message, range: • `boot` (0) • `calibrated` (1) • `not_calibrated` (2) • `application_event` (3) • `device_status` (4) • `device_configuration` (5) • `application_configuration` (6) **`protocol_version`** Version of the protocol, range: • For this protocol version the value is 2	uint8 bits [3..0] message_id bits [7..4] protocol version number	0

Uplink messages

Boot

The boot message is sent 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. continues communication failures, unforeseen situations etc). The boot message contains information 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.

JSON structure

```
"boot": {
      "device_type": <json string>,
      "version_hash": <json string>,
      "device_config_crc": <json string>,
      "application_config_crc": <json string>,
      "calibration_crc": <json string>,
      "reset_flags": <json number>,
      "reboot_counter": <json number>,
      "reboot_info": <json string>,
      "last_device_state": <json number>,
      "bist": <json number>
}
```

Binary encoding and description

Description	**Binary encoding**	**Byte index**
Header as described in Message header.	uint8	0
`device_type` The device type as known in the NEON family. For this device the value is: • "vs-qt" (2)	uint8	1
`version_hash` Version hash of the currently running firmware, represented as a hexadecimal string. Range: (hex string) `00000000 .. ffffffff`	uint32	2..5
`device_config_crc` CRC of the currently loaded protocol configuration, represented as a hexadecimal string. This can be used to verify if the corrected configuration is loaded. Range: (hex string) `0000 .. ffff`	uint16	6..7
`application_config_crc` CRC of the currently loaded application configuration, represented as a hexadecimal string. This can be used to verify if the corrected configuration is loaded. Range (hex string): `0000 .. ffff`	uint16	8..9

calibration_crc CRC of the currently loaded calibration data, represented as a hexadecimal string. This can be used to detect if a device is re-calibrated. Range (hex string): `0000 .. ffff`	uint16	10..11
reset_flags A bitmask field from the microcontroller that indicated the physical reason for the reset that caused the reboot. This can be used for analysis when a device is not working properly. bit 0 - Option Byte Loader bit 1 - Pin bit 2 - Power On bit 3 - Software. bit 4 - IWDG. bit 5 - WWDG bit 6 - Low Power	uint8	12
reboot_counter Counter of the number of reboots. Each time a reboot occurs the counter is increased. The counter is 8bit and will wrap after 255 to 0. This can be used for detecting abnormal rebooting behavior. Range: `0 .. 255`	uint8	13
reboot_info Informational string with information on the reboot. Example values: • `"swdog (ABCD)"` • `"assert (test:1234)"` • `"application (0xaabbccdd)"` • `"system (0xaabbccdd)"`	uint8[9] byte [0] reboot type byte [1..8] reboot type specific payload	14..22
last_device_state Last state the device was in before reboot. This can be used for analysis when a device is not working properly.	uint8	23
bist A bitmask with the result of the build in self test. At boot the device performs a self test in order to verify the working of essential components. This can be used for analysis when a device is not working properly.	uint8	24

bit value: ● 0: test failed ● 1: test succeeded bit 0: reserved (always 1) bit 1: reserved (always 0) bit 2: battery measurement bit 3: reserved (always 0) bit 4: sensor bit 5: lora module bit 6: provisioning bit 7: calibrated		

Calibrated

On a successful calibration a calibrated message is sent. It is an indication that the device is operational. The message contains additional information on the calibration internals, which can be ignored if not needed.

JSON structure

```
"calibrated": {
      "calibration_vector": {
            "x": <json number>,
            "y": <json number>,
            "z": <json number>
      },
      "open_verification_vector": {
            "x": <json number>,
            "y": <json number>,
            "z": <json number>
      },
      "closed_verification_vector": {
            "x": <json number>,
            "y": <json number>,
            "z": <json number>
      },
      "temperature": <json number>
}
```

Binary encoding and description

Description	Binary encoding	Byte index
Header as described in Message header.	uint8	0
`calibration_vector` Raw X, Y, Z sensor values of closed state during calibration. Range of each axis: `-32768 .. 32767`	int16[3]	1..6
`open_verification_vector` Result of verification measurement, performed right after calibration vector as part of calibration procedure. Vector of raw X,Y,Z values of open state. This can be used for analysis on the calibration procedure if valve state detection is not as expected. Range of each axis: `-32768 .. 32767`	int16[3]	7..12
`closed_verification_vector` Result of verification measurement, performed right after calibration vector as part of calibration procedure. Vector of raw X,Y,Z values of closed state. This can be used for analysis on the calibration procedure if valve state detection is not as expected. Range of each axis: `-32768 .. 32767`	int16[3]	13..18
`temperature` Temperature during calibration in units of 1 °C. This can be used for analysis on the calibration procedure if valve state detection is not as expected. Range of temperature: `-40 .. 80`	int8	19

Not calibrated

When the calibration is reset or the calibration procedure fails, a not calibrated message is sent. It is an indication that the device is not operational. The message contains the reason why it is not calibrated (anymore). It contains additional information of the calibration internals in case it fails during calibration, which can be used for analysis of failing calibrations and can be ignored if not needed.

JSON structure

```
"not_calibrated": {
      "reason": <json string>,
      "calibration_vector": {
            "x": <json number>,
            "y": <json number>,
            "z": <json number>},
      "open_verification_vector": {
            "x": <json number>,
            "y": <json number>,
            "z": <json number>},
      "closed_verification_vector": {
            "x": <json number>,
            "y": <json number>,
            "z": <json number>},
      "temperature": <json number>
}
```

Binary encoding and description

Description	**Binary encoding**	**Byte index**
Header as described in Message header.	uint8	0
`reason` Explanatory string on why it is not calibrated anymore. It is one of the following string: • `"Removed calibration"` (0) • `"Timeout; measure calibration"` (1) • `"Timeout; wait for open"` (2) • `"Timeout; verify open"` (3) • `"Expected open, measured closed"` (4) • `"Timeout; wait for closed"` (5) • `"Timeout; verify closed"` (6) • `"Expected closed, measured open"` (7)	uint8	1

In case the calibration is removed by holding the magnet key it will report "Removed calibration". All other values are indicating the cause of failing calibration procedure.		
`calibration_vector` Raw X, Y, Z sensor values of closed state during calibration. Only available for reason 2, 3, 5, 6, 7 Range of each axis: `-32768 .. 32767`	int16[3]	2..7
`open_verification_vector` Result of verification measurement, performed right after calibration vector as part of calibration procedure. Vector of raw X,Y,Z values of open state. This can be used for analysis on the calibration procedure if valve state detection is not as expected. Only available for reason 4, 5, 6, 7 Range of each axis: `-32768 .. 32767`	int16[3]	8..13
`closed_verification_vector` Result of verification measurement, performed right after calibration vector as part of calibration procedure. Vector of raw X,Y,Z values of closed state. This can be used for analysis on the calibration procedure if valve state detection is not as expected. Only available for reason 7 Range of each axis: `-32768 .. 32767`	int16[3]	14..19
`temperature` Temperature during calibration in units of 1 °C. This can be used for analysis on the calibration procedure if valve state detection is not as expected. Only available for reason 2, 3, 4, 5, 6, 7	int8	20

Range of temperature: `-40 .. 80`		

Application event

The application event message contains information on the valve state. It is either triggered by a state transition, change in open levels or periodic (scheduled).

JSON structure

```
"application_event": {
      "trigger": <json string>,
      "state": <json string>,
      "state_transition_sequence": <json number>,
      "angle": <json number>,
      "debug": {
            "temperature": <json number>,
            "vector": {
                  "x": <json number>,
                  "y": <json number>,
                  "z": <json number>
            }
      }
}
```

Binary encoding and description

Description	**Binary encoding**	**Byte index**
Header as described in Message header.	uint8	0
bit[0..1] state bit[2] trigger **`state`** String wit the possible values: • `"open" (0)` • `"closed" (1)` • `"error" (2)` **`trigger`** Source of trigger for the application event message. String wit the possible values: • `"state_transition" (0)` ○ On valve state transition from open to close, or close to open. • `"timer" (1)`	uint8	1

○ On the periodic timer		
state_transition_sequence Sequence number of valve state transitions. Every time a valve state transition is detected the number will be increased. This can be used to detect missed valve state transitions. Range: 0 .. 255 After 255 it will wrap to 1. Zero value is reserved for after boot only.	uint8	2
angle This signed integer provides angle vector measurement from a closed position in units of 0.1° from event detection. This can be used to indicate how far the valve is open. It depends on the type and size of the valve. It can be ignored if not needed. Range: -180.0° -- 180.0°	int16	3..4

Binary encoding and description debug

This information is added to the message when bit 1 of “Switch bitmask” in the device configuration is set.

Description	**Binary encoding**	**Byte index**
debug.temperature Temperature of PCB at moment of the event units of `1 °C`. Range: `-40 °C .. 80 °C`	int8 `1 °C` per LSB	9
debug.vector Vector of raw X,Y,Z values at the moment of event. Range of each axis: `-32768 .. 32767`	int16[3]	10..18

Device status

This message contains information for maintenance, device health and other analysis. It is sent periodically.

JSON structure

```
"device_status": {
      "device_config_crc": <json string>,
      "application_config_crc": <json string>,
      "calibration_crc": <json string>,
      "event_counter": <json number>,
      "unstable_counter": <json number>,
      "battery_voltage": {
            "low": <json number>,
            "high": <json number>,
            "settle": <json number>
      },
      "temperature": {
            "min": <json number>,
            "max": <json number>,
            "avg": <json number>
      },
      "tx_counter": <json number>,
      "avg_rssi": <json number>,
      "avg_snr": <json number>,
}
```

Binary encoding and description

Description	**Binary encoding**	**Byte index**
Header as described in Message header.	uint8	0
`device_config_crc` CRC of the currently loaded protocol configuration, represented as a hexadecimal string. This can be used to verify if the corrected configuration is loaded. Range: (hex string) `0000 .. ffff`	uint16	1..2
`application_config_crc` CRC of the currently loaded application configuration, represented as a hexadecimal string. This can be used to verify if the corrected configuration is loaded. Range (hex string): `0000 .. ffff`	uint16	3..4
`calibration_crc`	uint16	5..6

CRC of the currently loaded calibration data, represented as a hexadecimal string. This can be used to detect if a device is re-calibrated. Range (hex string): `0000 .. ffff`		
`event_counter` Counter for number of events. Every time an event message is sent this counter is increased. This can be used to detect missed event messages. Range: 0 .. 255 After 255 it will wrap to 1. Zero value is reserved for after boot only.	uint8	7
`unstable_counter` Counter of unstable measurements since the last device status message. This can be used to analyse issues with incorrect valve state detection. Range: 0 .. 255 It is reset after the device status message is sent. If the number of unstable measurements becomes bigger than 255 this field is set to 255.	uint8	8
`battery_voltage` Voltage measurement in Volt, meant as input for battery charge estimation in the backend. Note that a single voltage level on itself is not suitable for battery charge determination because of the type of battery. To feed the model on the backend with accurate data the voltage is measured on different moments. During **`low`** load, **`high`** load and after a **`settle`** time after high load. Range: `0.000 V .. 4.000 V`	uint16[3] `0.001 V` per LSB	9..14
`temperature` PCB temperature in units of `1 °C`. It is reported in the **`min`**, **`max`** and **`avg`** temperature since the last device status message.	int8[3] `1 °C` per LSB	15..17

Range: `-40 °C .. 80 °C`		
`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 bigger than 255 this field is set to 255.	uint8	18
`avg_rssi` The average RSSI of received messages since the last device status message. The value is in units of `1 dBm`. Range: `-255 dBm .. 0 dBm`	uint8 `-1dBm` per LSB	19
`avg_snr` The average SNR of received messages since the last device status message. Range: `0 dB .. 255 dB`	uint8	20

Downlink

Device configuration

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

JSON structure

```
"device_config" = {
	"switch_mask": {
		"enable_confirmed_changed_message": <json bool>,
		"enable_debug_data": <json bool>
	},
	"communication_max_retries": <json number>,
	"unconfirmed_repeat": <json number>,
	"periodic_message_random_delay_seconds": <json number>,
	"status_message_interval_seconds": <json number>,
	"status_message_confirmed_interval": <json number>,
	"lora_failure_holdoff_count": <json number>,
	"lora_system_recover_count": <json number>,
	"lorawan_fsb_mask": [
		<json number>,
		<json number>,
		<json number>,
		<json number>,
		<json number>
	]
}
```

Binary encoding and description

Description	Default value	Binary encoding	Byte index
Header as described in Message header.		uint8	0

`switch_mask`
Booleans to turn features on or off.
 	
`enable_confirmed_changed_message` `(0)`
Enable confirmed *application event* message
 	
`enable_debug_data` `(1)`
Enable extra debug data on the event message
	
	`true`
	
	`false`
	
	Binary (hex): 0x01
		uint8 bitmask
		
		bit 0
		
		bit 1
			1

		(other bits are unused)	
`communication_max_retries` The maximum number of retries on failing confirmed messages. Range: 1 - 254	3 Binary (hex): 0x03	uint8	2
`unconfirmed_repeat` The number of repeating on unconfirmed messages. Range: 0 - 5	2 Binary (hex): 0x02	uint8	3
`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 .. 255 seconds	60 Binary (hex): 0x3C	uint8	4
`status_message_interval_seconds` Interval in seconds at which periodic device status messages are sent. Range: `60 - 604800 seconds` (= 7 days)	86400 (once per day) Binary (hex): 0xA0 0x05	uint16 60 seconds per LSB	5..6
`status_message_confirmed_interval` Confirm every *n* messages, the messages in between are sent unconfirmed. Default is 1, such that all periodic messages are confirmed. The number can be increased to require less downlinks (for reasons of gateway RF duty cycle or network server costs), but will degrade the quality of service. Range: 0 - 100 DISCLAIMER: Although it is possible to make status messages always unconfirmed (by setting the value to 0) it is highly recommended	1 Binary (hex): 0x01	uint8	7

to not to use this to avoid problems when join sessions are invalidated by the network server (due to network server problems).			
lora_failure_holdoff_count In case of persistent network problems (not receiving acknowledgements on confirmed messages) the device tries to recover by a device reboot. This parameter configures the number of consecutive failed confirmed messages needed before it reboots. Range: 0 - 255	2 Binary (hex): 0x02	uint8	8
lora_system_recover_count The number of attempts the LoRa handler is trying to recover from a system failure (not responsive radio). Range: 0 - 255 DISCLAIMER: This value should not be changed for normal use.	1 Binary (hex): 0x01	uint8	9
lorawan_fsb_mask Frequency sub-band (FSB) mask for upstream [Only for US915] DISCLAIMER: The device is only tested and certified for the FCC 125KHz Hybrid transmission mode (1st 8 channels), therefore this channel configuration should not be changed for operational conditions	US915 Hybrid: {"0x00FF", "0x0000", "0x0000", "0x0000", "0x0000"} Binary (hex): 0xFF 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 EU868:	uint16[5]	10..19

	{“0x0000”, “0x0000”, “0x0000”, “0x0000”, “0x0000”} Binary (hex): 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00		
Device config CRC CRC of the configuration values above. It is calculated by the Encoder.	See Configuration CRC	uint16	20..21

Example

The device configuration message is constructed of the 1 byte header, the configuration payload and the 2 byte CRC. The default configuration is obtained by concatenating the header (0x25) with the default values of config payload and the calculated CRC.

For the EU868 variant this results in:

Header (hex)	Config payload (hex)	CRC (hex)
25	01,03,02,3c,a0,05,01,02,01,00,00,00,00,00,00,00,00,00	d4,55

For the US915 Hybrid variant this results in:

Header (hex)	Config payload (hex)	CRC (hex)
25	01,03,02,3c,a0,05,01,02,01,ff,00,00,00,00,00,00,00,00,00	c0,82

The calculation of the CRC is explained in chapter Configuration CRC.

Application configuration

JSON structure

```
"application_config" = {
      "device_type": <json string>,
      "magnet_measurement_interval_seconds": <json number>,
      "calibration_offset": <json number>,
      "angle_threshold": <json number>,
      "angle_hysteresis": <json number>,
      "angle_stability_threshold": <json number>,
      "angle_stability_window": <json number>,
      "periodic_event_message_interval_seconds": <json number>
}
```

Binary encoding and description

Description	Default value	Binary encoding	Byte index
Header as described in Message header.		uint8	0
device_type The device type as known in the NEON family. For this device, Valve Sensor-Quarter Turn, the value is: "vs-qt" (2)	"vs-qt" Binary (hex): 0x02	uint8 "vs-qt" = 2	1
magnet_measurement_interval_seconds Interval in seconds, at which the magnetometer is read. Changing this value has an effect on responsiveness and battery life. Range: 1 .. 255 seconds	2 Binary (hex): 0x02	uint8	2
calibration_offset	0	int8	3

Offsets that are added to the calibration vector. Under normal operation the value should be 0. Range: -12.0° .. 12.0°	Binary (hex): 0x0	0.1° per LSB	
`angle_threshold` Angle threshold for closed to open detection. The unit is in angle (0.1°). Range: 0.0° .. 25.5°	2.5 Binary (hex): 0x19	uint8 0.1° per LSB	4
`angle_hysteresis` Hysteresis window on the threshold for open to closed detection. The unit is in angle (0.1°). Range: 0.0° .. 25.5°	0.5 Binary (hex): 0x05	uint8 0.1° per LSB	5
`angle_stability_threshold` Threshold within the measurement is labeled as stable. The unit is in angle (0.1°). Range: 0.0° .. 25.5°	0.3 Binary (hex): 0x03	uint8 0.1° per LSB	6
`angle_stability_window` The number of angle measurements to label it as stable. Changing the parameter has an effect on responsiveness. Range: 1 .. 255	3 Binary (hex): 0x03	uint8	7
`periodic_event_message_interval_seconds` Interval in seconds at which the application event messages are sent additionally to the other triggers. Range: `60 - 604800 seconds` (= 7 days)	86400 (once per day) Binary (hex): 0xA0 0x05	uint16 60 seconds per LSB	8..9
Application config CRC CRC of the configuration values above. It is calculated by the Encoder.	See Configuration CRC	uint16	10..11

Example

The application configuration message is constructed of the 1 byte header, the configuration payload and the 2 byte CRC. The default configuration is obtained by concatenating the header (0x26) with the default values of config payload and the calculated CRC, which results in the following byte string:

Header (hex)	Config payload (hex)	CRC (hex)
26	02,02,00,19,05,03,03,a0,05	58,41

The calculation of the CRC is explained in chapter Configuration CRC.

Configuration CRC

The config messages contain a CRC to identify different configuration payloads. This CRC is reported back to the backend via the boot and status messages. The messages are constructed as followed:

Part	Size (bytes)
Header	1
Config payload	n
CRC	2

The payload length n is fixed per config type. The CRC is calculated over the config payload only, so without the message header.

CRC calculation

The two CRC bytes are the inverse of the lower two bytes of a CRC32 calculation. For the CRC32 calculation the CRC-32/ISO-HDLC is used. This is the default in many protocols and the implementation is widely available in many libraries. The two CRC bytes in the protocol can be extracted as followed (pseudo code):

```
crc = default_crc32([config_payload])
inverse_crc = crc ^ 0xFFFFFFFF
crc_byte0 = inverse_crc & 0xFF
crc_byte1 = (inverse_crc >> 8) & 0xFF
```

Python

CRC function in Python

```
import binascii
def calc_crc(bs):
    crc = binascii.crc32(bs)
    return (crc & 0xFFFF) ^ 0xFFFF
```

Example using the Python CRC function

```
bs = binascii.a2b_hex("0103023ca00501020100000000000000000000")
crc = calc_crc(bs)
print(hex(crc))
```

This results in: `0x55d4`, which are encoded as the two CRC bytes (hex): `d4,55`

Representing the CRC as little endian uint16 bytestring (pack the uint16 CRC and print as hex):

```
from struct import pack
print(binascii.b2a_hex(pack("<H", crc)))
```

Results in `b'd455'`

Javascript

CRC function in Javascript

```
// calc_crc inspired by https://github.com/SheetJS/js-crc32
function calc_crc(buf) {
  function signed_crc_table() {
    var c = 0, table = new Array(256);

    for (var n = 0; n != 256; ++n) {
      c = n;
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      c = ((c & 1) ? (-306674912 ^ (c >>> 1)) : (c >>> 1));
      table[n] = c;
    }

    return typeof Int32Array !== 'undefined' ? new Int32Array(table) :
            table;
  }
  var T = signed_crc_table();

  var C = -1, L = buf.length - 3;
  var i = 0;
  while (i < buf.length) C = (C >>> 8) ^ T[(C ^ buf[i++]) & 0xFF];
  return C & 0xFFFF;
}
```

Example using the Javascript CRC function

The `calc_crc` function can be used as followed (this can be tested using online services like https://jsfiddle.net/):

```
var configPayload = [0x01, 0x03, 0x02, 0x3c, 0xa0, 0x05, 0x01, 0x02, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00];
var crc = calc_crc(configPayload)
console.log("Calculated CRC: 0x" + crc.toString(16))
```

This results in `"Calculated CRC: 0x55d4"`, which are encoded as the two CRC bytes (hex): `d4,55`