# Wireless M-Bus Gateways

This page describes the Lobaro wireless M-Bus gateway firmware, called `app-nrf9160-wmbus`, which is executable on different hardware variants sold as different products.



> ⚠ Please find our new documentation an latest gateway here: wMbus Gateway V4

## Overview


The Lobaro wireless M-Bus gateways collect consumption values from up to 500 commercially available water meters, heat meters, heat cost allocators or similar with 868 MHz wireless M-Bus radio interface or Sensus RF Bubble Up and forward them encrypted via NB-IoT, LTE-M1 cellular radio or LoRaWAN networks for further processing on the Internet.

Forwarded meter values are transmitted, optionally additionally encrypted via DTLS, to a shared or private instance of the Lobaro IoT platform and can be viewed there or downloaded as a CSV file. Alternatively, standardised APIs such as MQTT, HTTP Push, SFTP or a REST interface are available to connect downstream systems or platforms easily and securely. When using LoRaWAN, the Lobaro Platform is optional. When using NB-IoT or LTE-M, on the other hand, it is mandatory. This requirement is explained in the Lobaro IoT Platform FAQ.

Thanks to the new NB-IoT mobile radio, optimised for sensor data, remote reading even works in places such as basements where smartphones have poor or no reception.

## Hardware Platforms and Variants

This firmware, following the naming sheme *app-nrf9160-wmbus-TZ2-VERSION-HARDWARE*, exists in versions targeted for different Lobaro hardware. On all hardware the workflows and functionality is the same. If a certain firmware feature is not available on a specific hardware variant, this will be indicated separately.

| **Available active Variants - Firmware compatible** |
| --- |
| |

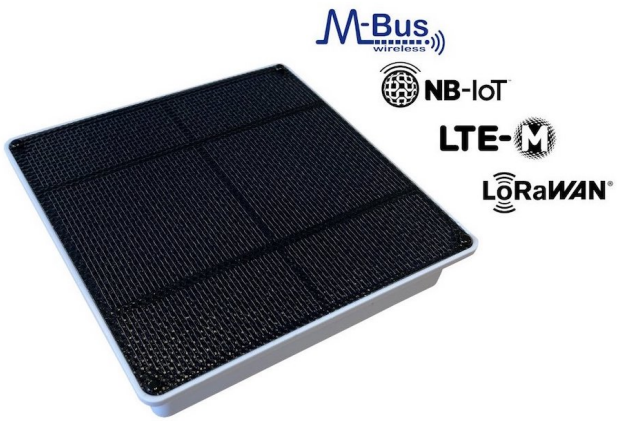
## LOB-GW-HYB-WMBUS


*"Wireless M-Bus Gateway V3"*

- Battery driven variant (3.6V D-Cell)
- Good for: Some meter readouts per month
- Example firmware name:  app-nrf9160-wmbus-TZ2-0.14.1**+hw3**
- Lobaro article number: #8000162 + #3000581 (Battery)

Variant specific information

## LOB-GW-SUN-WMBUS




*"Solar Wireless M-Bus Gateway"*

- Batteryless solar powered variant
- Good for: Frequent meter readings (actual intervals are location dependent)
- Example firmware name: app-nrf9160-wmbus-TZ2-0.14.1**+sun**
- Lobaro article number: #8000179

Variant specific information

## LOB-GW-DINRAIL-HYB-WMBUS

*"Wireless M-Bus Gateway (ext. Power, Din-Rail)"*

- External 12V-24V powered variant for din-rail mounting
- Good for: Very frequent and regular meter readouts
- Example firmware name:  app-nrf9160-wmbus-TZ2-0.14.1**+dinrail**
- Optionally bundled with 230V power-supply and extra shell housing
- Lobaro article number: #8000157 (standalone), #8000158 (bundle)

Variant specific information

## LOB-GW-DINRAIL-HYB-WMBUS-C

*"Wireless M-Bus Concentrator Gateway"*

- Extended Range (up to 1 km)
- Extended storage (over 2.500 telegrams)
- Example firmware name:  app-nrf9160-wmbus-TZ2-0.14.1**+dinrail**
- Lobaro article number: #8000182
- ⚠️ LoRaWAN not supported

Variant specific information

# Basic Workflow

The gateway remains in power-saving mode while not active most of the time. It leaves the low power sleep mode in the following situations:

- **Wake up at `listenCron` or after manual gateway reset / startup**
  1. Collect in all enabled modes sequentially
     a. Collect C1-Mode and T1-Mode telegrams (wMBUS) in parallel for `cmodeDurSec` (if not 0)
     b. Collect S1-Mode telegrams (wMBUS) for `smodeDurSec` (if not 0)
     c. Collect X-Mode telegrams (Sensus RF) for `xmodeDurSec` (if not 0)
     d. Collect U-Mode telegrams (Müller Funk) for `umodeDurSec` (if not 0)
  2. Upload all stored data via NB-IoT / LTE-CatM1 or LoRaWAN (see "WAN" configuration)
     a. Upload all collected metering telegrams & status message
     b. When the upload fails the upload is retried every 24h after daily status or until the next *listenCron* triggers.
  3. Sleep till next `listenCron` or status telegram upload.

- **Daily Status Wakeup**
  1. Upload status telegram normally at midnight 0:00h (UTC+0) | Sun Gateway: noon 12:00h (UTC+0)
  2. Upload remaining telegrams in memory, if any
  3. Sleep till next `listenCron` or status telegram upload.

**Key Facts**

- All telegrams are received "as is", e.g. encrypted or plain. Only always readable header information is parsed for filtering.
- Parsing and decryption happens in the backend, e.g. in the Lobaro IoT Platform. We also offer a standalone REST API parser for raw wireless M-BUS telegrams uploaded by the gateway.

## Compatible wireless meter protocols

- Wireless M-BUS S1, C1 or T1 modes, e.g. unidirectional 868 MHz modes following DIN EN 13757-4.
- Open metering specification (OMS, Annex O): PHY_A - 868 MHz (uplink only)
- Sensus RF Bubble UP - Manufacturer specific radio protocol (Xylem Inc.).
  - ⚠️ Decoding of Sensus RF telegrams needs the Lobaro Platforms telegram parser and unfortunately can't be open sourced.
- ME-Funk - Manufacturer specific radio protocol (Müller-electronic GmbH).
  - ⚠️ Decoding of ME-Funk telegrams needs the Lobaro Platforms telegram parser and unfortunately can't be open sourced.

💡 433 MHz variants available on special sales request:

- Open metering specification (OMS, Annex O): PHY_B - 433 MHz (uplink only)
- Sensus RF Bubble UP 433 MHz - Manufacturer specific (Xylem Inc.) radio protocol

## Anbindung an das Smart Meter Gateway (SMGW)

- SMGW Anbindung ()

# Configuration

Lobaro delivers all devices with a reasonable default configuration. Customer specific configurations are possible in different ways:

- Using our free Lobaro Maintenance Tool and the USB PC configuration adapter to be connected to the "config" connector on the hardware.
  - ⚠️ Not accessible in the waterproof version of the Solar Gateway.
- Remotely in the field using LoRaWAN downlink messages.
- Remotely using NB-IoT via the Lobaro Platform.

Configurations of up to 5.000 bytes are supported, which enables a Device-Whitelist of up to 500 Meter-IDs.

## Remote Configuration

- Devices using LTE-M or NB-IoT can be configured easily using the interface (search for 'config tab' inside devices) in the Lobaro Platform.
- Devices using LoRaWAN can configured by sending downlinks on port 128:
  - A single config parameter can be changed by sending `S<parameter>=<value>`
  - For example: changing the device to switch to using LTE-M or NB-IoT, you would change the parameter `"WAN"` to `"lte"` by sending: `"SWAN=lte"`
  - Depending on the used LoRaWAN network servers convention you will to encode this string in Base64 as `"U1dBTj1sdGU="` or Hex as `"5357414e3d6c7465"`.
  - Additionall Information: LoRaWAN Downlink Configuration

# Config Parameters

### WAN

Hybrid Connectivity

The device can upload / forward wireless M-BUS data using cellular IoT (NB-IoT or LTE-M) or LoRaWAN. The technology used is controlled by "WAN" configuration parameter.

This parameter can be used to set whether cellular IoT (NB-IoT, LTE-M) or LoraWAN is to be used for data transmission. With LoRaWAN, the type of network join can also be defined (ABP vs. OTAA).

| Name | Description | Default Value | Value Description & Examples |
|---|---|---|---|

| | | | |
|---|---|---|---|
| `WAN` | Technology used for connection and data uplinks to backend. This can be either cellular LTE (NB-IoT, LTE-M) or LoRaWAN | `lte` | • **lte**: use either cellular NB-IoT or LTE-M<br>• **lorawan**: use LoRaWAN with OTAA |

## LoRaWAN Parameters (`WAN` = "lorawan")

The connection to the LoRaWAN network is defined by multiple configuration parameters. This need to be set according to your LoRaWAN network and the way your device is supposed to be attached to it, or the device will not be able to send any data.

For a detailed introduction into how this values need to be configured, please refer to the chapter LoRaWAN configuration in our LoRaWAN background article.

| Name | Description | Type | Values | Default Value | Version |
|---|---|---|---|---|---|
| `OTAA` | Activation: OTAA or ABP | `bool` | `true`= use OTAA, `false`= use ABP | | |
| `DevEUI` | DevEUI used to identify the Device | `byte[8]` | e.g. `0123456789abcdef` | | |
| `JoinEUI` | Used for OTAA (called AppEUI in v1.0) | `byte[8]` | e.g. `0123456789abcdef` | | |
| `AppKey` | Key used for OTAA (v1.0 and v1.1) | `byte[16]` | | | |
| `NwkKey` | Key used for OTAA (v1.1 only) | `byte[16]` | | | |
| `SF` | Initial / maximum Spreading Factor | `int` | `7 - 12` | 12 | ⚠️ Removed since ??? |
| `ADR` | Use Adaptive Data Rate | `bool` | `true`= use ADR, `false`= don't | true | ⚠️ Removed since ??? |
| `OpMode` | Operation Mode | `string` | `A`= Class A, `C`= Class C | | ⚠️ Removed since ??? |
| `TimeSync` | Days after which to sync time | `int` | days, `0`=don't sync time | | |
| `RndDelay` | Random delay before sending | `int` | max seconds | | |
| `RemoteConf` | Support Remote Configuration | `bool` | `true`=allow, `false`=deactivate | true | ⚠️ Removed since ??? |
| `LostReboot` | Days without downlink before reboot | `int` | days, `0`=don't reboot | 5 | |
| `payloadFormat` | wMBUS Bridge LoRaWAN Payload Format | `int` | • 0 = Encoding in ports<br>• 1 = prefix bytes and time<br>• 2 = prefix bytes, time, and rssi | | |
| `loraMaxMsgSize` | Max. LoRa msg size before split (Payload Format `0` only) | `int` | `10-50` (bytes) | 50 | |

## NB-IoT Parameters (`WAN` = "lte")

The NB-IoT functionality is enabled if the `WAN` parameter is set to "lte". A SIM-Card has to inserted.

| Name | Description | Type | Values | Default Value | Version |
|---|---|---|---|---|---|
| `Host` | Hostname / IP of the Lobaro Platform API | `string` | • DTLS: coaps://platform.lobaro.com<br>• No DTLS: coap://platform.lobaro.com<br>• Plain IP: 94.130.20.37 (platform.lobaro.com)<br><br>From v0.14.x onwards different values can be separated by "," to define fallbacks, e.g. DTLS or DNS not working. | coaps://platform.lobaro.com | ⚠️ DTLS and DNS supported since v0.14.x |
| `Port` | Port number of the Lobaro Platform API | `int` | | `5683` | ⚠️ Removed since ???<br><br>Only Host is used now |
| `UdpHost` | Separate IP to upload plain telegrams via UDP | `string` | optional, empty = upload to Lobaro IoT Platform using Host parameter address | | |
| `UdpPort` | Separate Port to upload plain telegrams via UDP | `int` | only used when UdpHost is set | `0` | |

| | | | | | |
|---|---|---|---|---|---|
| `Operator` | Mobile Operator Code | `string` | 26201 (=Deutsche Telekom), for other operators, see above. | `Empty` | |
| `Band` | NB-IoT Band | `string` | "8", "20", "8,20", Empty = Auto detect (longer connecting time) | `8` | |
| `APN` | Mobile operator APN | `string` | 1nce: iot.1nce.net<br>Vodafone Easy Connect: lpwa.vodafone.com (l = littel L) | `iot.1nce.net` | |
| `PIN` | SIM PIN (since v0.7.0) | `string` | Empty or 4 digits (e.g. 1234) | | |
| `UseNbiot` | Try to connect with NB-IoT | `bool` | | `true` | |
| `UseLtem` | Try to fallback to LTE-M when supported by the Modem | `bool` | ⚠️ not supported with all Hardware revisions | `false` | |

## wMBUS / Metering Parameters

| Name | Description | Type | Values | Version |
|---|---|---|---|---|
| `listenCron` | Cron expression defining when and how often to collect wMBUS telegrams | `string` | `0 0/15 * * * *` (every 15 minutes) | extended in v0.15.4 |
| `cmodeDurSec` | Duration (Seconds) of C1/T1-mode receive | `int` | • `0` = Do not collect C1/T1 mode<br>• Max value = 36000 | |
| `smodeDurSec` | Duration (Seconds) of S1-mode receive | `int` | • `0` = Do not collect S1 mode<br>• Max value = 36000 | |
| `xmodeDurSec` | Duration (Seconds) of Sensus-RF-BUP-mode receive (Xylem) | `int` | • `0` = Do not collect Sensus RF BUP<br>• Max value = 36000 | since v0.7.2 (2021-01-27) |
| `umodeDurSec` | Duration (Seconds) of Müller Funk mode receive | `int` | • `0` = Do not collect Müller Funk<br>• Max value = 36000 | since v0.13.1 (2022-04-21) |
| `mFilter` | wMBus manufacturer filter sep. by `,` e.g. `dme,itw (Comma separated list WITHOUT spaces)` | `string` | `blank`= no filter | |
| `typFilter` | wMBus device type filter e.g. `08,07` for Heat Cost and Water | `string` | `blank`= no filter | |

| | | | | |
|---|---|---|---|---|
| `devFilter` | meter id filter e.g.<br><br>• cmode, smode (wmbus): `06198833` (exactly 8 digits with leading 0)<br>• xmode (Sensus RF): `10121335300` (11 digits, no "-"!)<br><br>**`(Comma separated list WITHOUT spaces: 88009035,13456035,56268931)`**<br><br>Up to 500 wMBus-IDs or 400 Sensus-RF-IDs are supported. | `string` | `blank`= no filter | |
| `ciFilter` | Collect only telegrams with specific values in the ci-Field, must we written as 2 hex digits (with leading zeros).<br><br>**`(Comma separated list WITHOUT spaces, e.g.: "8a,07,71")`** | `string` | `blank`= no filter | |
| `maxTelegrams` | Set hard limit on how many telegrams will be collected and uploaded. The bridge will stop collection, once this number has been collected, regardless of the passed time. Can be used save battery / data volume, should the device be in an area with a large number of meters.<br><br>Set to `0` for no limit. | `int` | | v0.8.5 |

## Battery runtime / Energy consumption

For all variants, the energy consumption strongly depends on the configuration of the '`listenCron`' and the '`modeDurSec`' parameters. We have a conservative Excel calculator that determines the expected battery runtime for the battery variant (Gateway V3). In many actual installations, longer run times can be achieved because the calculator is kept **very conservative**. As a general guideline which product variant to use can serve:

- Every 15 minutes data: Externally powered variants
- Hourly data: Externally powered variants, Solar Gateway with outdoor mounting
- Daily data with few meters (< 30): Externally powered variants, Solar gateway with indoor or outdoor mounting, Gateway V3 with battery
- Monthly data with many meters (250...500): Externally powered variants, Solar gateway with indoor or outdoor mounting, Gateway V3 with battery

# Filtering

The Wireless M-Bus Gateway has an (optional) filter mechanism, that can limit, which telegrams are processed by it. If filtering is used, any telegram is checked against the filters immediately after it has been received. Only if the telegram fits the criteria defined by all filters is it saved to the internal store and will be uploaded. All telegrams that don't fit will be dropped and not processed further. This can be important to save bandwidth and battery life of a device. In many areas there will be many wMBus devices that send telegrams you are not interested in.

There are 4 filters that check different aspects of a telegram:

- `mFilter` – Manufacturer filter – filters by the 3-letter manufacturer code that is present in every telegram (e.g. `LOB` for Lobaro GmbH).
- `typFilter` – Device Type filter – filters by the 2-hex-digit code defining the nature of the sending device (e.g. `07` for water meters).
- `devFilter` – Device filter – filters by the 8-digit ID, that is mandatory for each sending device (e.g. `87654321`).
- `ciFilter` – CI-Field filter – filters by the 2-hex-digit CI-Field present in every telegram. That is a technical code describing the purpose of a telegram (e.g. `8a`).

Each filter only checks for a single field of information in a telegram. If a filter is left blank, no filtering is done for that field. So if you for example leave `mFilter` blank, there will be no filtering over the manufacturer of the device. Each filter is processed independently.

A filter is a simple whitelist for its field. If a telegram's value is listed in the filter, the telegram will be collected. If not, it will be dropped. Entries in the list are separated by a single comma "`,`" (no spaces!). Starting with firmware version 0.17.0, the Gateway also supports blacklist filtering. If you add a single exclamation mark "`!`" in front of the list, the complete list will be treated as a blacklist.

**Example for an `mFilter` whitelist:**
"`SEN,ITW,DME`" will collect only telegrams from meters by Sensus Metering, Itron (Water), and Diehl Metering. All other telegrams will be dropped.

**Example for an `mFilter` blacklist:**
"`!SEN,ITW,DME`" will drop any telegram from meters by Sensus Metering, Itron (Water), or Diehl Metering. Any other manufacturer will be stored and uploaded. *This will not work with firmware versions below 0.17.0!*

## Telegrams with multiple layers

wMBus telegrams can have multiple headers in different layers. This can be the case if multiple devices are involved in creating the telegram, because it is read from the actual meter by an attached device that sends it out. Telegrams with multiple headers can have multiple different manufacturer codes, device types, and device IDs. These telegrams are accepted by a whitelist filter, if at least one of the values in the telegram is present in the whitelist. It will be dropped by a blacklist filter if at least one of the values from the telegram is present in the blacklist.

## Manufacturer filter

Each wMBus telegram has the manufacturer of the sending meter encoded as a 3-letter code assigned by the DLMS User Association. On their site you can find the complete list of manufacturer IDs. The field in the telegram that holds this information is called M-Field.

Entries in this filter must be exactly 3 letters long. The case is ignored.

Some examples of manufacturer codes:

| Code | Company |
|---|---|
| LOB | Lobaro GmbH, Hamburg, Germany |
| DME | Diehl Metering, Ansbach, Germany |
| QDS | Qundis GmbH, Erfurt, Germany |
| ARD | Arad Group Ltd, Dalia, Israel |
| SEN | Sensus Metering Systems, Ludwigshafen, Germany |
| SON | Sontex SA, Sonceboz, Switzerland |
| ITW | ITRON (Water), Issy-les-Moulineaux, France |

## Device Type Filter

Each wMBus device is of a type (e.g. water meter, heat cost meter). The type is encoded as a 2-hex-digit number. The entries must be exactly 2 hex digits long, case is ignored.

```
00: "Other",
01: "Oil",
02: "Electricity",
03: "Gas",
04: "Heat",
05: "Steam",
06: "Warm Water", // 30 - 90°C
07: "Water",
08: "Heat Cost",
09: "Compressed Air",
0A: "Cooling load meter (outlet)",
0B: "Cooling load meter (inlet)",
0C: "Heat (inlet)",
0D: "Heat / Cooling load meter",
0E: "Bus / System component",
0F: "Unknown",
10: "consumption meter",
11: "consumption meter",
12: "consumption meter",
13: "consumption meter",
14: "Calorific value",
15: "Hot Water", // >= 90°C
16: "Cold Water",
17: "Dual Water meter", // Hot and Cold
18: "Pressure",
19: "A/D Converter",
1A: "Smoke detector",
// 1B - DD: "Reserved"
1B: "Room", // e.g. temp, humidity
1C: "Gas detector",
1D: "Sensor",
1E: "Sensor",
1F: "Sensor",
20: "Breaker (electricity)",
21: "Valve (gas or water)",
22: "Switching device",
23: "Switching device",
24: "Switching device",
25: "Customer unit (display device)",
26: "Customer units",
27: "Customer units",
29: "Garbage",
2A: "Carbon dioxide",
30: "system device",
31: "Communication controller",
32: "Unidirectional repeater",
33: "Bidirectional repeater",
34: "system device",
35: "system device",
36: "Radio converter (system side)",
37: "Radio converter (meter side)",
38 - 0x3F // Reserved for system devices
40 - 0xFE: Reserved
FF: "Invalid", // Wild card searching [EN 13757-3:2013], 11.3 and 11.5.3
```

## Device Filter

Each wMBus device has an 8-digit device ID or Address. This ID is normally printed on the device. The combination of Manufacturer Code and Device ID should be globally unique. This is the most specific filter and can be used to tune the Gateway to only collect telegram from individual devices. Up to 500 devices can be listed in the filter.

Leading zeros can be omitted and are 1-8 digit long numbers. This filter also accepts 11-digit IDs for filtering Sensus-RF telegrams in X-mode.

The number of telegrams uploaded can be higher than the number of IDs in the whitelist, because some meters send out multiple different telegrams.

## CI-Field Filter

The CI-Field is a 2-hex-digit number that is used to encode type and purpose of a telegram. Some meters send multiple different types of telegrams that can be identified by this field. Entries in this list must be exactly 2 hex digits long, case is ignored.

## Filtering Strategies

There is no all-purpose-strategy for filtering. What is best for you will depend on your use cases.

Setting an explicit device list for each of your Gateways will lead to the most efficient use of battery and bandwidth. But it comes with huge administrative overload, as each Gateway will need an individually composed list. When new meters are installed, the list must also be updated.

For bigger role-outs it might be easier to work with device type filters. If you are only interested in water meters, a `typFilter` set to `"06,07,15,16,17"` might be a sufficient setting for your bridges; it will filter out any head cost meters and smoke detectors. Unintentionally received water meters can simply be ignored in the backend. A combination of `typFilter` and `mFilter` will often reduce the number of unwanted telegrams sufficiently.

Individual tuning can be done by initially installing a Gateway with no filters configured. After the first uploads, the wanted telegrams can be identified in the backend, and filters can then be created and sent to the Gateway way remote configuration downlink.

Starting with firmware version 0.17.0 it is also possible to put meters on a blacklist. If the Gateway uploads telegrams that you do not want, you can put each unwanted meter on the device blacklist via remote configuration.

## Filter fine tuning using `maxTelegrams`

The reception time the Gateway for collecting wMBus telegrams can be minimised by a combination of filters and `maxTelegrams`. If `maxTelegrams` is set to a number different than 0, the Gateway will stop collection and start uploading as soon as that many telegrams are in the store. Set the the IDs of all wanted Meters in `devFilter` and set `maxTelegrams` to the number of telegrams you expect. Be aware, that some devices will send out multiple different telegrams. Set `maxTelegrams` accordingly.

## Filtering Müller Funk (U-mode)

U-Mode is a special listening mode of the Gateway, that collects proprietary telegrams from meters by Müller-electronic GmbH. Only the `devFilter` will be used on U-mode telegrams, the other filters will be ignored. Device IDs for Müller Funk have the same format as in wMBus: 8 digits. When U-mode is used on a device that also uses C/T-mode or S-mode, put the IDs for both modes in the `devFilter` list. The IDs in the list will be used for both modes.

## Filtering Sensus RF (X-mode)

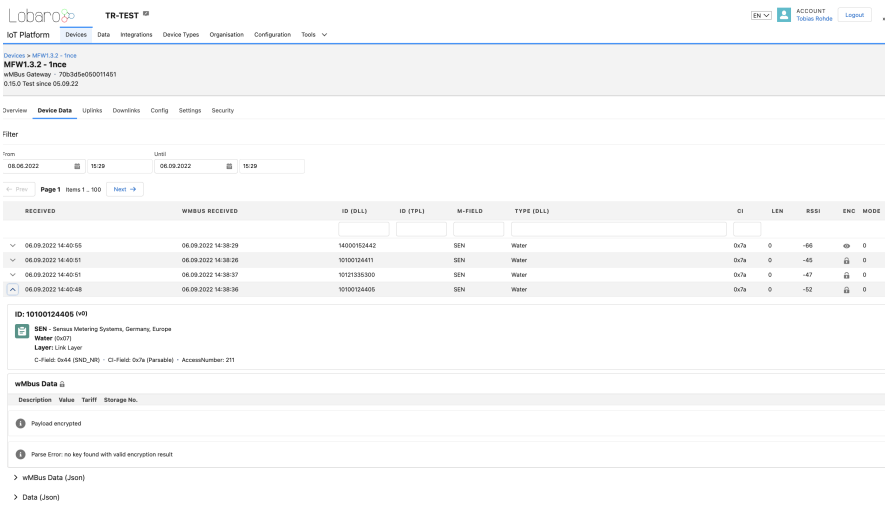
X-mode is a special listening mode of the Gateway, that collects proprietary telegrams from meters by Sensus/Xylim. Only the `devFilter` will be used on X-mode telegrams. The IDs used for Sensus-RF meters are 11 digit long and written on the meter in a format with dashes, like this: `1010-012-4411`. To use the device filter for X-mode, add the meter IDs to the list in `devFilter` without the dashes. Sensus-RF IDs must be entered using exactly 11 digits, e.g. `10100124411`.

If the a single device uses X-mode together with any other mode (C/T, S, U), the `devFilter` is used for both. Any ID that has 11 digits will be used for X-mode only. IDs with 1-8 digits will be used for all the other modes, but not for X-mode. If there are only 11-digit IDs in `devFilter`, it will be blank (= no device filter) for modes C/T, S, and U. If there are only 1-to-8-digit IDs in `devFilter`, it will be blank (= no device filter) for X-mode.

# Lobaro Platform

Wireless M-BUS data can be viewed and further processed, e.g. using MQTT, HTTP-Push or the REST interface, in the Lobaro IoT platform. It can also be connected with various available LoRaWAN network servers or used directly using cellular IoT, e.g. NB-IoT or LTE-M.

When using NB-IoT or LTE-M for data upload it is currently **mandatory**. This requirement is explained in the Lobaro IoT Platform FAQ.




The SaaS instance is available at platform.lobaro.com. It's free for testing purposes.

# LoRaWAN

The Gateway can use LoRaWAN als Uplink technology for wMbus Telegrams. It is not a LoRaWAN Gateway, thus other LoRaWAN Devices **can not** be received.

## Limitations in LoRaWAN vs. NB-IoT / LTE-M

- LoRaWAN Uplinks and Downlinks are limited to 52-222 Bytes depending on the Spreading Factor (Connection Quality).
  - Uplink with wMbus Telegrams might be split over multiple LoRaWAN Messages
  - Downlinks with big configuration values (e.g. long whitelist) must be split over multiple Downlinks which might be difficult to implement.
- Limited mount of Metatada:
  - Smaller Gateway Status Telegram (see below).
  - Raw Telegram with optional RSSI and Timestamp of the Telegram depending on the Payload Format.

# Uplink Payload formats

After collecting wireless M-Bus telegrams over the air, the Bridge starts uploading data via LoRaWAN. There exist two data formats that are transmitted over different LoRaWAN ports. As LoRaWAN can only transmit very short messages, the message formats contain only data bytes. The meaning of a byte is determined by its position within a message. The following describes the package formats used by the wireless M-Bus Bridge.

M-Bus telegrams can be longer as the maximal size of a LoRaWAN-Message. For this cases, the Bridge needs to split a telegram into multiple pieces and upload it using multiple LoRaWAN-Messages. There are two different methods this is done, according by the Payload Format you set in the Bridge's configuration.

Payload Format `0` is focused on easy reassembly of the pieces. The parts are encoded by port numbers and the data can just be concatenated together. Payload Formats `1` and `2` add additional information to the telegram. They focus on putting as much of a telegram in a single LoRaWAN-Message as possible with respecting the current Spreading Factor.

| Port | PayloadFormat | Message |
|---|---|---|
| 1 | any | Status message |
| 2 | any | Optional if SCD41 environmental sensor is installed |
| 3 | any | Optional if BME280 environmental sensor is installed |
| 11-99 | 0 | Default PayloadFormat. Part of split telegrams is encoded in Port (e.g. Port 24 = Telegram 2 of 4). |
| 101 | 1 | Data Message with timestamp and without RSSI. Part of split telegrams is encoded in payload. |
| 102 | 2 | Data Message with timestamp and with RSSI. Part of split telegrams is encoded in payload. |

### Status Packet (Port 1)

Port 1 - In order to provide some information about the health & connectivity state of the device itself, the device sends a status update at a daily basis. The status packet is sent on the first upload phase after activation of the device (after reboot) and then repeatedly in every upload phase that takes place a day or longer after the previous status packet. It has a length of 7 or 8 bytes. The battery voltages and ambient temperature are encodes as 16 bit integer using little endian encoding.

| name | type | bytes | description | example |
|---|---|---|---|---|
| version | uint8[3] | 0-2 | Version of the firmware running on the device | 1, 5, 1  v1.5.1 |
| v_bat | uint16 | 3-4 | Battery voltage in mV | 2947  2.947V |
| temp | int16 | 5-6 | Temperature measured inside the device in 1/10 °C | 246  24.6°C |
| flags | int8 | 7 | Bit 7 (e.g. 0x01) = No wMbus Telegram received (**added in v2.5.0**) | 0x01 |

# Environment Data (Port 2)

Port 2- Data from optionally installed SCD41 addon sensor. Uses BigEndian Encoding.

⚠️ Addon data only on LOB-GW-HYB-WMBUS and LOB-GW-SUN-WMBUS possible

| name | type | bytes | description | example |
|---|---|---|---|---|
| flag | uint8 | 0 | Error flag | 0  no error |
| temp | uint32 | 1-4 | m°C | 10456 10,456 °C |
| humidity | uint32 | 5-8 | mRH | 30000  30% RH |
| CO2 | uint16 | 9-10 | ppm | 400 = 400 ppm |

## Environment Data (Port 3)

Port 3 - Data from optionally installed BME280 addon sensor. Uses BigEndian Encoding.

⚠️ Addon data only on LOB-GW-HYB-WMBUS and LOB-GW-SUN-WMBUS possible

| name | type | bytes | description | example |
|---|---|---|---|---|
| flag | uint8 | 0 | Error flag | 0  no error |
| temp | uint32 | 1-4 | m°C | 10456 10,456 °C |
| humidity | uint32 | 5-8 | mrH | 30123  30,123 % rH |
| pressure | uint32 | 9-12 | Pa | 101537  101537 Pa  1015,37 hPa  1,01537 Bar |

> ⚠️ **Temperature Sensor**
>
> The temperature sensor is not present anymore on dedicated V2 hardware, instead 0xffff will be returned.

We provide a JavaScript reference implementation of a decoder for this status packet on GitHub, which can be used directly for decoding in The Things Network.

## Data Packet (Port 11-99, PayloadFormat 0) - Default

After each wMBUS collecting phase, all saved telegrams (up to 500 can be stored) will be uploaded via LoRaWAN uplink messages as fast as possible. The received wMBUS telegrams that did pass the configured white list filters will be uploaded without any modification in one or more LoRaWAN messages. If a wMBUS telegram is bigger than the bridge configuration parameter loraMaxMsgSize the transmission will be done using multiple LoRaWAN messages. This parameter is limited to  50 bytes due to LoRaWANs maximum payload size restrictions. In case of telegram splitting is needed the receiving backend application server as to reassemble the original wMBUS telegram before decryption & parsing of the meter data. This is done by simply joining the messages together in the order of receive. The LoRaWAN port encodes identifies a LoRaWAN fragment of the original wireless M-Bus telegram. This way partial messages can be identified using the LoRaWAN Port:

- 10 < LoRaWAN Port < 100  (Part Number | Total Parts)

Gaps in the LoRaWAN Frame Counter are giving a hint for missing telegram parts which can happen in LoRaWAN since it's a ALOHA based protocol, e.g. collisions and some packet losses are accepted by principle of operation. In case the backend noticed a missing packet the wMBUS telegram can't be assembled anymore as described before.

> **Note**
>
> **Reference Implementation in GoLang**

Examples

Examples (with `loraMaxMsgSize` = 50):

- A 48 Byte wMBUS telegram will be send on LoRaWAN port 11. Port 11 says it is the first message of only one message (no splitting).
- A 75 byte wMBUS telegram will be send in two messages on LoRaWAN ports 12 and 22. Port 12 means this part one of a wMBUS telegram that got splitted into two LoRaWAN messages. Port 22 means that this data is the 2nd part of the original wMBUS data. Both parts have to been concatenated in the order of receive by the backend.
- A 101 byte wMBUS telegram will be send in three messages on LoRaWAN ports 13, 23 and 33. Port 13 means this part one of a wMBUS telegram that got splitted into three LoRaWAN messages. Port 23 means that this data is the 2nd part of the original wMBUS data. Port 33 means that this data is the 3rd part of the original wMBUS data. All three parts have to been concatenated in the order of receive by the backend.

## Data Packet without RSSI (Port 101, PayloadFormat 1)

When using Payload Format 1, collected telegrams are uploaded on a single Port: 101. For each telegram there will be added the timestamp of reception. The first byte of messages on Port 101 encodes splitting of messages as follows.

Splitting

Every Uplink on Port 101 is prefixed with a single byte, where the least significant Bit indicates if that Uplink is the first part of a message, and the second least significant Bit indicates if that Uplink is the last part or a message. So there are 4 different possible values for the first Byte of an Uplink on Port 101:

| Value | Meaning |
|---|---|
| `0x03` | This Uplink is both first and final part of a message. So the remaining Bytes in this Uplink contain the whole message. |
| `0x02` | This Uplink is the last but not the first part of a message. There has been at least one Uplink before this one, that contained data that needs to be prepended to the current Uplink in order to get the full Message |
| `0x01` | This Uplink is the first but not the last part of a message. There follows at least one Uplink that contains more data to be appended to the current's data in order to get the full message. |
| `0x00` | This Uplink is neither first nor last part of a message. There has been at least one Uplink before this one that contains more data of the current Message, and there follows at least one more Uplink with data for this Message. |

So each message sent on Port 101, whether it is contained in a single Uplink or spread over multiple ones, starts with an Uplink where the least significant Bit of the first Byte is set. Each Message ends with an Uplink where the second least significant Bit of the first Byte is set. In cases where the Message fits in a single Uplink, that Uplink is both first and last Uplink, and therefore both Bits are set.

The combination of those two Bits and the Frame Counter of the Uplinks makes it possible to upload Messages of any length while allowing the receiving side to now exactly, if a Message has been transferred completely, or if part of it is missing (when there are Frame Counter values missing).

The Bridge puts as many Bytes in each Uplink as possible for the current Spreading Factor, even if the Spreading Factor changes between Uplinks because of ADR.

When the data of all Uplinks that are part of a single Message are appended in order of reception (after removing the first Byte of each Uplink), you get the payload Data of a full message.

Payload (Format 1)

The Payload Data after reassembly of the split parts consists of a 5 Byte Timestamp, that marks the point in time the Bridge did receive that telegram, followed by the Data of the Telegram. The Timestamp follows the convention of all our 40bit-Timestamps; you can find the details under Timestamp in our LoRaWAN Background Information.

Examples

For easier understanding, the wMBus-Telegram in the examples will always be `0102030405060708090a0b0c0d0e0f`.

**A message sent in a single Uplink**

```
# An Uplink of 21 Bytes on Port 101:
'03005e53f31a0102030405060708090a0b0c0d0e0f'
# Analised:
'03' -> First and Last Uplink of Message -> complete Message in this Uplink
'005e53f31a' -> Unix Timestamp 1582560026 -> 2020-02-24T16:00:26 UTC
'0102030405060708090a0b0c0d0e0f' -> wMBus Telegram
```

**A message split over two Uplinks**

```
# An Uplink of 11 Bytes on Port 101, Frame Counter 341:
'01005e53f31a0102030405'
'01' -> First Uplink of Message, more Uplinks follow
'05e53f31a0102030405' -> First Part of Message Data.
# Another Uplink of 11 Bytes on Port 101, Frame Counter 342:
'02060708090a0b0c0d0e0f'
'02' -> Last (but not first) Uplink of Message.
'060708090a0b0c0d0e0f' -> Second and final Part of Message Data.
# We Received a 'first' Part with Frame Counter 341 and a 'last'
# Part with Frame Counter 342, so we know we did not miss any
# Parts in between. We can now assembly the complete payload:
'05e53f31a0102030405060708090a0b0c0d0e0f'
# Payload anaylsed:
'005e53f31a' -> Unix Timestamp 1582560026 -> 2020-02-24T16:00:26 UTC
'0102030405060708090a0b0c0d0e0f' -> wMBus Telegram
```

**A message split over three Uplinks**

```
# An Uplink of 8 Bytes on Port 101, Frame Counter 519:
'01005e53f31a0102'
'01' -> First Uplink of Message, more Uplinks follow
'05e53f31a0102' -> First Part of Message Data.
# Another Uplink of 8 Bytes on Port 101, Frame Counter 520:
'00030405060708 09'
'00' -> Middle Part of Message, there have been some Parts already, more Uplinks follow
'03040506070809' -> Second Part of Message Data.
# Another Uplink of 7 Bytes on Port 101, Frame Counter 521:
'020a0b0c0d0e0f'
'02' -> Last (but not first) Uplink of Message.
'0a0b0c0d0e0f' -> Third and final Part of Message Data.
# Frame Counters are consecuetive, so the complete Message is:
'05e53f31a0102030405060708090a0b0c0d0e0f'
```

**Uplinks with a missing a Part**

```
# An Uplink of 8 Bytes on Port 101, Frame Counter 123:
'01005e53f31a0102'
'01' -> First Uplink of Message, more Uplinks follow
'05e53f31a0102' -> First Part of Message Data.
# Another Uplink of 7 Bytes on Port 101, Frame Counter 125:
'020a0b0c0d0e0f'
'02' -> Last (but not first) Uplink of Message.
'0a0b0c0d0e0f' -> Third and final Part of Message Data.
# Frame Counter indicates, that a Part in the middle is missing,
# so we have to drop the Message.
```

## Data Packet with RSSI (Port 102, PayloadFormat 2)

Upload Format 2 works like Upload Format 1, with the same logic for splitting messages, but uploads are sent on Port 102. The Payload consists of a 5 Byte Timestamp marking the time of reception, followed by a `uint_8` that holds the (negated) RSSI value for that reception, followed by the Data of the Telegram.

Examples

```
# An Uplink of 22 Bytes on Port 102:
'03005e53f31a3f0102030405060708090a0b0c0d0e0f'
# Analised:
'03' -> First and Last Uplink of Message -> complete Message in this Uplink
'005e53f31a' -> Unix Timestamp 1582560026 -> 2020-02-24T16:00:26 UTC
'3f' -> 63 -> RSSI of wMBus reception = -63
'0102030405060708090a0b0c0d0e0f' -> wMBus Telegram
```

Upload Speed / Duration

The bridge has to work in compliance with the European SRD 868 1% duty-cycle regulations. This implies as a rule of thumb the device can upload at most wMBUS telegrams via LoRaWAN for 36 seconds every hour.

The actual transmit time ('ToA: time on air') for each LoRaWAN message depends on the byte size and the used LoRa spreading factor (SF) which defines how redundant LoRa data is send. This means a device with good connectivity and consequently using LoRa SF7 (ToA  0,050s) can upload much faster more data than a node using LoRa SF11 (ToA  1s) due to a hard to reach LoRaWAN gateway. The bridge will upload in conformity with the regulations automatically as fast as possible. When it has to wait it enters a low power sleep mode until the next transmission is possible again. The next data collection phase will be started only after completion of the previous upload phase in respect to the configured `listenCron` parameter. Because of this it is advisable to define the cron parameter with an estimation of the upload duration in mind. This will avoid unexpected 'skipping' of data collection phases.

Downlinks

| Port | Message |
|---|---|
| 128 | Remote Confiuration |
| 132 | wMbus Bridge Commands |

Remote Configuration (Port 128)

Update of Configuration parameters is documented in our LoRaWAN downlink messages documentation.

**Supported downlink messages:**

| Char | Command | Parameter | Hex | Version required |
|---|---|---|---|---|
| ? | Request firmware and version | None | 3F | |
| g | Get config parameter value | `<name>` | 67 | |
| r | Reset config parameter value | `<name>` | 72 | |
| s | Set config parameter value | `<name>=<value>` | 73 | |
| S | Set config parameter value + Save and reboot | `<name>=<value>` | 53 | ??? |
| a | Append to config parameter value | `<name>=<value>` | 61 | |
| b | Reboot device without saving | None | 62 | |
| w | Save config and reboot device | None | 77 | |

- <name> is the ASCII encoded name of the parameter
- <value> is the ASCII encoded value

Special Commands (Port 132)

| Port | Action | FW Version | Payload (ASCII) | Payload (Hex) | Payload Base64 |
|---|---|---|---|---|---|
| 132 | Ad-hoc readout | **> 2.4.0** | `read` | `72656164` | `cmVhZA==` |

Ad-hoc readout

A downlink that triggers an Ad hoc readout, independent of CRON triggers. The Ad-hoc readout is using the same parameters (filters and listening duration) as a CRON triggered readout.

Decoding wMBUS telegrams

After receiving the raw wireless M-Bus telegrams from your LoRaWAN network provider the actual metering data has to be decrypted and decoded by a backend service for further processing. The details of this are described in the EN 13757 norm and the newer **OMS** specification, which is a clarification of the original underlying norm.

A universal wireless M-Bus decoder is a relatively complicated piece of software if you start implementing it from scratch since the norm covers many different use cases, units, meter types and data formats. If you know in advance the exact telegram format of the deployed meters in your setup a hard coded data decoding may be a feasible approach. This is because wireless M-Bus devices often send the same telegram format in every transmission. Please contact the manufacturer of your meters for the needed telegram format details.

An an alternative to support a quick evaluation of our hardware Lobaro offers a easy to use webservice which is designed to decode all sorts of wMBUS input data including decryption if the correct key has been provided. You can access the decoder service for free during testing. The API can be licensed for production usages.

**Free online wMBUS decoder (for testing)**

- **Lobaro wMBUS Online Parser**
- **Lobaro wMBUS REST API**

**Your meter fails to parse correctly?**

Since wireless MBUS is a complex and grown specification some meters may fail to decode correctly. We try to fix any decoding issues as quickly as possible if you report us problems with your specific wMBUS device.

# Example Parser

TTN / Chirpstack / Lobaro Platform (see wrapper functions)

```
function readVersion(bytes, i) {
    if (bytes.length < 3) {
        return null;
    }
    return "v" + bytes[i] + "." + bytes[i + 1] + "." + bytes[i + 2];
}

function parse_sint16(bytes, idx) {
    bytes = bytes.slice(idx || 0);
    var t = bytes[0] << 8 | bytes[1] << 0;
    if( (t & 1<<15) > 0){ // temp is negative (16bit 2's complement)
        t = ((~t)& 0xffff)+1; // invert 16bits & add 1 => now positive value
        t=t*-1;
    }
    return t;
}

function Decoder(bytes, port) {
    // Decode an uplink message from a buffer
    // (array) of bytes to an object of fields.
    var decoded = {};

    if (port === 9) {
        decoded.devStatus = bytes[0];
        decoded.devID = bytes[1] | bytes[2] << 8 | bytes[3] << 16 | bytes[4] << 24;
        decoded.dif = bytes[5];
        decoded.vif = bytes[6];
        decoded.data0 = bytes[7];
        decoded.data1 = bytes[8];
        decoded.data2 = bytes[9];
    }

    // example decoder for status packet by lobaro
    if (port === 1 && bytes.length == 9) { // status packet - old
        decoded.FirmwareVersion = String.fromCharCode.apply(null, bytes.slice(0, 5)); // byte 0-4
        decoded.Vbat = (bytes[5] | bytes[6] << 8) / 1000.0; // byte 6-7 (originally in mV)
        decoded.Temp = parse_sint16(bytes,7) / 10.0; // byte 8-9 (originally in 10th degree C)
        decoded.msg = "Firmware Version: v" + decoded.FirmwareVersion + " Battery: " + decoded.Vbat + "V
Temperature: " + decoded.Temp + "°C";
    } else if (port === 1 && bytes.length >= 7) {
        decoded.FirmwareVersion = readVersion(bytes, 0); // byte 0-2
        decoded.Vbat = (bytes[3] | bytes[4] << 8) / 1000.0; // originally in mV
        decoded.Temp = parse_sint16(bytes,5) / 10.0; // byte 8-9 (originally in 10th degree C)
```

```
        decoded.msg = "Firmware Version: " + decoded.FirmwareVersion + " Battery: " + decoded.Vbat + "V
Temperature: " + decoded.Temp + "°C";
        if (bytes.length == 8) { // added in v2.5.0
            decoded.Flags = bytes[7];
        }
    }


    return decoded;
}

// Wrapper for Lobaro Platform
function Parse(input) {
    // Decode an incoming message to an object of fields.
    var b = bytes(atob(input.data));
    var decoded = Decoder(b, input.fPort);

    return decoded;
}

// Wrapper for Loraserver / ChirpStack
function Decode(fPort, bytes) {
    return Decoder(bytes, fPort);
}

// Wrapper for Digimondo niota.io
// Uncomment only when used in niota!
/*
module.exports = function (payload, meta) {
    const port = meta.lora.fport;
    const buf = Buffer.from(payload, 'hex');

    return Decoder(buf, port);
}*/
```