# LoRaWAN Modbus Gateway

The Lobaro LoRaWAN Modbus Gateway is a **LoRaWAN Gateway** with integrated LoRaWAN Network Server **providing sensor data via Modbus**.

## Hardware Components

| LoRaWAN Gateway | |
| --- | --- |
| **IMST Version** | **RAK Version** |
| | |
| Currently not available | New Version based on RAK Wireless |
| <ul><li>LoRaWAN</li><li>Connectivity: LAN</li><li>Order number: 8000101</li><li>Type: LOB-GW-MODBUS-LW-IMST</li></ul> | <ul><li>LoRaWAN</li><li>Connectivity: LTE / LAN / WLAN</li><li>Order number: 8000202</li><li>Type: LOB-GW-MODBUS-LW-RAK</li></ul> |

| USB-Modbus Adapter |
| --- |
| |

## Software Components

- Chirpstack Network Server
  - Semtech Packet Forwarder
  - Chirpstack Gateway Bridge
  - Chirpstack Network Server

- Chirpstack Application Server
- Postgres
- Redis
- Lobaro Modbus Server

Usually you will not need to change anything inside the Chirpstack Application Server. All devices are managed by the Lobaro Modbus Server.

# Remote access

Per default the gateway obtains the IP address via DHCP. If configured with a fixed IP address, the gateway has a label with the configured IP address and subnet.

## SSH Access

The gateway can be accessed via SSH on port 22. Default login credentials are:

- User: pi
- Password: lobarogw
- IP: DHCP with fallback to 192.168.0.1/24 (IMST) or 192.168.0.1/24 (RAK)
  - On RAK with latest image also possible via WLAN AP:
    - Default SSID "RAKMBG_XXYY" (XX and YY last bytes of WLAN adapter MAC) and password "lobarowireless", RAK IP 192.168.230.1

## Management UI

- IMST version: http://192.168.100.26:8081/
- RAK version:
  - LAN: http://192.168.0.1:8081/
  - WLAN: http://192.168.230.1:8081/
- User & Password: Same as Chirpstack

## Chirpstack

- http://192.168.0.1:8080/ or IP from DHCP
- User: admin
- Password: lobarogw

# SD Card write protection

⚠️  Write protection on the SD card was removed in current firmware releases

To change any filed on the SD Card (including all config files) you need to execute the script:

```
~/enableWriteAccess.sh
```

To disable write access, restart the gateway or execute:

```
~/disableWriteAccess.sh
```

# Lobaro Modbus Server

The Lobaro Modbus Server (`lobaro-modbus-server`) is responsible for fetching data from the local LoRaWAN Network Server and provides received data via modbus.

ⓘ  `vim` can be used to edit files.

To use WinSCP with the user "pi" the files need write access:

```
sudo chmod o+wr /etc/lobaro-modbus-server/lobaro-modbus-server.yml
```

Open or change configuration of the Lobaro Modbus Server:

```
sudo vim /etc/lobaro-modbus-server/lobaro-modbus-server.yml
```

After editing the configuration `lobaro-modbus-server` must be restarted:

```
sudo systemctl restart lobaro-modbus-server
```

Check the status with

```
sudo systemctl status lobaro-modbus-server
```

Check the logs with

```
sudo journalctl --no-pager -e -u lobaro-modbus-server
```

## Weitere Services

There are other services running to operate the gateway.

- lobaro-modbus-server
- redis-server
- postgresql / postgresql@9.6-main.service
- mosquitto
- IoTSemtech
- chirpstack-gateway-bridge
- chirpstack-network-server
- chirpstack-application-server

Useful commands:

```
# Status:
sudo systemctl status <service-name>
# Start / Stop / Restart
sudo systemctl start <service-name>
sudo systemctl stop <service-name>
sudo systemctl restart <service-name>
# Logs
sudo journalctl --no-pager -e -u <service-name>
```

### Debugging

Beside checking the logs, you can also analyze the files in the `dataDir` (see config).

There are two files that help writing the config and checking the results.

- `device-data.json` - contains raw json received from Chirpstack. Can be used to verify `mapping[].register.value` configuration.
- `register-map.json` - contains all register values provided via modbus. All values are formatted as `int`.

Example `device-data.json`:

```
{
  "0000000000000000-1": {
    "adr": true,
    "applicationID": "1",
    "applicationName": "default",
    "data": "AAMEAOQN1g==",
    "devEUI": "0000000000000000",
    "deviceName": "0000000000000000",
    "fCnt": 1,
    "fPort": 1,
    "object": {
      "temp": 22.8,
      "vBat": 3.542,
      "version": "v0.3.4"
    },
    "rxInfo": [
      {
        "gatewayID": "0000000000000000",
        "loRaSNR": 8.8,
        "location": {
          "altitude": 0,
          "latitude": 0,
          "longitude": 0
        },
        "name": "default",
        "rssi": -36,
        "uplinkID": "ce2e086a-d747-4813-9428-b7a4a45abcc8"
      }
    ],
    "txInfo": {
      "dr": 0,
      "frequency": 868300000
    }
  }
}
```

Example `register-map.json`:

```
{
  "Register": {
    "100": {
      "Val": 0,
      "Type": 1,
      "UpdatedAt": "2020-02-07T13:44:03.39750918Z"
    }
  }
}
```

## Configuration file

### Register Types

For `mapping.[device].register.type` the following types are valid:

| "type" Parameter | Register Count |
| --- | --- |
| int16 | 1 |
| uint16 | 1 |
| uint32 | 2 |
| int32 | 2 |
| float32 | 2 |

| downlink | 1 |
| --- | --- |

**Example Configuration**

```
# The application stores persistent data at this path
dataDir: /mnt/ssd/var/data/lobaro-modbus-server/

# Chipstack configuration. Required to manage configured LoRaWAN devices.
chirpstack:
  server: http://localhost:8080
  broker: localhost
  appId: 1
  username: admin
  password: admin

# Modbus configuration.
# <v1.2.0: Serial mode is fixed at: 8 Data bits, Even Parity, 1 Stop bit
(8E1)
modbus:
  baud: 19200
  dataBits: 8 # since v1.2.0
  parity: "even" # no, even (default), odd - since v1.2.0
  stopBits: 1 # 1 (default), 1.5, 2 - since v1.2.0
  slaveId: 1
  port: /dev/ttyUSB0

# Mapping from LoRaWAN Sensors to Modbus Registers
mapping:
  # LoRaWAN Sensor parameters
  - devEUI: 0000000000000000
    appKey: 00000000000000000000000000000000
    # Chirpstack Device Profile to use. Includes the Payload Parser.
    devProfile: lobaro-environment
    devName: "name of device in chirpstack"
    # Register mapping for this device
    # One device can fill any number of registers.
    # The server will check for overlapping definitions on start.
    register:
        # Modbus Address (do NOT prefix with 0, else it's octal)
      - addr: 1
        # The value to be mapped.
        # Usually the value is taken from the Chirpstack Parser result JSON
        # and can be selected via JSON Path as handled by https://github.
com/tidwall/gjson
        # There are some special values:
        # @age - age of last update in minutes (for any register of this
device)
        # @now - Current time as Unix Timestamp
        value: "@age" # age of last update in minutes (for any register of
this device)
        # Data type of the value. Default byte order is LittleEndian
        # Supported types are: int16, uint16 (more will come in future
versions)
        type: int16
        # The value is only for messages on the specified port, 0 for
"every". Default: 0
        port: 0
        # The register value is multiplied with the given factor, 0 is
irgnored. Default: 1
        factor: 1

      - addr: 2
        port: 1 # status packet
        value: "object.vBat"
        type: int16
        factor: 1000
      - addr: 3
        port: 2
```

```
          value: "object.temperature"
          type: int16
          factor: 10
       - addr: 4
          port: 2
          value: "object.humidity"
          type: int16
          factor: 10
       - addr: 5
          port: 2
          value: "object.pressure"
          type: int16
          factor: 10
       - addr: 6
          port: 2
          value: "rxInfo.0.rssi"
          type: int16
       - addr: 7
          port: 2
          value: "txInfo.dr"
          type: int16
            - addr: 8
                   port: 128
                   value: "0x0102" # value to be sent as downlink, either as
hex (prefixed with "0x") or base64 string
                   type: downlink

  # A second device as example
  - devEUI: 0000000000000000
    appKey: 00000000000000000000000000000000
    devProfile: lobaro-one-wire
    register:
       - addr: 100
          value: "@age" # age of last update in minutes (for any register of
this device)
          type: int16
       - addr: 101
          port: 1 # status packet
          value: "object.vBat"
          type: int16
          factor: 1000
       - addr: 102
          port: 2
          value: "object.sensors.0.temp"
          type: int16
          factor: 10
       - addr: 103
          port: 2
          value: "rxInfo.0.rssi"
          type: int16
       - addr: 104
          port: 2
          value: "txInfo.dr"
          type: int16
       - addr: 105
                   port: 128
                   value: "dGVzdF9kb3dubGluaw==" # value to be sent as
downlink, either as hex (prefixed with "0x") or base64 string
                   type: downlink
            - addr: 106
          type: downlink-var
          len: 5 # 107-111
```

## Send Fixed Downlinks

Send a predefined downlink packet to a configures port to a LoRaWAN device with a single write to a modbus register.

```
mapping:
    register:
      - addr: 8
        port: 128
        value: "0x0102" # value to be sent as downlink, either as hex
(prefixed with "0x") or base64 string
        type: downlink
```

Set the register to type "downlink" to allow sending the "value" via LoRaWAN to the deivce.

**With the example above:**

- Write any value to modbus register with address 8
- A downlink "0x0102" will be queued on port 128
- The register will keep the written value in case of success. In case of error the value will be 0.

## Send Variable Downlinks

Send variable downlinks to a variable ports. Payload must be written to a set of defined modbus registers per LoRaWAN Device.

First, specify a new register under the device of your choice with type `downlink-var`. Then, set the following options for it:

- `Len`: How many modbus registers (following this register) should be assigned as storage for the variable downlink.
- `Confirm`: Whether to ask the device for acknowledgement of reception of the sent downlink (`false` or `true`)

```
mapping:
        register:
          - addr: 123
        type: downlink-var
        len: 7 # storing in registers 124-130
        confirm: true
```

The resulting structure at the configured address *addr* will be:

| Downlink Trigger | Downlink data | ... | Downlink data |
|---|---|---|---|
| addr | addr+1 | ... | addr+len |

After (re-)starting the modbus server to apply the new config:

1. Write your desired downlink (bytes) to the downlink data registers, starting at *addr+1*.
   a. You can only write 2 * *len* bytes at maximum! Extra bytes will be discarded.
2. Optional: Write 0x0000 to the downlink trigger register at *addr* to clear the register (no downlink will be queued!)
3. Write *<port><length>* to the downlink trigger register at *addr*, using 1 byte each for the designated FPort and downlink length **in bytes**.
   a. Example: Send downlink to port 128 (*0x80*), 10 (*0x0A*) bytes long: Write value `0x800A` to register *addr*.

To check if the downlink was successful, read the trigger register and check its value:

- *Register value = <port><length>*: The downlink was successfully queued.
- *Register value = 0*: The downlink couldn't be queued for your device. Check the server log for more details.

# Chirpstack

The gateway uses a local Chirpstack server. Access management interface on `https://<gw-ip>:8080`.

Documentation can be found on Chripstack.io.

For each type of device the `lobaro-modbus-server` needs to reference a Device Profile. See: Chirpstack Device Profile Management. The Device Profile of each LoRaWAN device must be referenced by its name or UUID in the `lobaro-modbus-server.yml` config file.

## Gateway administration

When ever any file on the SD-Card need to change make sure to execute

```
~/enableWriteAccess.sh
```

### Change password

Login via SSH (see: Remote access)

```
passwd
```

### Change IP address

#### IMST

```
sudo vim /etc/network/interfaces
```

```
pi@LoRaGateway:~ $ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

# DHCP (Default, comment line to disable DHCP)
iface eth0 inet manual

# Fixed IP (Uncomment to enable or use /etc/dhcpcd.conf)
#auto eth0
#iface eth0 inet static
#    address 10.0.0.42/24
#    gateway 10.0.0.1
```

#### RAK

```
sudo vim /etc/dhcpcd.conf
```

Edit last lines to:

```
# RAK_eth0_IP
profile static_eth0
static ip_address=192.168.0.1/24
static routers=192.168.0.1

interface eth0
fallback static_eth0
```