EDL21 electricity meter bridge (LoRaWAN)

The EDL21 over LoRaWAN bridge is a device that can be used to readout modern utility meters with standardized infrared "INFO" interface.

These type of meters are called in Germany to be "EDL21"-compatible - hence the name. The meter outputs over its infrared "INFO" interface a serial protocol conforming to the Smart Meter Language Protocol 1.04 (SML). This interface is intended to be used by end-users and **not** for billing purposes of the electricity supplier. The read information normally contains the current consumption values of the meter and gets interpreted and forwarded by the EDL21 bridge via a LoRaWAN network to web based applications interested in further processing this data.

Warning

Older meters with "infrared pulse" output are **not** compatible to the Lobaro EDL21 bridge. Please check our list of compatible meters to make sure it is equipped with the correct interface.

Consider using the latest firmware on your hardware

- **See available firmware downloads**

Top Features

- ✓ LoRaWAN 1.0.x and 1.1 network servers supported
- ✓ LoRaWAN time synchronisation
- ✓ Configuration via USB or remotely via LoRaWAN downlink
- ✓ Compatible with many electrical utility meters
- ✓ Configure up to 25 Obis Codes to be read
- ✓ RGB Status LED
- ✓ Variant with external power-supply available on request
- ✓ Separation of infrared readout head and LoRaWAN antenna possible

Compatible utility meters

Any meter that adheres to the standard can be read. The following list contains meters that we successfully tested.

Electricity meter	Manufacturer
DTZ541-ZEBA	Holley
LK13 series	Logarex
OpenWay® 3.HZ	iTron

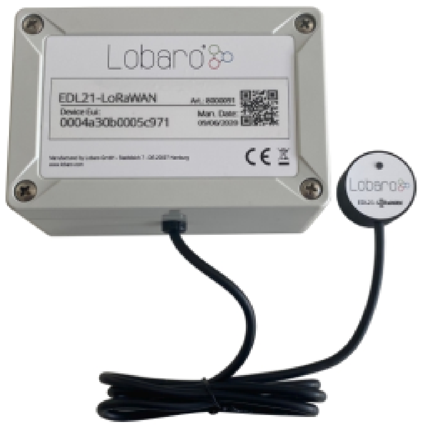
SGM-C4 series	efr
SGM-D series	efr
eHZ-K series	EMH
mMe4.0 series	EMH
ED300 series	EMH
eBZD series	EMH
E320	Landis+Gyr
MT681	ISKRA

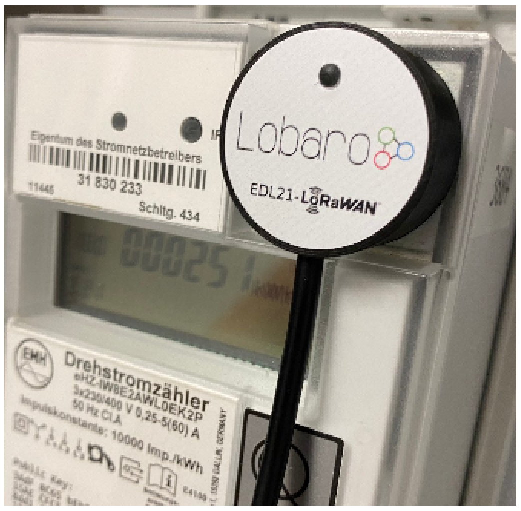
Alternative IEC 62056-21 protocol

On request we offer also the integration of electricity meters using the D0 interface conforming to **IEC 62056-21**. This interface is not compatible to the SML protocol. Please contact us if you need an offer for a custom firmware supporting your meter of interest.

Product variants

```
EDL21-LoRaWAN Bridge (universal head, XH battery connector, int. Ant.), Order
number: 8000091
```





Customization Options

The product variant shown above is the *standard* variant for use with a ER34614 3.6V D-cell Battery connected over a XH connector.

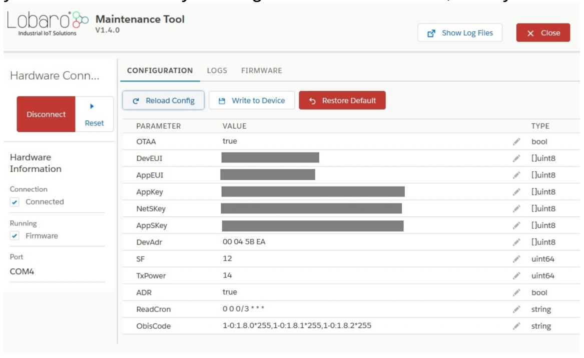
Other power supply options & housing are available on request

- External antenna
- External power-supply
- NB-IoT instead of LoRaWAN
- Different infrared data formats other than SML

Contact us via support@lobaro.de if you need our offer for a special variant.

Quickstart

1. Connect to the device with the Lobaro Tool using the Lobaro Config Adapter
2. Under Configuration click "Reload Config" and change the fields ReadCron and ObisCode as you need followed by clicking on "Write to Device", here you can see a configuration example.



PARAMETER	VALUE	TYPE
OTAA	true	bool
DevEUI	[redacted]	[]uint8
AppEUI	[redacted]	[]uint8
AppKey	[redacted]	[]uint8
NetSKey	[redacted]	[]uint8
AppSKey	[redacted]	[]uint8
DevAdr	00 04 5B EA	[]uint8
SF	12	uint64
TxPower	14	uint64
ADR	true	bool
ReadCron	0 0 0/3 * * *	string
ObisCode	1-0:1.8.0*255,1-0:1.8.1*255,1-0:1.8.2*255	string

3. Register the device in your LoRaWAN network
4. Connect ER34614 3.6V D-cell Battery via XH connector / Connect external powersupply
5. If not connected to anything the red LED will start blinking as long as no data is received, after 1 minute it will sleep for 15 seconds after every 5 retries
6. Tighten the screws and install the bridge beside your electric meter
7. Place the EDL21 opto head on the "Info" interface
8. As soon as the EDL21 receives data its green LED will light up for 5 seconds, when connected to a LoRa Network its blue LED will light up for 5 seconds
9. Check the sent data (port 3), if payload is zero the EDL21 was not able to read data, recheck proper alignment

Configuration

The configuration is done using Lobaro Maintenance Tool and the Lobaro USB PC adapter.

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.

Downlink Configuration

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
`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
`ADR`	Use Adaptive Data Rate	`bool`	`true`= use ADR, `false`= don't
`TimeSync`	Days after which to sync time	`int`	days, `0`=don't sync time
`RndDelay`	Random delay before sending	`int`	max seconds
`RemoteConf`	Not supported by this firmware	`bool`	`false`=deactivate
`LostReboot`	Days without downlink before reboot (triggers downlinks)	`int`	days, `0`=don't reboot

Operation

Configuration values defining the behaviour of the device.

name	description	example value
`ReadCron`	Cron expression defining when to read	`0 0/15 * * * *` for every 15 minutes
`ObisCode`	Comma separated list of ObisCodes to select a subset of the available information	`1-0:1.7.255*255` = Leistung (Momentan)
`PayloadFormat`	Format used for data upload (include timestamps or not)	1=no timestamp, 2=include timestamp

See also our Introduction to Cron expressions and our Introduction to Obis Codes.

LED blinking patterns

The following pattery are explained in the order in which they appear after initial power on / reset of the device.

color	duration	description
`red/green/blue`	300ms each	initial pattern after reset
`red/green`	1s	NEW in 0.3.2: single readout success/failure before OTAA join
`red`	short, blinking	trying to receive meter optical data for the first time after OTAA join
`green`	5 seconds	successfully received meter optical data
`blue`	5 seconds	LoRaWAN network join
`blue`	short	sending LoRaWAN data uplink
`off`	-	low-power mode until next sendout cycle

As you can see by this the device will start the LoRaWAN join only after receiving optical data at least once.

Payload Format Status Packet (Port 1)

Once per day a status packet will be sent. It contains basic information about the device. The Battery Voltage is transmitted in 1/1000 V and the temperature in 1/10 °C. Both are in Big Endian byte order.

Version Major	Version Minor	Version Patch	Flags	Battery Voltage	Temperature

1 byte	1 byte	1 byte	1 byte	2 byte	2byte
unsigned	unsigned	unsigned	unsigned	unsigned	signed

Payload

Payload Format 1 (default, Port 3, with exponent)

This Format is used, when the configuration parameter `PayloadFormat` is set to `1` (which is the default value).

The payload consists of multiple entries, one entry per OBIS code given in the configuration. Each entry follows the following structure:

OBISCode (hex)	length of value (n)	value	exponent
6 bytes	1 byte	n bytes, LSB first	1 byte (signed)

Example packet: 0100010800FE08FF01000000000000 ff 0100010800FE08FF02000000000000 02

Entry 1:

OBISCode (hex)	length of value (n)	value	exponent
0100010800FE	08	FF01000000000000	ff
1-0:1.8.0*254	8	511	-1
Value = 511 * 10^-1 = 51.1			

Entry 2:

OBISCode (hex)	length of value (n)	value	exponent
0100010800FE	08	FF02000000000000	02
1-0:1.8.0*254	8	767	2
Value = 767 * 10^2 = 76700			

Multiple messages

The Bridge puts as many values in a single data message as possible (respecting the current Spreading Factor). When it cannot fit all values in a single message, it will send multiple data messages until all values are uploaded. It will never split a single value. Since every value is prefixed with the Obis code, the parser can easily assign values to Obis codes.

Payload Format 2 (extended, Port 4, with timestamp)

This Format is used, when the configuration parameter `PayloadFormat` is set to 2.

The Obis Codes and Data Values are transmitted as in Payload Format 1, but each uploaded LoRaWAN-Message with data is prefixed with a 5 byte Timestamp, indicating when the values where requested from the attached meter. This allows for a more precise timing information then using the time of reception, as the upload can be delayed quite heavily due to our random delay feature and potentially due to duty cycle restrictions. The timestamp also makes it easy to reassociate values from multiple uplinks to a single reading, when multiple uplinks must be used to upload all values. If a readout is spilt over multiple uplinks (because of LoRaWAN's length restrictions), every uplink from that reading will have the same timestamp (which is the time of requesting the values from the meter).

The Timestamp is sent as a UNIX-Timestamp encoded as a bigendian signed 40-bit number.

Timestamps might not represent real time

Be aware that the timestamps are generated using the device's internal clock, which does not necessarily represent real time! If you want exact timestamps, you should use the TimeSync feature that synchronises the device's clock over LoRaWAN. Be sure that your network server supports it. Even with incorrect clock, the timestamps can still be used to reassociate split uploads.

For more information, please refer to the section on Timestamps in LoRaWAN of our LoraWAN background article.

Payload Format 0 (legacy, Port 2, without exponent)

⚠️ This payload was used by previous versions of the firmware and is not supported in the current version. ⚠️

The payload consists of multiple entries, one entry per OBIS code given in the configuration. Each entry follows the following structure:

OBISCode (hex)	length of value (n)	value
6 bytes	1 byte	n bytes, LSB first

Example packet: `0100010800FE08FF01000000000000 0100010800FE08FF02000000000000`

Entry 1:

OBISCode (hex)	length of value (n)	value
0100010800FE	08	FF01000000000000
1-0:1.8.0*254	8	511

Entry 2:

OBISCode (hex)	length of value (n)	value
0100010800FE	08	FF02000000000000
1-0:1.8.0*254	8	767

Reference decoder

This is a decoder written in JavaScript that can be used to parse the device's LoRaWAN messages. It can be used as is in The Things Network.

```
function readName(bytes, i) {
    return bytes.slice(i, i + 6);
}

function readValue(len, bytes, i) {
    if (len <= 0) {
        return [];
    }
    return bytes.slice(i, i + len);
}

function toHexString(byteArray) {
    var s = '';
    byteArray.forEach(function (byte) {
        s += ('0' + (byte & 0xFF).toString(16)).slice(-2);
    });
    return s;
}

function signed(val, bits) {
    if ((val & 1 << (bits - 1)) > 0) { // value is negative (16bit 2's
complement)
        var mask = Math.pow(2, bits) - 1;
        val = (~val & mask) + 1; // invert all bits & add 1 => now
positive value
        val = val * -1;
    }
    return val;
}
function uint40_BE(bytes, idx) {
    bytes = bytes.slice(idx || 0);
    return bytes[0] << 32 |
        bytes[1] << 24 | bytes[2] << 16 | bytes[3] << 8 | bytes[4] << 0;
}
function uint16_BE(bytes, idx) {
    bytes = bytes.slice(idx || 0);
    return bytes[0] << 8 | bytes[1] << 0;
}
function int40_BE(bytes, idx) {return signed(uint40_BE(bytes, idx), 40);}
function int16_BE(bytes, idx) {return signed(uint16_BE(bytes, idx), 16);}
function int8(bytes, idx) {return signed(bytes[idx || 0], 8);}

function toNumber(bytes) {
    var res = 0;

    for (var i = bytes.length-1; i >= 0 ; i--) {
        res *= 256;
        res += bytes[i];
    }

    return res;
}

function readVersion(bytes) {
    if (bytes.length<3) {
        return null;
    }
    return "v" + bytes[0] + "." + bytes[1] + "." + bytes[2];
}

function decodeStatus(bytes) {
    var decoded = {
        "version":readVersion(bytes),
        "flags": bytes[3],
        "vBat": uint16_BE(bytes, 4) / 1000,
        "temp": int16_BE(bytes, 6) / 10,
    };

    return decoded;
}

function decodeSmlValuesV1(bytes) {
    var decoded = {
        values: [],
    };

    if (bytes.length === 1) {
        // No Data! Read error?
        return decoded;
    }

    var pos = 0;
    while (pos < bytes.length) {
        var name = readName(bytes, pos);
        pos += 6;
        var len = bytes[pos];
        pos += 1;
        var value = readValue(len, bytes, pos);
        pos += len;

        var val = {
            nameHex: toHexString(name),
            len: len,
            value: toNumber(value),
            valueHex: toHexString(value)
        };

        decoded.values.push(val);
    }

    return decoded;
}

function decodeSmlValuesV2(bytes) {
    var decoded = {
        values: [],
    };

    if (bytes.length === 1) {
        // No Data! Read error?
        return decoded;
    }

    var pos = 0;
    while (pos < bytes.length) {
        var name = readName(bytes, pos);
        pos += 6;
        var len = bytes[pos];
        pos += 1;
        var value = readValue(len, bytes, pos);
        pos += len;
        if (len > 0) {
            var exponent = int8(bytes, pos);
            pos += 1;
        }
        var val;
        if (len > 0) {
            val = {
                nameHex: toHexString(name),
                len: len,
                value: toNumber(value) * Math.pow(10, exponent),
                valueHex: toHexString(value),
            }
        } else {
            val = {
                nameHex: toHexString(name),
                len: len,
                value: toNumber(value),
                valueHex: toHexString(value),
            }
        }

        decoded.values.push(val);
    }

    return decoded;
}

function decodeSmlValuesV3(bytes) {
    // Like V2, but with 5B timestamp as prefix:
    var decoded = decodeSmlValuesV2(bytes.slice(5));
    decoded.time = int40_BE(bytes, 0) * 1000;
    return decoded;
}

function Decoder(bytes, port) {
    // Decode an uplink message from a buffer
    // (array) of bytes to an object of fields.
    switch (port) {
        case 1:
            return decodeStatus(bytes);
        case 2:
            return decodeSmlValuesV1(bytes);
        case 3:
            return decodeSmlValuesV2(bytes);
        case 4:
            return decodeSmlValuesV3(bytes);
    }
}
```

Example parser result

```
Test input (Port 3): 0100010800FE08FF01000000000000FF

{
  "values": [
    {
      "len": 8,
      "nameHex": "0100010800fe",
      "value": 51.1,
      "valueHex": "ff01000000000000"
    }
  ]
}
```

Appendices

Technical characteristics

Product	
Type name	LOB-S-EDL21-LW
Description	Electricity meter over LoRaWAN Bridge
RF transceiver	
Type	Semtech SX1272
Frequency	863 MHz to 870 MHz
Max. TX Power	max. +14 dBm
Typical RF Range	2km
Ideal RF Range	10km (free line of sight)
LoRa communication	
Protocol	Class A LoRaWAN 1.0.1 EU868
Activation method	Over-the-air-activation (OTAA) Activation by personalization (ABP)
Encryption	AES128
Environmental Requirements	
Operating temperature	-20°C – 55°C
Max installation height	2m
Standards	

Disposal / WEEE / Entsorgung

Information about the disposal of the Device.

CE Declaration of Conformity

CE Declaration of Conformity (pdf).