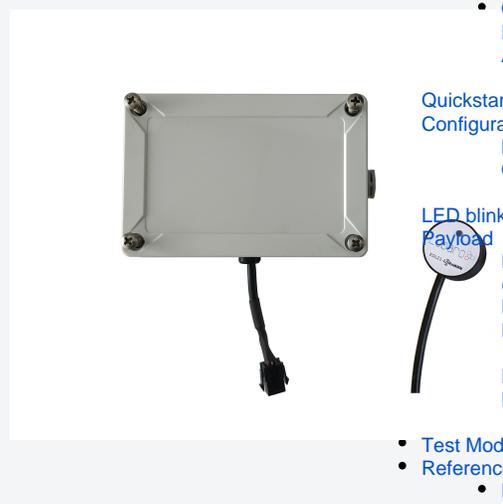


# EDL21 electricity meter bridge V2 (LoRaWAN)

		<ul style="list-style-type: none"><li>• <a href="#">Top Features</a></li><li>• <a href="#">Compatible utility meters</a></li><li>• <a href="#">Product variants</a></li><li>• <a href="#">Additional Equipment</a></li></ul> <p><a href="#">Quickstart</a> <a href="#">Configuration</a> <a href="#">LoRaWAN</a> <a href="#">Operation</a></p> <p><a href="#">LED blinking patterns</a> <a href="#">Payload</a></p> <ul style="list-style-type: none"><li>• <a href="#">Payload Format Status Packet (Port 64)</a></li><li>• <a href="#">Payload Format 1 (default, Port 3, with exponent)</a></li><li>• <a href="#">Multiple messages</a></li><li>• <a href="#">Payload Format 2 (extended, Port 4, with timestamp)</a></li></ul> <ul style="list-style-type: none"><li>• <a href="#">Test Mode</a></li><li>• <a href="#">Reference decoder</a><ul style="list-style-type: none"><li>• <a href="#">Example parser result</a></li></ul></li></ul>
Powered by Current Transformer	Battery Powered (Battery not included)	<ul style="list-style-type: none"><li>• <a href="#">Appendices</a></li><li>• <a href="#">Technical characteristics</a></li><li>• <a href="#">Disposal / WEEE / Entsorgung</a></li><li>• <a href="#">CE Declaration of Conformity</a></li></ul>
Order Number: <b>8000160</b>	Order Number: <b>8000161</b>	

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 Lobar 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
- ✓ Separation of infrared readout head possible
- ✓ Testmode for protocol and supported obis code identification

## 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, current transformer, int. Ant.), Order number: 8000160



### **i** Customization Options

The product variant shown above is the *standard* variant.

**Other power supply options & housing are available on request**

- External antenna
- ER34614 3.6V D-cell Battery
- External power-supply
- NB-IoT instead of LoRaWAN
- Different infrared data formats other than SML

Contact us via [support@lobaro.de](mailto:support@lobaro.de) if you need our offer for a special variant.

## Additional Equipment

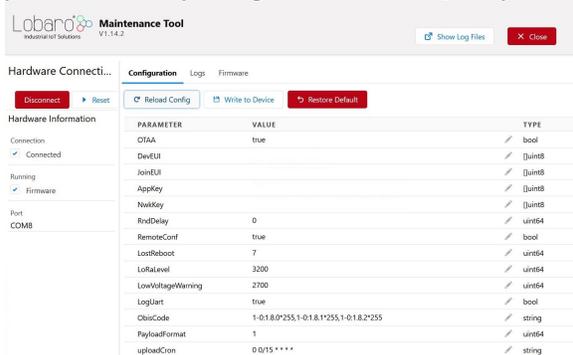
Battery Pack, Order Number: 8000164



The optional Battery Pack has 2 functions. The main function is to supply the device with energy for initial testing without having to wait for the current transformer. After connecting the battery pack for the first time it will take about 30 seconds to load the bridges supercapacitor. The secondary function is to provide a simple way to reset the device with an inbuilt magnet. If the red dot on the battery pack is brought near the bridge the bridge will signal its presence by activating its LED. After 1 second it will blink rapidly and reset.

## 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.



3. Register the device in your LoRaWAN network
4. Place the EDL21 opto head on the "Info" interface
5. Install Current Transformer and connect it to the device (or for a quick check use the optional battery pack)
6. 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
7. If data is available it will be transmitted on port 3 or 4 depending on the chosen format. If not a status message will be sent. Check the Error Flags for possible reasons.

## 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.

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]	
RndDelay	Random delay before sending	int	max seconds
RemoteConf	Enable Remote Configuration	bool	true=activate, 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	Type	Values
LoRaLevel	Minimum Voltage for packet transmission in mV	int	e.g. 3200
LowVoltage Warning	Voltagelevel that will trigger a Status Message with enabled Warning Flag	int	e.g. 2700
LogUart	Enables or Disables parts of the logging to save energy	bool	true = enabled, false = disabled
ObisCode	Comma separated list of ObisCodes to select a subset of the available information	string	e.g. 1-0:1.7.255*255 = Leistung (Momentan)
PayloadFormat	Format used for data upload (include timestamps or not)	int	1=no timestamp, 2=include timestamp
uploadCron	Cron expression defining when to read and upload	string	0 0/15 * * * * for every 15 minutes
TestMode	Enables Testmode to check compatibility	bool	true = enabled, false = disabled

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/repetitions	description
red/green/blue	500 ms each	initial pattern after reset
green	300 ms	data message received
red	300 ms	no incoming data
green	300 ms	sml parser successful
red	300 ms	sml parser failed
blue	2 x 300 ms	LoRaWAN network join successful
blue	300 ms	sent LoRaWAN data uplink
off	-	low-power mode until next sendout cycle

## Payload

### Payload Format Status Packet (Port 64)

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

The "Flags" field represents different error conditions by setting the individual bits:

Bit	Error Condition
0 (LSB)	Parsing the obis code failed
1	No incoming data from the opto head

2	Brown Out
3	None of the defined Obis Codes could be found in the incoming data
4-7	Unused

## 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.

# Test Mode

If TestMode is enabled in the configuration the device will not enter normal operation but check for different possible meter output protocols. Received data will be shown in the Log and parsed if possible. After completing the detection the result will be presented 10 times as blinking pattern via LED. After that the testing cycle will start again. The Log-Output can also be used to identify available OBIS Codes.

blinking pattern	description
1 x green	Ascii output without activation
2 x green	SML
3 x green	IEC 62056-21
1x red	no data received / data could not be parsed



## IEC 62056-21 not yet supported

IEC 62056-21 protocol can be detected in testmode but is not yet supported in normal mode.

# 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

<b>Product</b>	
Type name	LOB-S-CUR-LW-EDL21
Description	Electricity meter over LoRaWAN Bridge
<b>RF transceiver</b>	
Type	Semtech SX1261
Frequency	863 MHz to 870 MHz
Max. TX Power	max. +14 dBm
Typical RF Range	2km
Ideal RF Range	10km (free line of sight)
<b>LoRa communication</b>	
Protocol	Class A LoRaWAN 1.0.1 EU868
Activation method	Over-the-air-activation (OTAA) Activation by personalization (ABP)
Encryption	AES128
<b>Environmental Requirements</b>	
Operating temperature	-20°C – 55°C
Max installation height	2m
<b>Standards</b>	
 	

### Disposal / WEEE / Entsorgung

[Information about the disposal of the Device.](#)

### CE Declaration of Conformity

[CE Declaration of Conformity \(pdf\).](#)

