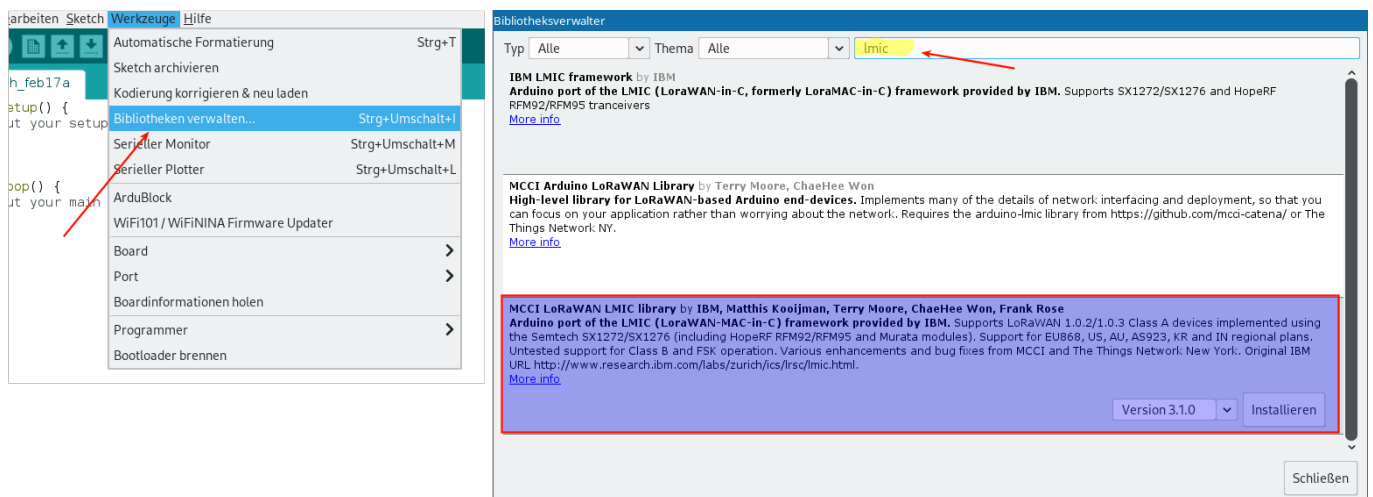


# Ein "Standardsensor" mit dem LoRaShield

## LoraShield aufstecken

## Bibliothek installieren

Um das Shield ansprechen zu können, muss die LMIC Bibliothek installiert werden:



## Sketch

Klicken, um den Beispielsketch zu sehen

hellolora.ino

```
// MIT License
//
// https://github.com/gonzalocasas/arduino-uno-dragino-lorawan/blob/master
// LICENSE
// Based on examples from
// https://github.com/matthijskooijman/arduino-lmic
// Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman

// Adaptions: Andreas Spiess

#include <lmic.h>
#include <hal/hal.h>

static const u1_t NWKSKEY[16] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00};
```

```
static const u1_t APPSKEY[16] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
static const u4_t DEVADDR = 0x00000000;

// These callbacks are only used in over-the-air activation, so they
// are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will
// complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 20;

// Pin mapping Dragino Shield
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
void onEvent (ev_t ev) {
    if (ev == EV_TXCOMPLETE) {
        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));
        // Schedule next transmission
        os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
    }
}

void do_send(osjob_t* j){
    // Payload to send (uplink)
    static uint8_t message[] = "hi";

    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare upstream data transmission at the next possible
        time.
        LMIC_setTxData2(1, message, sizeof(message)-1, 0);
        Serial.println(F("Sending uplink packet..."));
    }
}
```

```
    }  
    // Next TX is scheduled after TX_COMPLETE event.  
}  
  
void setup() {  
    Serial.begin(115200);  
    Serial.println(F("Starting..."));  
  
    // LMIC init  
    os_init();  
  
    // Reset the MAC state. Session and pending data transfers will be  
    discarded.  
    LMIC_reset();  
  
    // Set static session parameters.  
    LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);  
  
    // Disable link check validation  
    LMIC_setLinkCheckMode(0);  
  
    // TTN uses SF9 for its RX2 window.  
    LMIC.dn2Dr = DR_SF9;  
  
    // Set data rate and transmit power for uplink (note: txpow seems  
    to be ignored by the library)  
    LMIC_setDrTxpow(DR_SF12, 14);  
  
    // Start job  
    do_send(// MIT License  
//  
// https://github.com/gonzalocasas/arduino-uno-dragino-lorawan/blob/master  
// LICENSE  
// Based on examples from  
// https://github.com/matthijskooijman/arduino-lmic  
// Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman  
  
// Adaptions: Andreas Spiess  
  
#include <lmic.h>  
#include <hal/hal.h>  
// #include <credentials.h>  
  
#ifdef CREDENTIALS  
static const u1_t NWKSKEY[16] = NWKSKEY1;  
static const u1_t APPSKEY[16] = APPSKEY1;  
static const u4_t DEVADDR = DEVADDR1;  
#else  
static const u1_t NWKSKEY[16] =  
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};  
static const u1_t APPSKEY[16] =  
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};  
static const u4_t DEVADDR = 0x00000000;
```

```
static const u1_t APPSKEY[16] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
static const u4_t DEVADDR = 0x00000000;
#endif

// These callbacks are only used in over-the-air activation, so they
// are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will
// complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 20;

// Pin mapping Dragino Shield
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
void onEvent (ev_t ev) {
    if (ev == EV_TXCOMPLETE) {
        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));
        // Schedule next transmission
        os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
    }
}

void do_send(osjob_t* j){
    // Payload to send (uplink)
    static uint8_t message[] = "hi";

    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare upstream data transmission at the next possible
        time.
        LMIC_setTxData2(1, message, sizeof(message)-1, 0);
    }
}
```

```
        Serial.println(F("Sending uplink packet..."));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
    Serial.begin(115200);
    Serial.println(F("Starting..."));

    // LMIC init
    os_init();

    // Reset the MAC state. Session and pending data transfers will be
    discarded.
    LMIC_reset();

    // Set static session parameters.
    LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power for uplink (note: txpow seems
    to be ignored by the library)
    LMIC_setDrTxpow(DR_SF12,14);

    // Start job
    do_send(&sendjob);
}

void loop() {
    os_runloop_once();
}
&sendjob);
}

void loop() {
    os_runloop_once();
}
```

## Neues Device bei TTN erstellen

Nun benötigt man in der TTN Console eine Application und muss dort ein neues Device anlegen. Man vergibt einen Namen und lässt die übrigen Werte berechnen. Das Ergebnis sieht so aus, wie im Screenshot zu sehen.

## Wichtig:

- Das Device muss als Activation Method ABP haben
- Um die passenden IDs im Sketch einzutragen, müssen diese das passende Format haben - die Darstellung kann man durch anklicken von <> umschalten.

Applications > qg\_sensoren > Devices > beispielsensor

Overview Data Settings

### DEVICE OVERVIEW

Application ID qg\_sensoren

Device ID beispielsensor

Activation Method ABP

Device EUI <> msb { 0x00, , 0x5F, 0x80 }

Application EUI <> 70 B3 D5 7E D0 02 A7 D8

Device Address <> 26 01 1B 7F

Network Session Key <> msb { 0x7E, , .6, 0x1D, 0xCF, }

App Session Key <> .....

Status ● never seen

Frames up 0 [reset frame counters](#)

Frames down 0

## IDs in den Sketch eintragen

- NWKSKEY: Network Session Key
- APPSKEY: Application Session Key
- DEVADDR: Device Address

```
static const u1_t NWKSKEY[16] = { 0x7E, , , , , , , , , , , , , , , , 0xD3, 0x16, 0x1D, 0xCF, 0xA2, 0x54, 0xD6, 0x7E };
static const u1_t APPSKEY[16] = { 0x3A, 0x5A, 0x30, 0xDC, 0x43, , , , , , , , , , , , 0xFB, 0x1A, 0x70, 0x5A };
static const u4_t DEVADDR = 0x237F;
```

From:

<https://wiki.qg-moessingen.de/> - **QG Wiki**

Permanent link:

<https://wiki.qg-moessingen.de/faecher:nwt:lorawan:uebersicht:lorashield:start?rev=1581965417>



Last update: **17.02.2020 19:50**