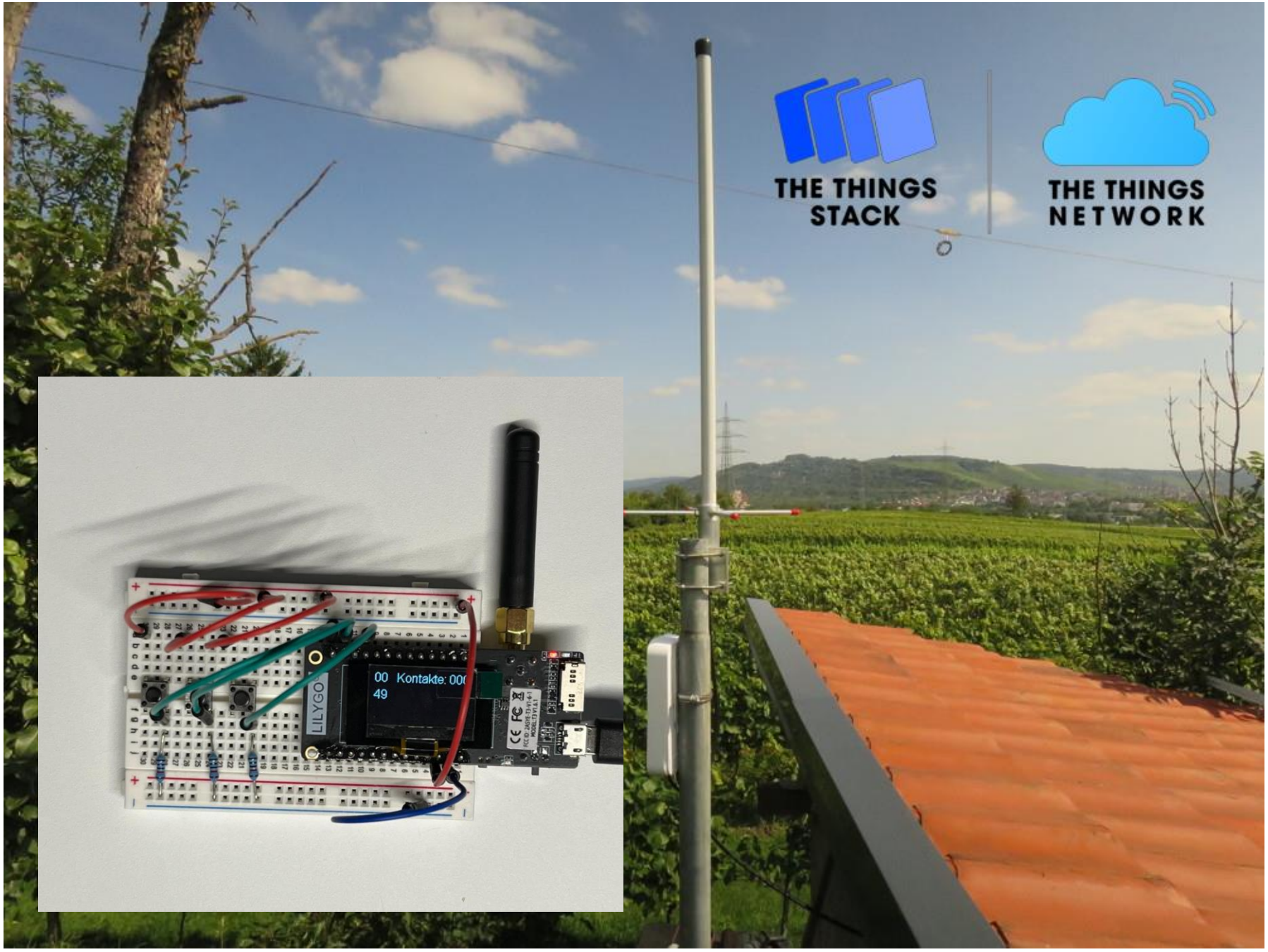
HamGroup

LoRa / LoRaWAN / IoT

Internet der Dinge

Online-Workshop

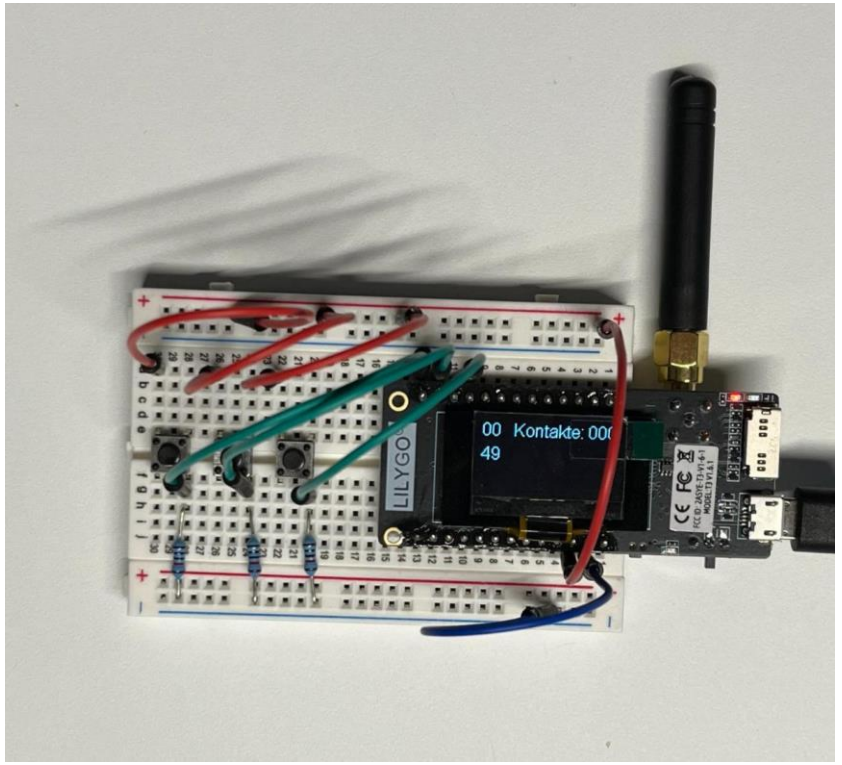
Taster-Node – 3-fach



LORAWAN-WORKSHOP

» Ziel:

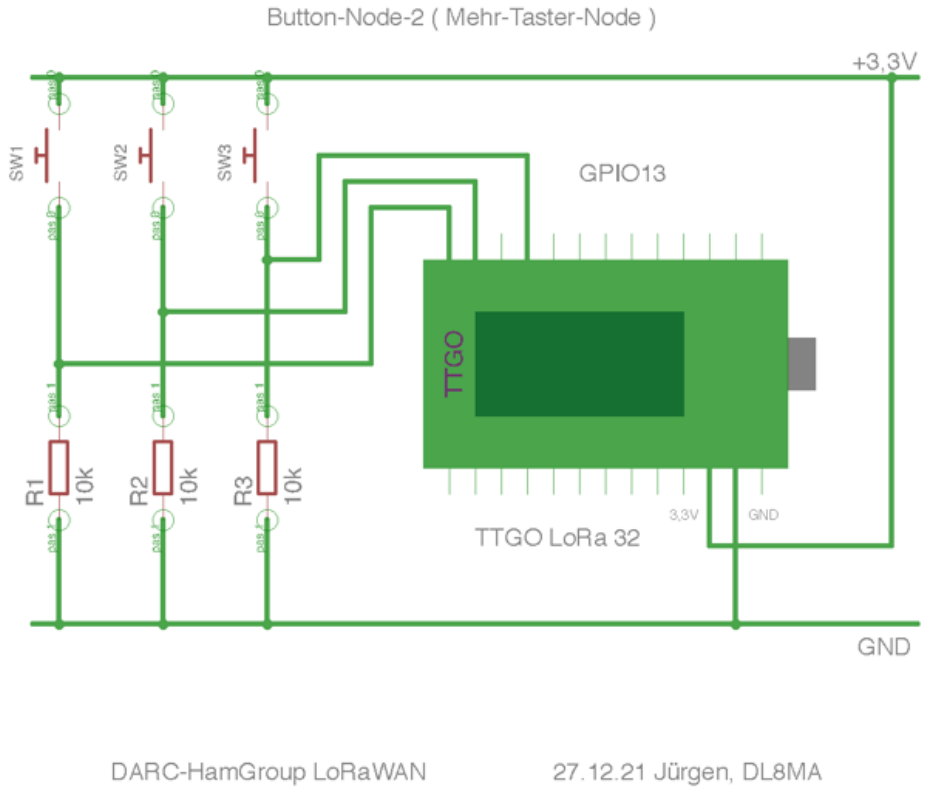
» Tasten-Node programmieren der Tastendrucke
über LoRaWAN zu einem Applikationsserver schickt



LORAWAN-WORKSHOP

Drei Taster wird an die GPIO-Pin 34, 36 und 39 angeschlossen.

Je ein 10k-Widerstand dient als Pulldown-Widerstand der dafür sorgt das bei geöffnetem Taster der Eingangs-Pin auf definiertem Pegel hängt.



LORAWAN-WORKSHOP

- **Taster-Node**
- **Quellcode:**
  - herunterladen
  - entpacken
  - Keys eintragen
  - compilieren
  - hochladen

taster-node2.ino | keys.h

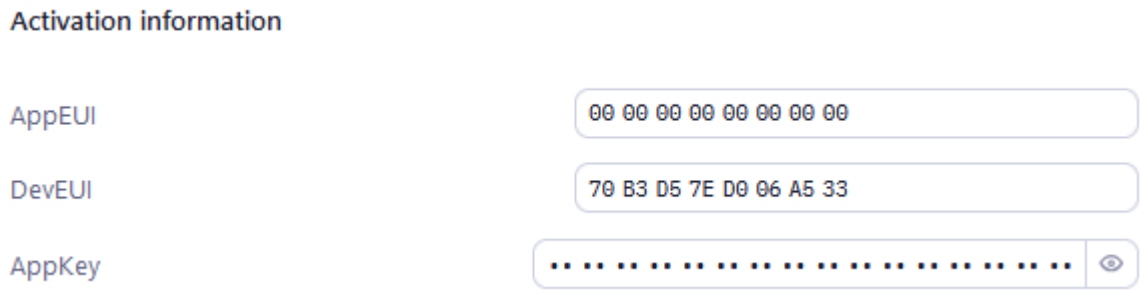
```
/*************************************
  DARC-HamGroup LoRaWAN

  Button-Node-1

  ein Taster bzw. Schalter löst bei Betätigung eine LoRaWAN-Aussendung aus
  nach der letzten Aussendung wird nach einer einstellbaren Zeit zyklisch eine Statusmeldung ausgesendet

  Based on the LoRaWAN example from Thomas Telkamp and Matthijs

  20.12.2021  Jürgen, DL8MA
  02.10.2024  Jürgen, DL8MA - 2. Workshop

  http://www.p37.de/LoRaWAN

  passender Payload-Formatter befindet sich ganz unten in einem Kommentarfeld

*/
#include <lmic.h>                           // LMIC library from the libary-manager
#include <hal/hal.h>                        // includet in the lmic library
#include <SPI.h>
#include "SSD1306Wire.h"                    // ThingPulse OLED SSD1306 - Library  https://github.com/ThingPulse/esp8266-oled-ssd1306
#include "keys.h"

SSD1306Wire display(0x3c, SDA, SCL);        // Display-Anschlüsse sind schon über die Board-Auswahl definiert
```

https://amateurfunk.de/lora/vortraege/241023/taster-node2.zip

- **Taster-Node**

  - Für die Anmeldung ("Join") des Nodes müssen im Quellcode noch einige Keys und EUIs angepasst werden:

    - **JoinEUI** 64-bit globally unique identifier (war: AppEUI)
    - **DevEUI** 64 bit end-device identifier
    - **AppKEY** 128 bit Applikationsschlüssel

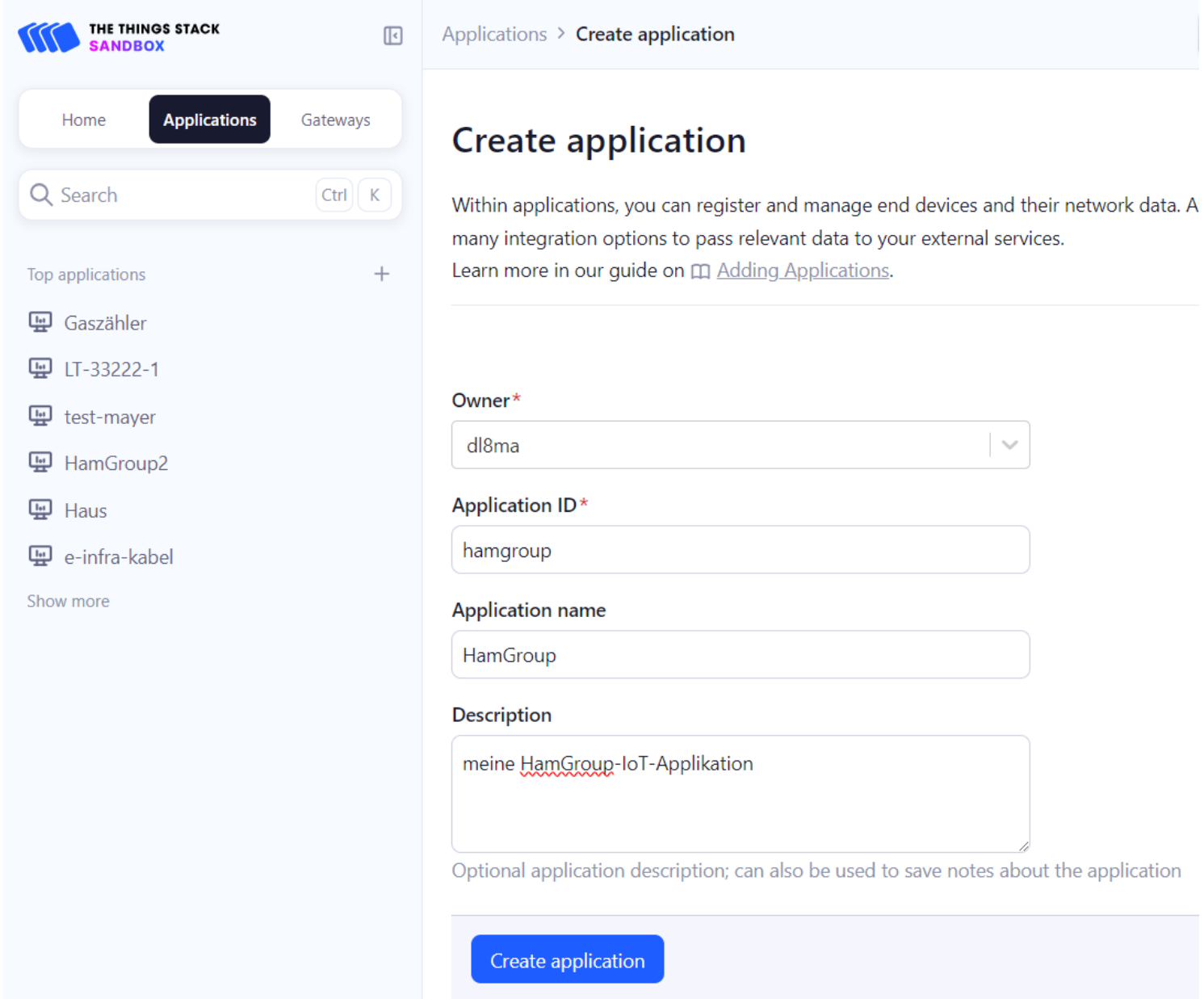


  - Die DEVEUI und der APPKEY wird beim Anlegen des Nodes auf dem LoRaWAN-Server dort generiert. Diese müssen dann in den Quellcode reinkopiert werden.

  - Dazu benötigt man einen Account im The Things Network.

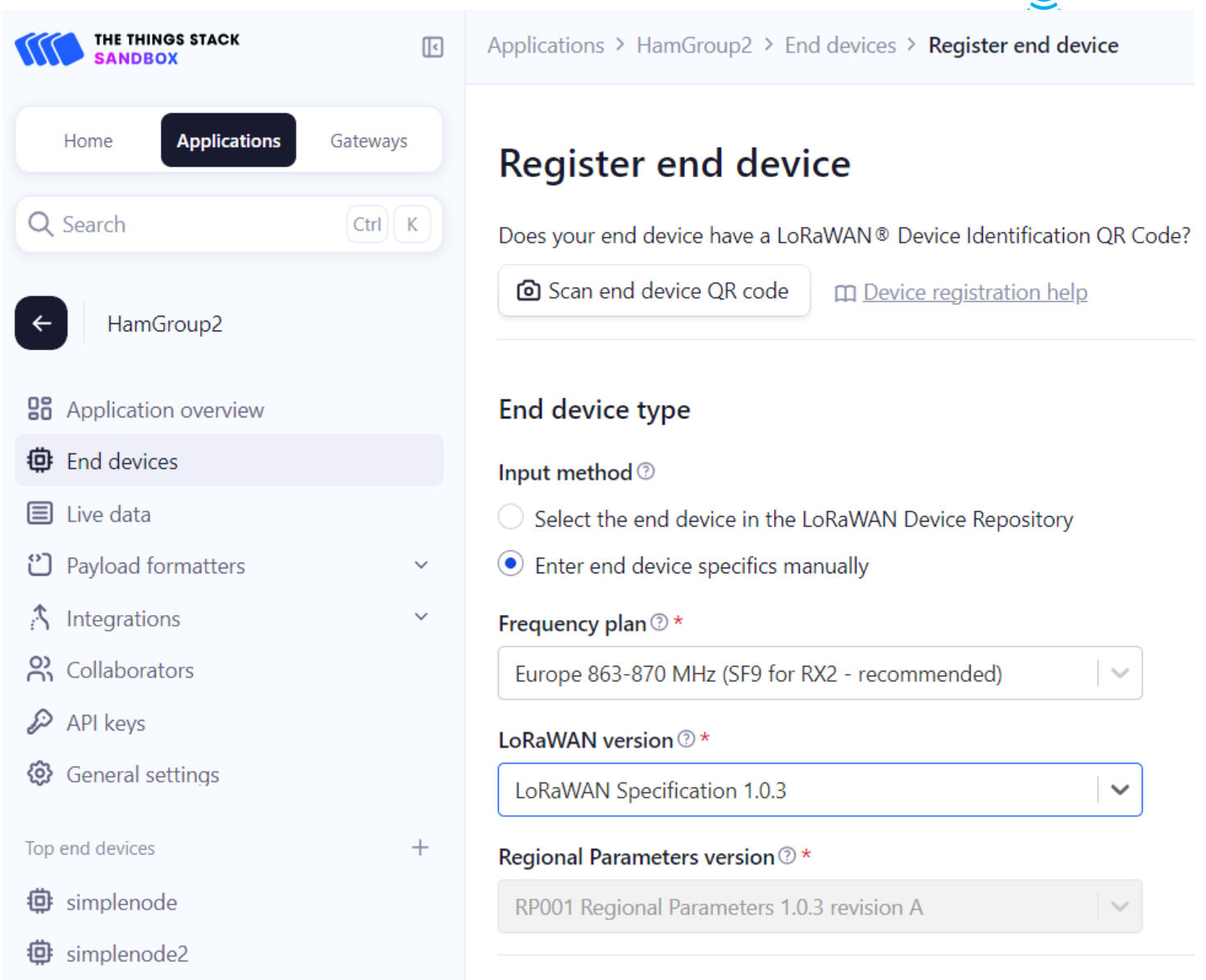
LORAWAN-WORKSHOP

- **Taster-Node**
- **Node auf TTN einrichten:**
  - 1. Applikation anlegen (falls notwendig)
    - Home => Applications => Add application
    - Ausfüllen
    - Create Application



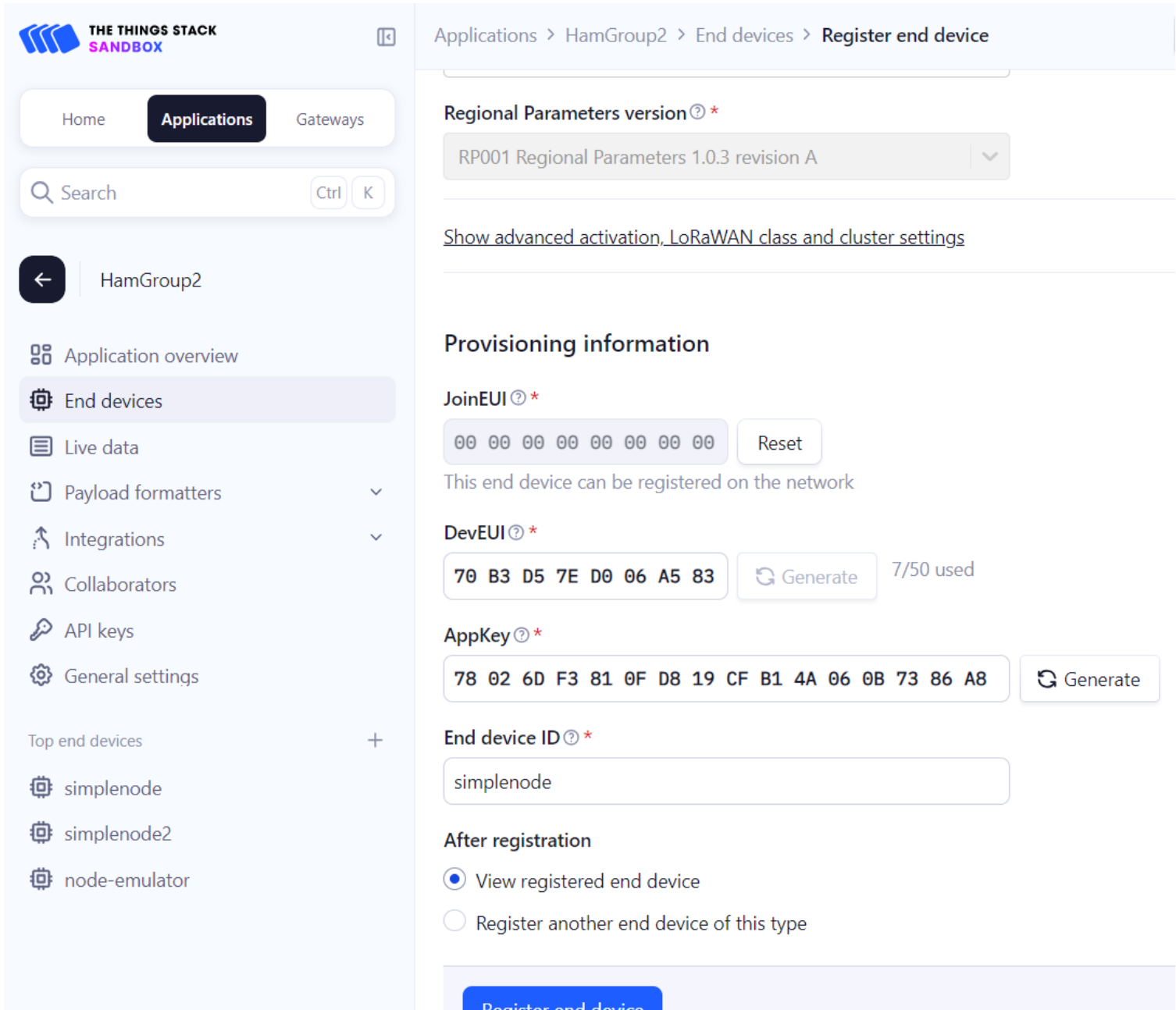
LORAWAN-WORKSHOP

- **Taster-Node**
- **Node auf TTN einrichten:**
  - 2. Device (Node) anlegen
    - Applications => HamGroup2 => End devices
    - => Register end device
    - ausfüllen:
      - „manually“
      - Europe 863-870 Mhz
      - LoRaWAN Specification 1.0.3
    - …



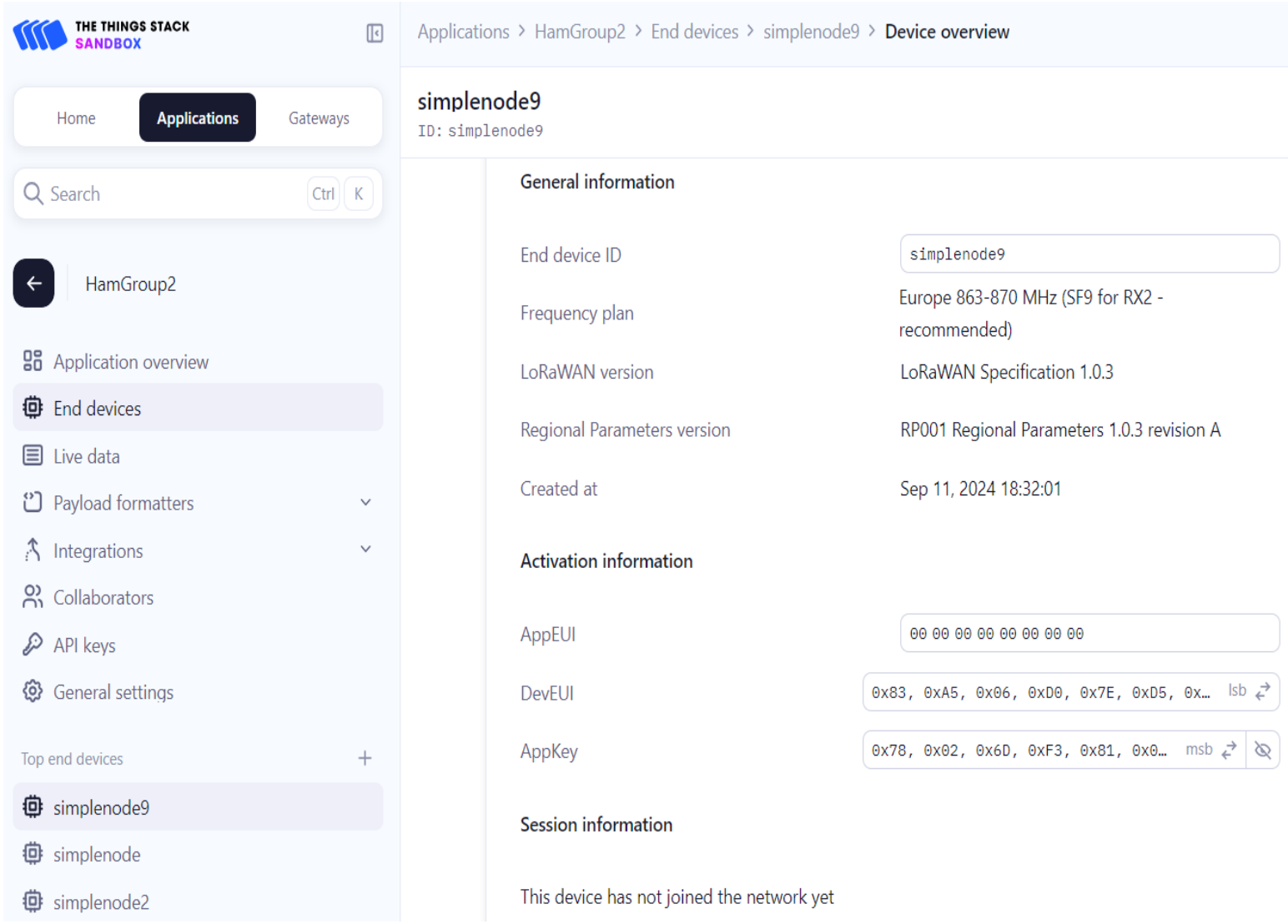
LORAWAN-WORKSHOP

- **Taster-Node**
- **Node auf TTN einrichten:**
  - 2. Device (Node) anlegen
    - JoinEUI: „00 00 00 00 00 00 00 00“
    - DevEUI: generieren
    - AppKey: generieren
    - End device ID: taster-node-1
    - => Register end device



LORAWAN-WORKSHOP

- **Taster-Node**
- **Node auf TTN einrichten:**
  - Activation information
  - AppEUI 0000000000000000
  - DevEUI 70B3D57ED006A583
  - AppKey 78026DF3810FD819CFB14A060B7386A8



LORAWAN-WORKSHOP

- **Taster-Node**
- **Keys übertragen:**
  - Falls die Einträge ausgeblendet sind kann man die durch den Button 1 sichtbar machen.
  - Mit den Buttons 2 schaltet man die Hex-Darstellung ("0x##") ein.
  - Der Quellcode mit der LMIC-Lib erwartet (leider) für die DevEUI und den AppKey unterschiedliche Reichenfolgen der Hex-Zahlen:
  - DevEUI: lsb "little endian"
  - AppKey: msb "big endian„
  - Die jeweilige Reihenfolge (lsb oder msb) stellt man mit dem Button 3 bzw. 4 um.
  - Bei der DevEUI muss im TTN-Netz die 0x70 ganz rechts stehen: 0x7F, 0x9D, 0x04, 0xD0, 0x7E, 0xD5, 0xB3, 0x70
  - Nun kann man mit dem Kopierbutton (jeweils ganz rechts) den Key bzw. die EUI in die Zwischenablage kopieren.

- **Taster-Node**
- **Keys übertragen in keys.h:**

```
// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.

static const u1_t PROGMEM APPEUI[8]={ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]={ 0x33, 0xA5, 0x06, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
static const u1_t PROGMEM APPKEY[16] = { 0xEE, 0xBE, 0x16, 0xC1, 0xA4, 0xD1, 0xAD, 0x4F, 0x71, 0xF8, 0xA4, 0x3D, 0x9F, 0xED, 0x29, 0x9B };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
```

» **Taster-Node**

- **Taster-Node**
- **Payload-Formater:**

- Es wird ein Byte übertragen. Darin sind zwei
- Informationen in den zwei Nibbles gespeichert:

- 1. Nibble: 3 Taster-Kontakte
- jeweils
  - 0 = offen
  - 1 = geschlossen

- 2. Nibble: Trigger-Info 0 oder 1

» Taster-Node

In diesem Beispiel sind drei Kontakte angeschlossen

Die Kontaktzustände werden im 1. Nibble binär übertragen:

0xxx 000y

x = Kontakt 1 - 3

Beispiel:

0x51 => 01010001

=> SW1 geschlossen

=> SW2 offen

=> SW3 geschlossen

=> Trigger-Info = 1 (= mind. ein Kontakt wurde geschlossen oder geöffnet)

- Wir haben zwei einstellige Zahlen:
  - Tasten jeweils 0 oder 1
  - Trigger 0 oder 1
  - Die müssen jetzt in ein Byte gepackt werden
  - => 0xxx000y
    - => xxx = Tasten 1 – 3
    - => y = Trigger

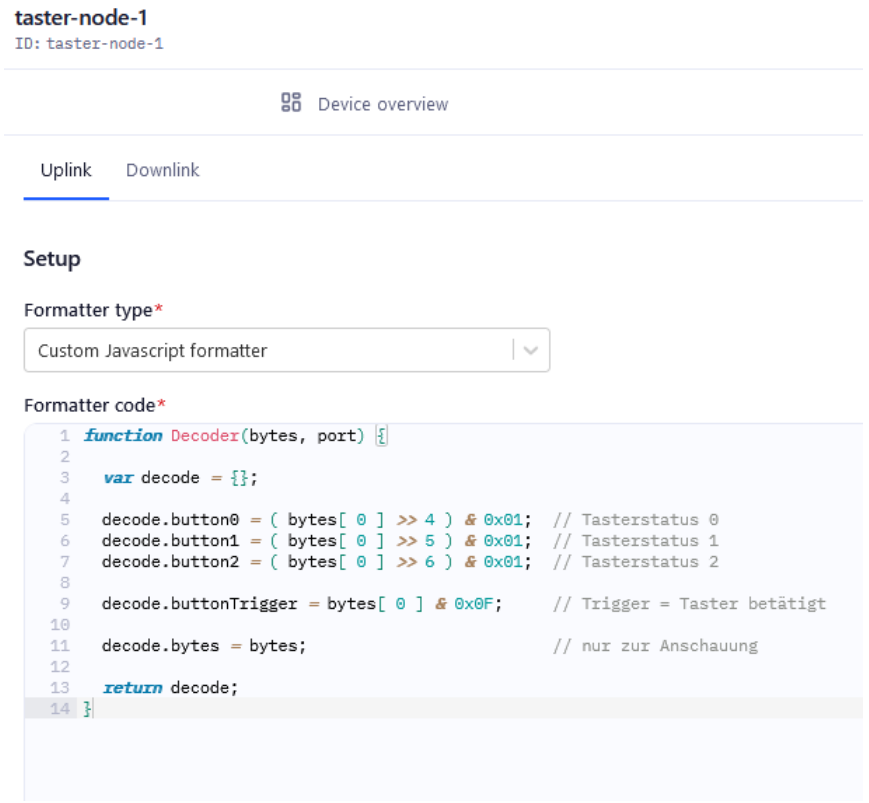
```
mydata[ 0 ] = 0;                                                    // 1 Byte ( = 2 Nibble )
for( i = 0; i < kontaktAnzahl; i++ ) {
  mydata[ 0 ] = mydata[ 0 ] | buttonState[ i ] << ( 4 + i );        // 1. Nibble des Bytes: aus den Kontakte zusammenbauen
}
mydata[ 0 ] = mydata[ 0 ] | 0x01;                                   // 2. Nibble des Bytes: Trigger-Info = 1 ( Kontakt geändert )
```

LORAWAN-WORKSHOP

- **Taster-Node**
- **Payload-Dekoder:**

```
function Decoder(bytes, port) {
  var decode = {};

  decode.button0 = ( bytes[ 0 ] >> 4 ) & 0x01;  // Tasterstatus 0
  decode.button1 = ( bytes[ 0 ] >> 5 ) & 0x01;  // Tasterstatus 1
  decode.button2 = ( bytes[ 0 ] >> 6 ) & 0x01;  // Tasterstatus 2

  decode.buttonTrigger = bytes[ 0 ] & 0x0F;     // Trigger = Taster betätigt

  decode.bytes = bytes;                         // nur zur Anschauung

  return decode;
}
```

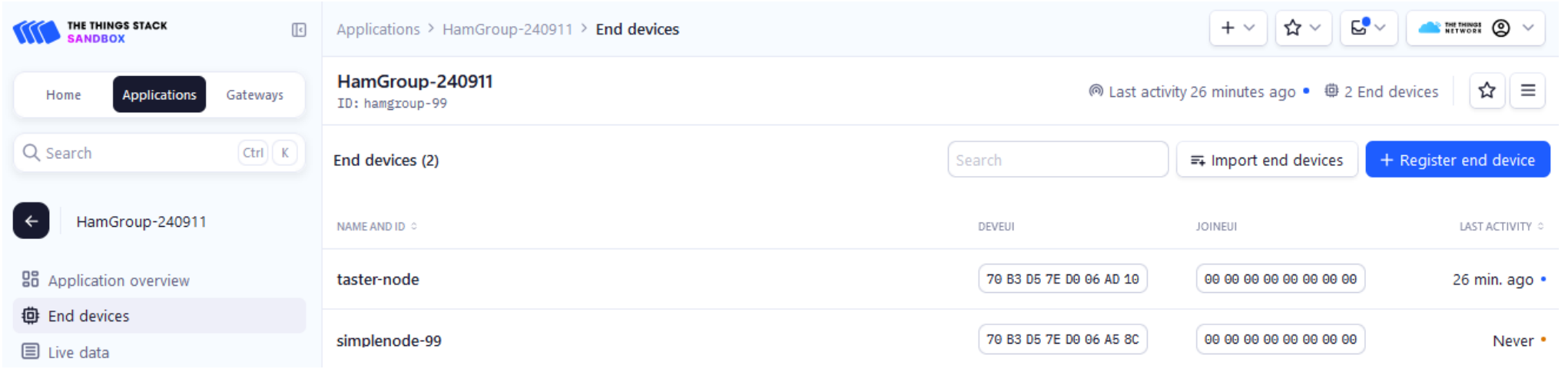
=> eintragen!



- **Taster-Node**
- **Funktionstest:**

Hier:

2. Eintrag läuft



» Taster-Node

„Live Data“

taster-node-1
ID: taster-node-1

Device overview | Live data | Messaging

TIME	TYPE	DATA PREVIEW
19:52:10	Forward uplink data message	DevAddr: 26 0B E4 12 Payload: { button0: 0, button1: 1, button2: 0, buttonTrigger: 1, bytes: [33] } 21 FPort: 1 Data rate: SF7BW125 SNR: 9.5 RSSI: -72
19:52:10	Successfully processed data message	DevAddr: 26 0B E4 12
19:51:36	Forward uplink data message	DevAddr: 26 0B E4 12 Payload: { button0: 0, button1: 0, button2: 0, buttonTrigger: 0, bytes: [0] } 00 FPort: 1 Data rate: SF7BW125 SNR: 9.25 RSSI: -72
19:51:36	Successfully processed data message	DevAddr: 26 0B E4 12

LORAWAN-WORKSHOP

» **Detailseite**

02.10.2024 18:44:53

---

entschlüsselte Payload:

0x**10**

---

dekodierte Payload:

```
stdClass Object
(
    [button] => 1
    [buttonTrigger] => 0
    [bytes] => Array
        (
            [0] => 16
        )

)
```

Datenrate:

Speading Faktor	SF 7
Brandbreite	125 kHz
QRG	867.3 MHz
Sendezeit:	0.046336s

Device-EUI: taster-node

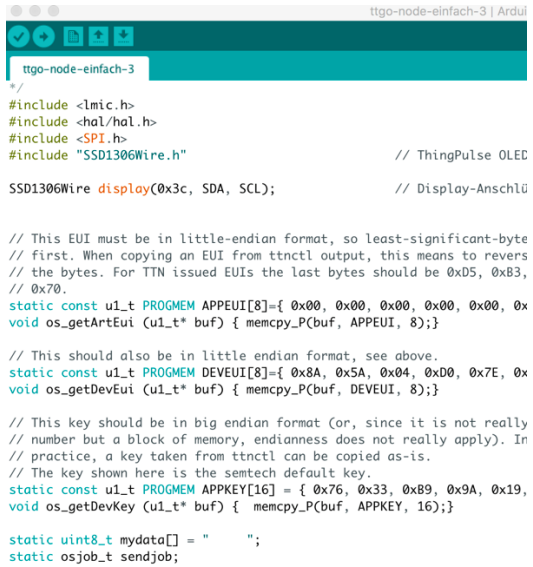
Gateways:

Gateway-Id	RSSI	SNR	Locator		
einfragmbh	-90	13.5	48.706603532361	9.0315863396137	X

Mitmachen:

- auf der Mailingliste anmelden

- Materialliste folgt noch auf **www.amateurfunk.de/lora**



```
ttgo-node-einfach-3 | Ardui
ttgo-node-einfach-3
*/
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include "SSD1306Wire.h"                    // ThingPulse OLED

SSD1306Wire display(0x3c, SDA, SCL);        // Display-Anschlü


// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to revers
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8]={ 0x00, 0x00, 0x00, 0x00, 0x00, 0x
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]={ 0x8A, 0x5A, 0x04, 0xD0, 0x7E, 0x
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
static const u1_t PROGMEM APPKEY[16] = { 0x76, 0x33, 0xB9, 0x9A, 0x19,
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

static uint8_t mydata[] = "     ";
static osjob_t sendjob;
```

Vielen Dank für Eure Aufmerksamkeit!

73 de Jürgen, DL8MA

Für Fragen stehe ich gerne zur Verfügung.

dl8ma@amateurfunk.de