

Einstieg in LoRaWAN mit dem Wio-E5 und Python

In diesem Workshop geben wir eine kurze Einführung in LoRaWAN und die Verwendung zusammen mit einem Seedstudio Wio E5 und einem Raspberry Pi. Der verwendete Python Code sollte mit jedem Raspberry Pi funktionieren, wir verwenden für die Workshops der Einfachheit halber, einen Pi 400 oder Pi 500.

Was ist LoRaWAN?

The Things Network

The Things Network (TTN) versteht sich als globales, kollaboratives Ökosystem, das die [LoRaWAN-Technologie](#) durch einen offenen Community-Ansatz demokratisiert. Die besondere Bedeutung für die LoRaWAN-Community ergibt sich aus dem kostenlosen Zugang zu einer robusten Netzwerk-Infrastruktur, die es Bastlern und Profis gleichermaßen erlaubt, Sensordaten über große Distanzen zu übertragen, ohne auf teure kommerzielle Anbieter angewiesen zu sein. Die Integration erfolgt dabei flexibel über den [TTN-Stack](#), der Daten via MQTT oder Webhooks an Endanwendungen weiterreicht und durch integrierte Payload-Formatierer die Interpretation binärer Datenströme vereinfacht. Da TTN auf dem Crowdsourcing-Prinzip basiert – jeder kann ein Gateway beisteuern und so die Abdeckung erweitern – wächst das Netzwerk organisch durch seine [weltweiten Communities](#) und schafft so eine flächendeckende Konnektivität, die auf gegenseitiger Unterstützung und offenen Standards fußt.

Hier geht es zur Registrierung: [Jetzt bei The Things Network registrieren](#) Registeriere dich, und lege deine erste Application an, hier fügen wir dann später unseren Wio-E5 ein.

Raspberry Pi und E5 verbinden

Verbinde den Raspberry Pi und den E5 über die Pins des Raspberry Pi's wie folgt:

- 3.3V → Rotes Kabel
- Ground → Schwarzes Kabel
- UART0 Tx (GPIO 14) → Weißes Kabel
- UART0 Rx (GPIO 15) → Gelbes Kabel

[Suche gerne nach einem sogenannten Pinout im Internet zu dem von dir verwendeten Pi damit du die Kabel richtig anschließen kannst]

Die Kabelfarben stimmen mit den offiziellen Grove Kabel Farben überein. 3.3V und Ground sind für die Spannungsversorgung zuständig. Tx und Rx sind Teil des von dem Wio-E5 Modul verwendeten Serial Protokolls. Tx steht für Transmit und Rx für Receive. Hier ist Obacht geboten! Dies ist dann keine 1:1 Verbindung sondern diese müssen Überkreuz angeschlossen werden. Rx vom Modul geht zum Tx vom Raspberry Pi und so auch das Tx vom Modul zum Rx vom Pi.

Um die serielle Schnittstelle auf dem Raspberry Pi zu aktivieren, nutzt man den Befehl `sudo raspi-config` welchen man im Terminal eingibt (öffnen kann man dies über den schwarzen Kasten in der oberen Leiste). Navigiere dort zu **Interface Options** → **Serial Port** (Navigation über Pfeiltasten und Enter). Deaktiviere die **Login shell** (No) und aktiviere die **Serial port hardware** (Yes). Ein anschließender Neustart mit `sudo reboot` ist erforderlich.

Das Terminal werden wir auch im weiteren Teil des Workshops verwenden.

Erste manuelle Tests

Falls du das LoRa-Modul ohne Python-Skript direkt über die vom Modul verwendeten AT-Befehle testen möchtest, kannst du das Terminalprogramm **Minicom** verwenden. Dies ist besonders nützlich, um die RX/TX-Crossover-Funktionalität zu prüfen oder die Hardware-Verbindung zu verifizieren.

1. Installation

Zuerst muss Minicom auf dem Raspberry Pi installiert werden:

```
sudo apt install minicom
```

2. Minicom starten

Starte die Kommunikation über die serielle Schnittstelle mit einer Baudrate von 9600:

```
sudo minicom -b 9600 -D /dev/serial0
```

Hinweis: Der Pfad /dev/serial0 verweist auf die primäre UART-Schnittstelle des Pi.

3. Konfiguration im Terminal

Da Minicom standardmäßig deine Eingaben nicht anzeigt, musst du das lokale Echo aktivieren. Die Steuerung erfolgt über **STRG+A**, gefolgt von einer weiteren Taste:

Befehl	Funktion
STRG+A ⇒ E	Echo aktivieren: Damit siehst du, was du tippst.
STRG+J	Eingabe bestätigen: Sendet den Befehl an das Modul.
STRG+A ⇒ X	Beenden: Minicom verlassen und zum Terminal zurückkehren.
STRG+A ⇒ C	Terminalfenster aufräumen (Clear).

4. Wichtige AT-Testbefehle

Sobald die Verbindung steht, kannst du das Modul mit folgenden Befehlen direkt steuern[cite: 65]:

AT-Befehl	Beschreibung
AT	Testet die RX/TX Kommunikation. Das Modul sollte mit "OK" antworten[cite: 65].
AT+ID	Zeigt die Geräte-IDs an (DevEui, DevAddr, JoinEui), die für TTN benötigt werden[cite: 65].
AT+MODE	Zeigt den aktuellen Modus des Moduls an[cite: 65].
AT+JOIN	Sendet eine Beitrittsanfrage an das LoRaWAN-Netzwerk[cite: 65].
AT+MSG="Text"	Sendet eine einfache Textnachricht als ASCII-Zeichen[cite: 65].

Wichtig: Befehle zur Kommunikation (wie das Senden von Nachrichten) funktionieren erst nach einer erfolgreichen Netzwerk-Anfrage (Join)[cite: 10].

Tests mit Python (Beispiel Skript)

Für eine komfortable Steuerung des LoRa-Moduls steht das Skript `LoRaWorkshop.py` zur Verfügung. Dieses automatisiert die AT-Befehle und bietet eine menügeführte Oberfläche im Terminal.

Vorbereitung

Bevor das Skript gestartet werden kann, muss die benötigte Python-Bibliothek installiert werden:

```
pip install pyserial
```

Hier ist außerdem das Skript. Referenziere es gerne um bestimmte Funktionen zu verstehen:

[LoRaWorkshop.py](#)

```
import serial
import time

PORT = "/dev/serial0"
BAUD = 9600

def send_command(ser, cmd, wait_for="OK", timeout=5):
    """Sendet einen Befehl und wartet auf eine Antwort."""
    ser.write(f"{cmd}\r\n".encode()) # wichtig ist \r\n zur bestätigung
    im Emulator
    start_time = time.time()
    response = ""

    while (time.time() - start_time) < timeout:
        if ser.in_waiting > 0:
            line = ser.readline().decode(errors='ignore').strip()
            if line:
                print(f" [Modul]: {line}")
                response += line
                if wait_for in line:
                    return True

    return False
```

```

def main_menu():
    try:
        ser = serial.Serial(PORT, BAUD, timeout=1)
        print("--- LoRa-E5 Schnellwahl-Terminal ---")

        while True:
            print("\nWas küt ?")
            print("1 Test (AT)")
            print("2 IDs auslesen (AT+ID)")
            print("3 Mit TTN verbinden (AT+JOIN)")
            print("4 Nachricht senden (Text)")
            print("5 Nachricht senden (Hex)")
            print("6 Reset des Moduls") # Falls timeout wegen zuviel
anfragen
            print("7 Nachricht im Loop senden (alle 45s)")
            print("q Beenden")
            # print("f Mit TTN verbindung Forcen (AT+JOIN=FORCE)")

            choice = input("\nAuswahl: ").strip().lower()

            if choice == '1':
                send_command(ser, "AT")

            elif choice == '2':
                send_command(ser, "AT+ID")

            elif choice == '3':
                print("Join-Prozess gestartet...")
                keywords = ["done", "joined already"]
                ser.write(b"AT+JOIN\r\n")
                start_time = time.time()
                joined = False

                while (time.time() - start_time) < 30:
                    if ser.in_waiting > 0:
                        line =
ser.readline().decode(errors='ignore').strip()
                        if line:
                            print(f" [Modul]: {line}")
                            line_lower = line.lower()
                            if any(key in line_lower for key in
keywords):
                                joined = True
                                break
                            if "failed" in line_lower or "busy" in
line_lower:
                                break

                if joined:
                    print("Bereit: Modul ist im Netz.")

```

```

else:
    print("Timeout oder Fehler beim Join.")

elif choice == '4':
    msg = input("Gib deine Nachricht ein: ")
    send_command(ser, f'AT+MSG="{msg}"')

elif choice == '5':
    hex_payload = input("Gib den Hex-Code ein (z.B. AA BB
01): ")
    hex_payload = hex_payload.replace(" ", "") # falls
Eingabe Leerzeichen beinhaltet
    send_command(ser, f'AT+MSGHEX="{hex_payload}"')

elif choice == '6':
    send_command(ser, "AT+RESET")

elif choice == 'f':
    send_command(ser, "AT+JOIN=FORCE")

elif choice == '7':
    msg = input("Gib die Nachricht ein, die geloopt werden
soll: ")
    print("\nLoop gestartet. Drücke STRG+C (bzw. Ctrl+C),
um den Loop zu beenden!")
    try:
        while True:
            print(f"\nSende Nachricht: {msg}")
            send_command(ser, f'AT+MSG="{msg}"') # Auch mit
Sensordaten Möglich einfach msg ändern zu dem Sensor Output
            print("Warte 45 Sekunden auf die nächste
Sendung...")
            time.sleep(45) # Zeitintervall zur loop
übermittlung # Es gibt Gesetzliche Vorgaben
(5min) für den Abstand von Signalen # Wollen ja nicht die
Bundesnetzagentur im nacken haben
        except KeyboardInterrupt:
            print("\nLoop durch Benutzer abgebrochen. Kehre zum
Menü zurück...")

elif choice == 'q':
    print("Programm beendet.")
    ser.close()
    break
else:
    print("Ungültige Auswahl.")

time.sleep(1)

```

```
except serial.SerialException as e:
    print(f"Fehler: Konnte Port {PORT} nicht öffnen ({e})")

if __name__ == "__main__":
    main_menu()
```

Skript starten

Navigiere im Terminal in den Ordner, in dem die Datei liegt, und führe sie aus:

```
python LoRaWorkshop.py
```

Funktionsübersicht

Nach dem Start erscheint ein Auswahlmenü mit folgenden Optionen:

Auswahl	Funktion	Beschreibung
1	Test (AT)	Prüft, ob die RX/TX-Verbindung zum Modul steht.
2	IDs auslesen	Zeigt DevEui, DevAddr und AppEui für die TTN-Registrierung an.
3	Mit TTN verbinden	Startet den Join-Prozess (AT+JOIN). Ein Gateway muss erreichbar sein.
4	Nachricht senden (Text)	Übermittelt eine Nachricht im Klartext.
5	Nachricht senden (Hex)	Übermittelt Daten als Hexadezimal-Zeichen (ASCII-Code).
6	Reset	Startet das LoRa-Modul neu, falls es nicht reagiert.
7	Loop-Modus	Sendet automatisch alle 45 Sekunden eine Nachricht.
f	Verbindung erzwingen	Verlässt bestehende Sitzungen und erzwingt einen neuen Join (Force).
q	Beenden	Schließt das Skript (auch per STRG+C möglich).

Hinweis: Die Funktionen zur Datenübertragung (Text/Hex/Loop) stehen erst zur Verfügung, wenn die Netzwerkanfrage erfolgreich war und das Modul "Network joined" meldet.

Payload Decoder

Um die Daten, welche in TTN über LoRaWAN ankommen, zu decodieren kann ein sogenannter Payload Decoder hinterlegt werden. Da die Daten in diesem Fall über die ASCII Tabelle (Gerne mal googeln) codiert wurden in Hexadezimal, sieht der Decoder in diesem Fall so aus:

[decoder.js](#)

```
function decodeUplink(input) {
    var bytes = input.bytes;

    // Konvertierung der Bytes in einen String (ASCII)
    var text = "";
    for (var i = 0; i < bytes.length; i++) {
```

```
    text += String.fromCharCode(bytes[i]);
}

return {
  data: {
    type: "TEXT_MSG",
    message: text,
    length: bytes.length
  }
};
}
```

Credits

Die Entwicklung des Python Scripts sowie die Anleitung für die manuellen Tests wurden durch den Studenten Tim Oswald im Rahmen seiner Bachelorarbeit durchgeführt. Hier der Link zu seinem erstellten Cheatsheet:

Cheatsheet

From:
<https://wiki.eolab.de/> - **HSRW EOLab Wiki**

Permanent link:
https://wiki.eolab.de/doku.php?id=lets_plaiy:lorawan:start&rev=1777303400

Last update: **2026/04/27 17:23**

