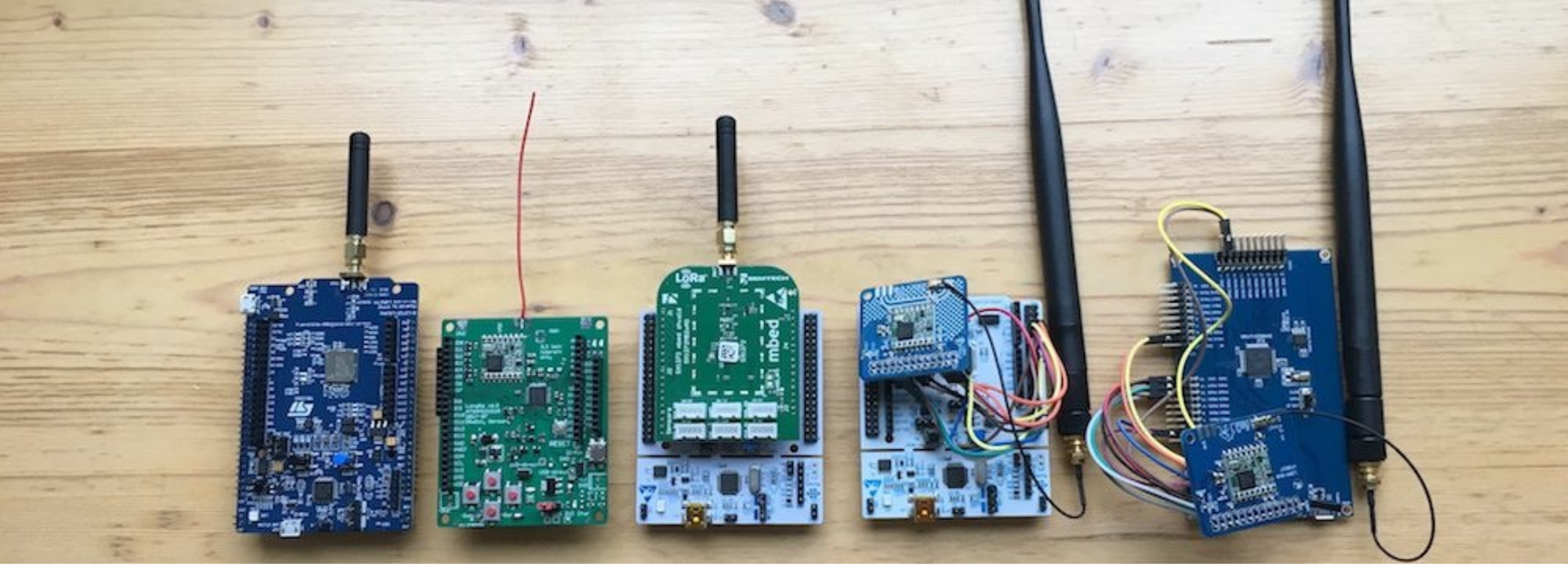
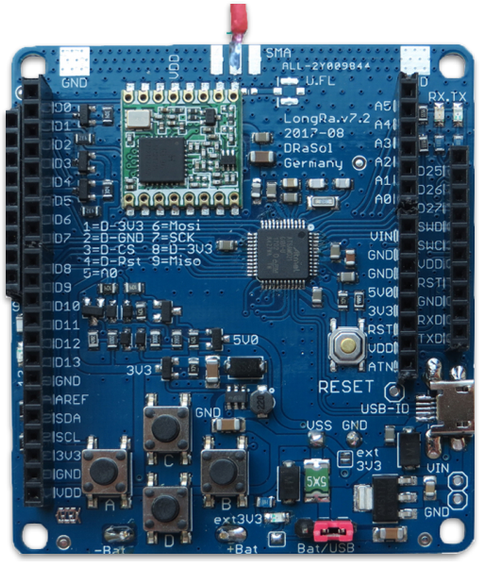


Non-Volatile Settings

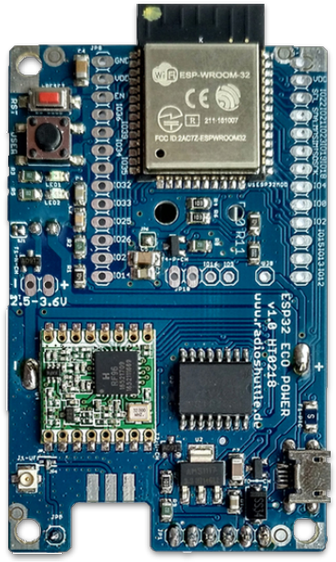
Permanent application settings for embedded devices
RadioShuttle NVProperty API
for Arduino and Mbed OS

Helmut Tschemernjak

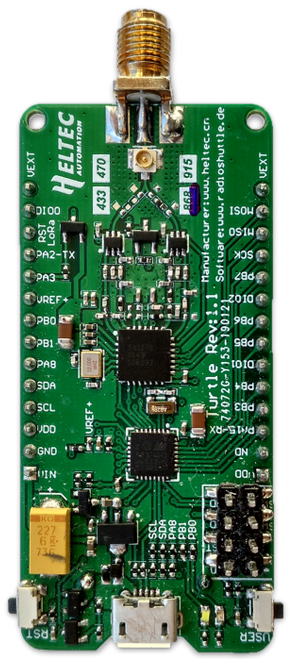
NVProperty Supported Boards



LongRa
aka
Arduino Zero D21



ECO Power
aka
Arduino ESP32

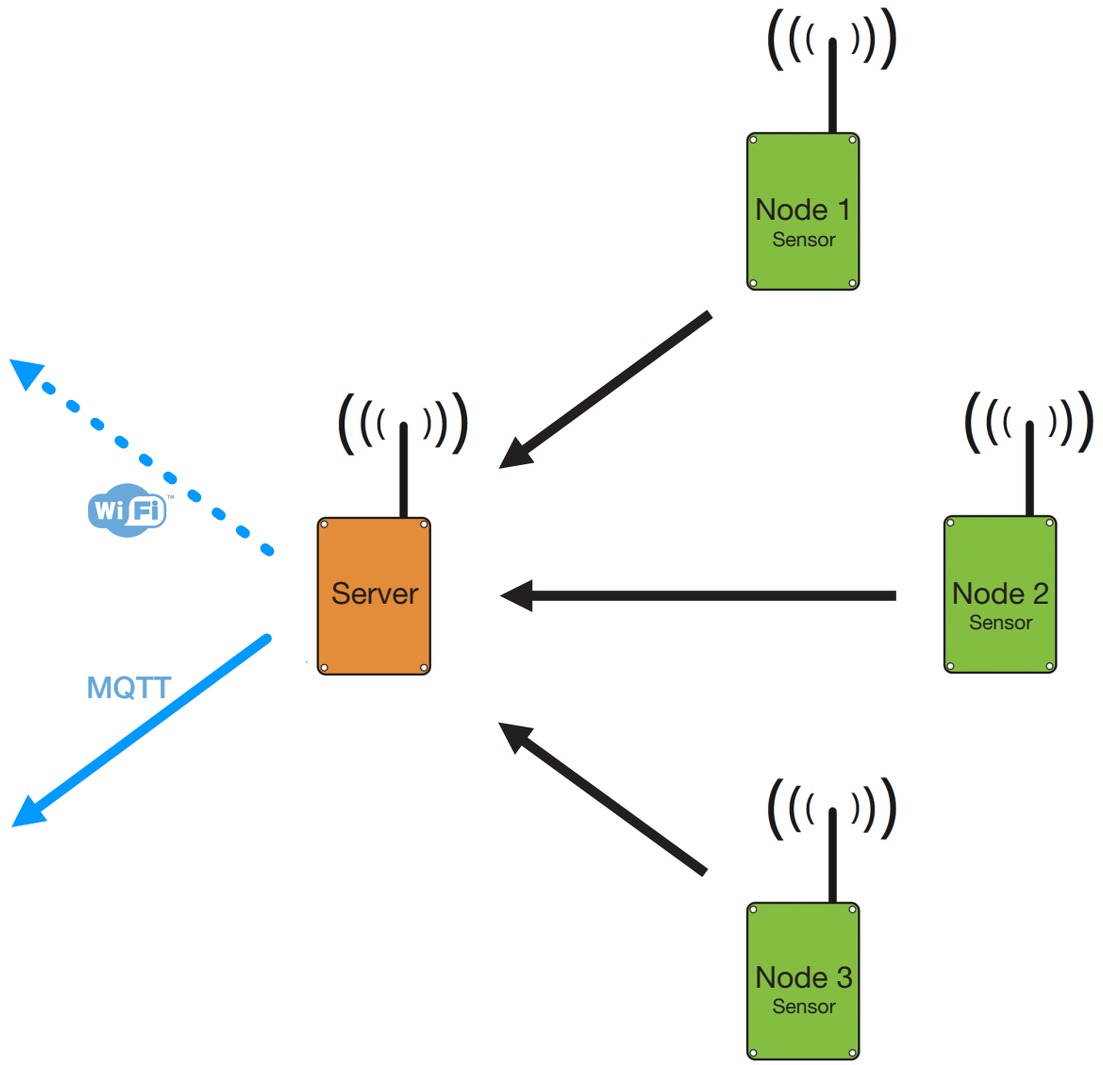


Turtle
aka
Mbed OS (STM32L4)

Custom Settings

Settings required per device

- WiFi SSID
- MQTT broker URL
- LoRa station ID
- Server password
- Channel / frequency
- Spreading factor
  (e.g. SF 7 or SF 11)
- …



Where To Store Settings?

Separate settings from the application code

- On Windows:

  Settings stored in the “Registry”

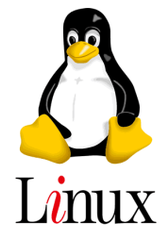
- On Linux / Mac / Windows:

  Settings stored by use of environment variables, e.g.

  USER=username

  PATH=/<path>/…

  TERM=vt100

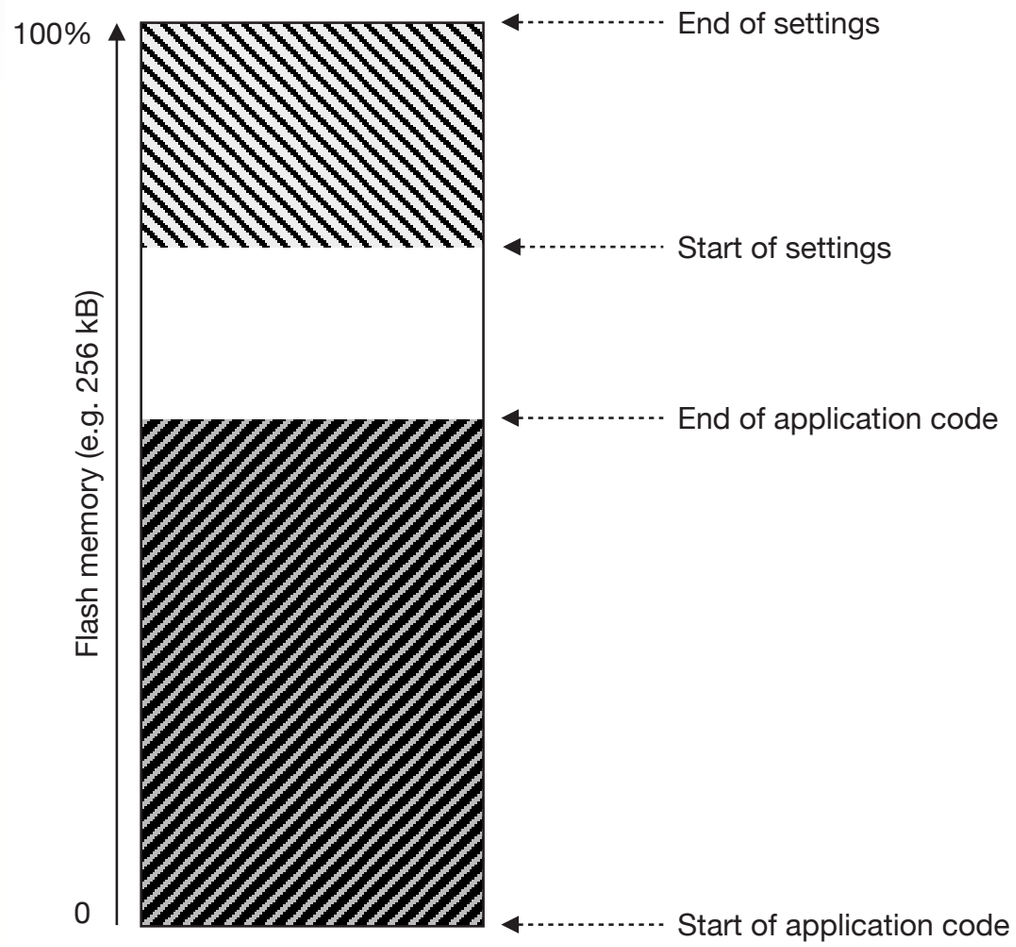


- On embedded devices:

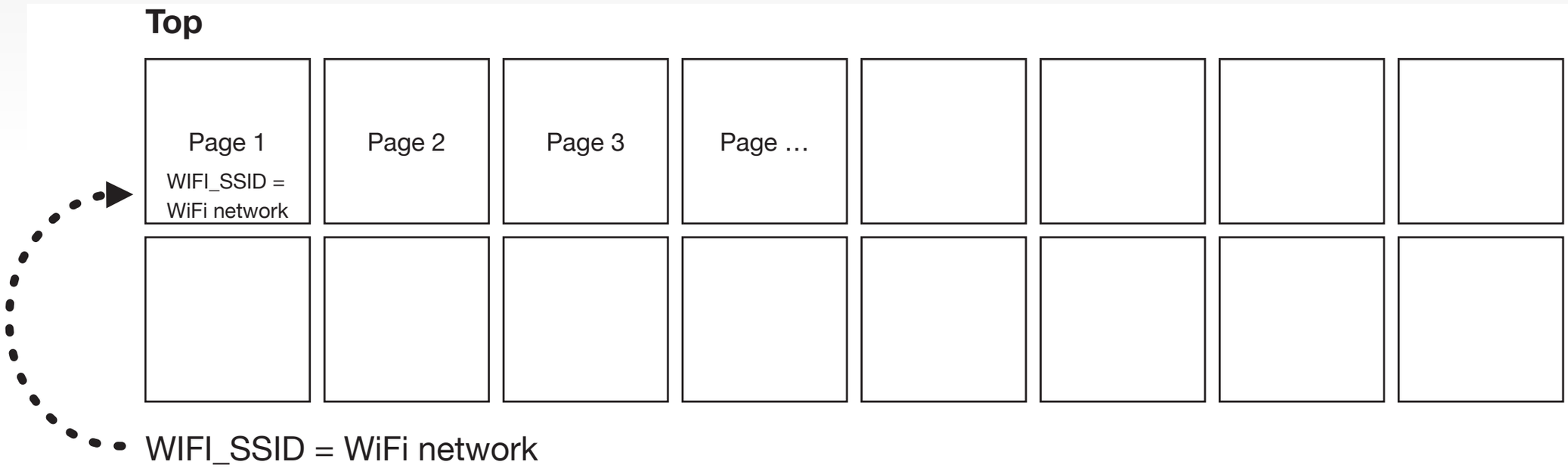
  Settings …?

Embedded Devices

Storage structure in MCU flash memory



Flash Memory Storage



- Settings stored in the flash memory are permanent (non-volatile)
- Permits read/write access by the app

Problem: Storage Sizes

- Flash size is limited, therefore data should be stored in an optimized way, e.g. using IDs:

  😟 WIFI_SSID=*WiFi network name*

  😀 30=*WiFi network name* (“ID 30” points to “WIFI_SSID”)

- Use types, e.g.

  “T_STR” 0 = strings

  “T_32BIT” 1 = integers

Problem: Limited Number Of Write Cycles

- Flash page erase/write cycles are limited
  - Between 1000 and 100000 cycles, vendor depending
- A flash memory storage system will not last very long if data is written to the flash because every write to a previously written block must first be read, erased, and re-written to the same location

Solution: Wear Leveling

- Wear leveling rotates data on flash memory pages to overcome the write cycles limitation
- Writes are distributed across all pages of the flash memory

10 writes	10 writes	11 writes	10 writes	10 writes	11 writes	10 writes	10 writes
11 writes	10 writes	11 writes	11 writes	10 writes	11 writes	10 writes	10 writes

RadioShuttle NVProperty Library

- Non-volatile properties settings in MCU flash memory
- Supports Arduino ESP32 (e.g. “ECO Power”)
- Supports Arduino D21 (e.g. “LongRa”)
- Supports Mbed OS boards (e.g. “Turtle”)
- Supports automatic “wear leveling”
- API: easy and identical on Arduino and Mbed OS

  Supports *add*, *delete* and *update* of properties
- RadioShuttle “Property Editor”

  Allows defining settings without programming knowledge

NVProperty Storage Location

Lookup Order

First — **RAM**

Random-Access Memory

Easy to set and to overwrite. Data stored in RAM memory is volatile, i.e. it gets lost upon loss of power

Second — **Flash**

Flash Memory

Stores data that is to be preserved (non-volatile)

Third — **OTP**

One Time Programmable

Non-volatile memory that permits data to be written to memory only once and which retains its value upon loss of power (non-volatile)

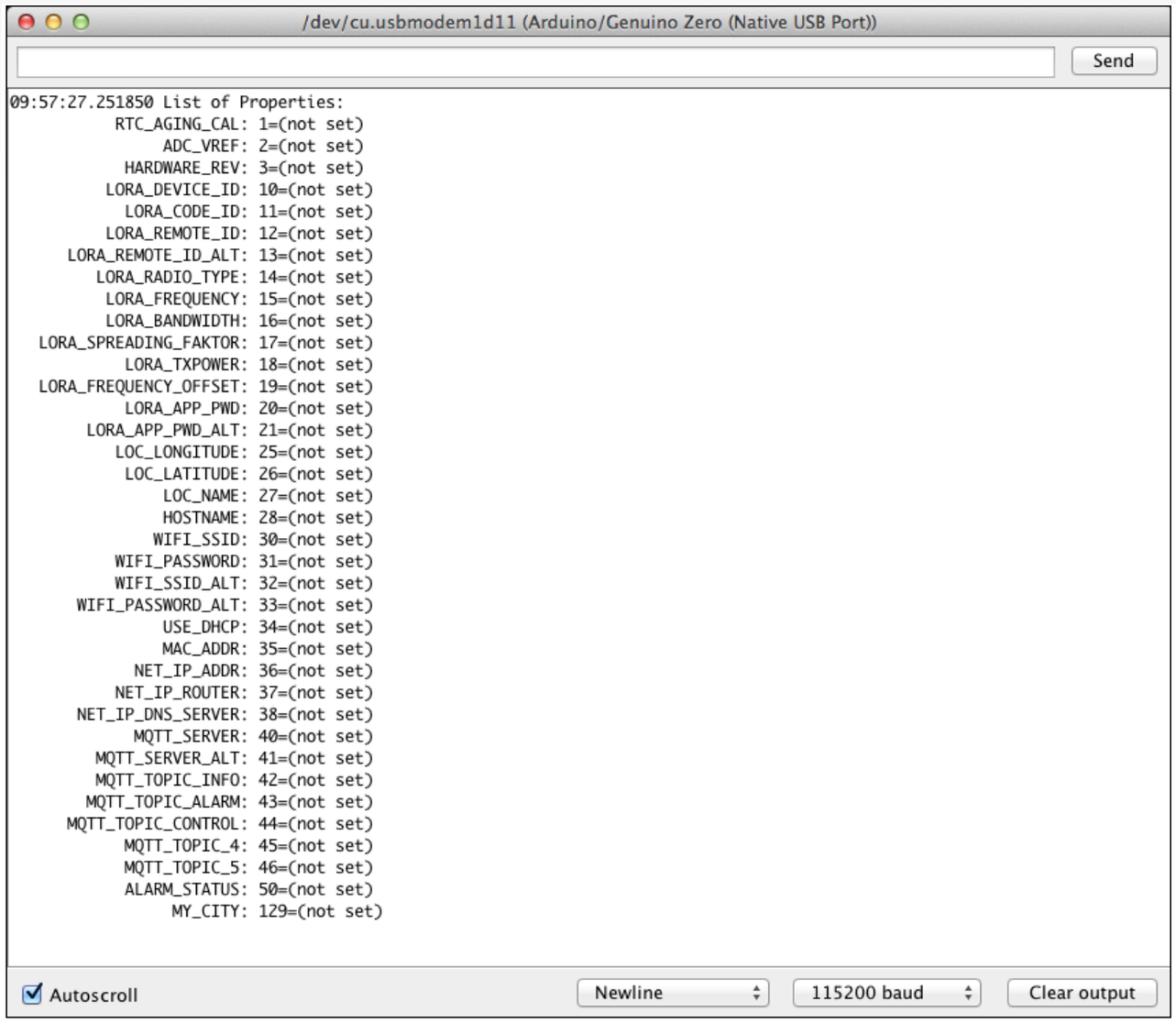
Note:

The lookup order has been implemented to allow overwriting settings from a different memory type.
For example, a value that was defined in flash can be overridden with a value specified in RAM.

NVProperty List

Property	ID	Type	Default
LORA_DEVICE_ID	10	T_32BIT	preset
LORA_CODE_ID	11	T_32BIT	preset
LORA_REMOTE_ID	12	T_32BIT	not set
LORA_REMOTE_ID_ALT	13	T_32BIT	not set
LORA_RADIO_TYPE	14	T_32BIT	not set
LORA_FREQUENCY	15	T_32BIT	not set
LORA_BANDWIDTH	16	T_32BIT	not set
LORA_SPREADING_FACTOR	17	T_8BIT	not set
LORA_TXPOWER	18	T_8BIT	not set
LORA_FREQUENCY_OFFSET	19	T_32BIT	not set
LORA_APP_PASSWD	20	T_STR	not set
LORA_APP_PASSWD_ALT	21	T_STR	not set
LOC_LONGITUDE	25	T_STR	not set
LOC_LATITUDE	26	T_STR	not set
LOC_NAME	27	T_STR	not set
HOSTNAME	28	T_STR	not set
WIFI_SSID	30	T_STR	not set
WIFI_PASSWORD	31	T_STR	not set

Property Editor (Arduino)



Commands for the Property Editor:

- `l` List all properties
- `s` Set property, e.g. `s28=myhost`
- `d` Delete property, e.g. `d28`

Property Editor (Mbed)

henk — tty.usbmodem1d11 — screen

```
$ l
List of Properties:
           RTC_AGING_CAL: 1=(not set)
                ADC_VREF: 2=(not set)
            HARDWARE_REV: 3=(not set)
          LORA_DEVICE_ID: 10=11000 (0xfaf8)
            LORA_CODE_ID: 11=-588302282 (0xdcef3836)
          LORA_REMOTE_ID: 12=(not set)
      LORA_REMOTE_ID_ALT: 13=(not set)
         LORA_RADIO_TYPE: 14=(not set)
          LORA_FREQUENCY: 15=(not set)
          LORA_BANDWIDTH: 16=(not set)
   LORA_SPREADING_FACTOR: 17=(not set)
           LORA_TXPOWER: 18=(not set)
   LORA_FREQUENCY_OFFSET: 19=(not set)
           LORA_APP_PWD: 20=(not set)
       LORA_APP_PWD_ALT: 21=(not set)
          LOC_LONGITUDE: 25=(not set)
           LOC_LATITUDE: 26=(not set)
               LOC_NAME: 27=(not set)
               HOSTNAME: 28=(not set)
              WIFI_SSID: 30=(not set)
          WIFI_PASSWORD: 31=(not set)
          WIFI_SSID_ALT: 32=(not set)
      WIFI_PASSWORD_ALT: 33=(not set)
               USE_DHCP: 34=(not set)
               MAC_ADDR: 35=(not set)
            NET_IP_ADDR: 36=(not set)
         NET_IP_ROUTER: 37=(not set)
      NET_IP_DNS_SERVER: 38=(not set)
            MQTT_SERVER: 40=(not set)
        MQTT_SERVER_ALT: 41=(not set)
        MQTT_TOPIC_INFO: 42=(not set)
       MQTT_TOPIC_ALARM: 43=(not set)
     MQTT_TOPIC_CONTROL: 44=(not set)
           MQTT_TOPIC_4: 45=(not set)
           MQTT_TOPIC_5: 46=(not set)
           ALARM_STATUS: 50=(not set)
                MY_CITY: 129=(not set)

$
```

Commands for the Property Editor:

- `l` List all properties
- `s` Set property, e.g. `s28=myhost`
- `d` Delete property, e.g. `d28`

Get Properties

```
        /*
         * A simple GetProperty returns its values as an int or int64
         * The order should always be S_RAM,S_FLASH, S_OTP
         */
        int GetProperty(int key, int defaultValue = 0);
Example: int id = GetProperty(LORA_REMOTE_ID);

        /*
         * GetProperty receives a copy of the property, use free to release it.
         */
        const char *GetProperty(int key, const char *defaultValue = NULL);
Example: const char *password = GetProperty(WIFI_SSID);

        /*
         * when a block is returned, the buffer is filled up to the property
         * or max at the bsize length.
         * If the buffer is NULL only the size value will be set
         */
        int GetProperty(int key, void  *buffer, int *size);
Example: int ret = GetProperty(LORA_APP_PWD, buffer, sizeof(buffer));
```

Set Properties

```
        /*
         * SetProperty
         * It requires to use OpenPropertyStore and finally ClosePropertyStore(true)
         * to write out all properties.
         * Properties are limited to 256 bytes.
         * (e.g. 255 long strings or 256 bytes blobs)
         *
         * Number properties e.g. 0 or 1, or 123, are highly optimized in storage
         * sizes.
         * Therefore the value is automatically compressed to a bit or a low
         * number value to use less flash storage space.
         */
        int SetProperty(int key, NVPType ptype, int64_t value, NVPStore store =
        S_FLASH);
Example: SetProperty(LORA_REMOTE_ID, T_32BIT, 123, S_FLASH);
```

Erase Properties

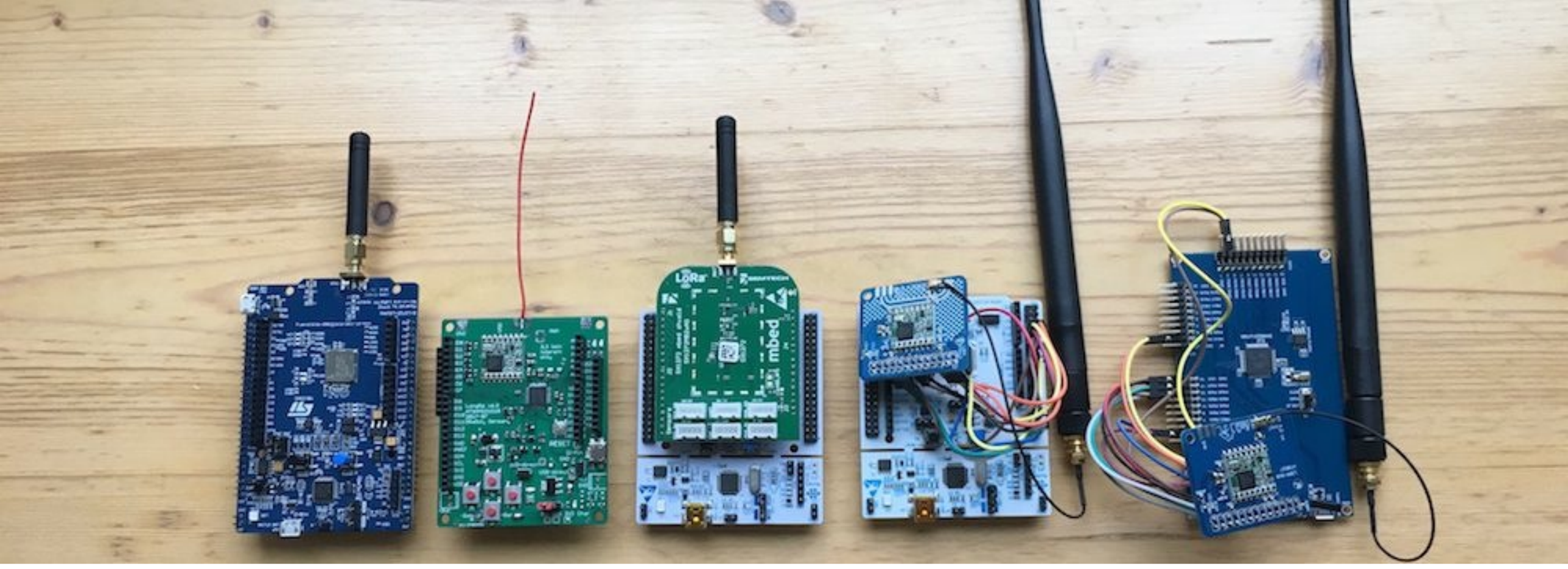
```
        /*
         Erase removes the property from the specified store
         */
        int EraseProperty(int key, NVPStore store = S_FLASH);
```

Example: `EraseProperty(WIFI_SSID, S_FLASH);`

NVProperty Summary

- Allows millions of property writes
- No EEPROM or additional flash chip required
- Easy to use (Property Editor, API)
- Supports Arduino (ESP32, D21) and Mbed OS
- Licensed by RadioShuttle.de

  Software included with RadioShuttle boards



www.radioshuttle.de

See you in the Comments section!

Helmut Tschemernjak