Rust for embedded devices

BLE & WiFi

EchoKit

Star, clone and fork 👍

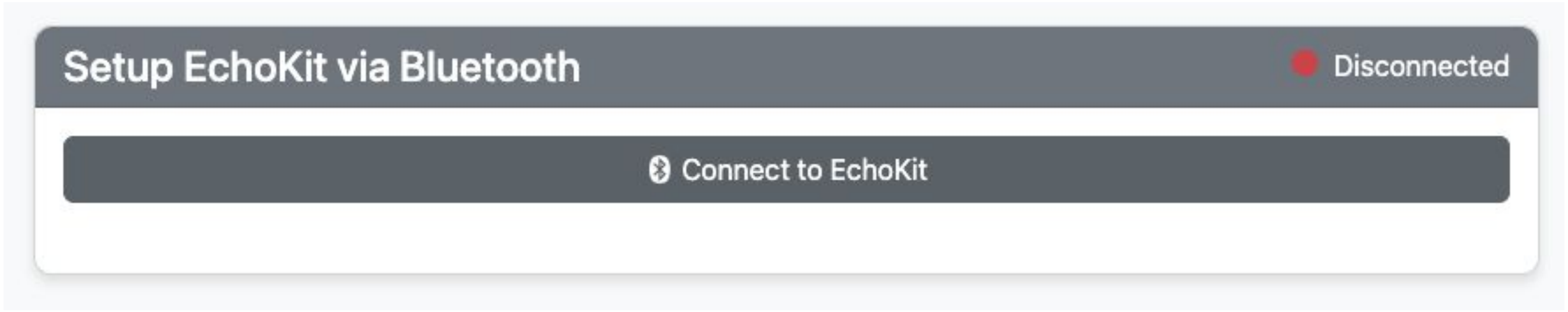
EchoKit devices: https://github.com/second-state/echokit_box

EchoKit server: https://github.com/second-state/echokit_server

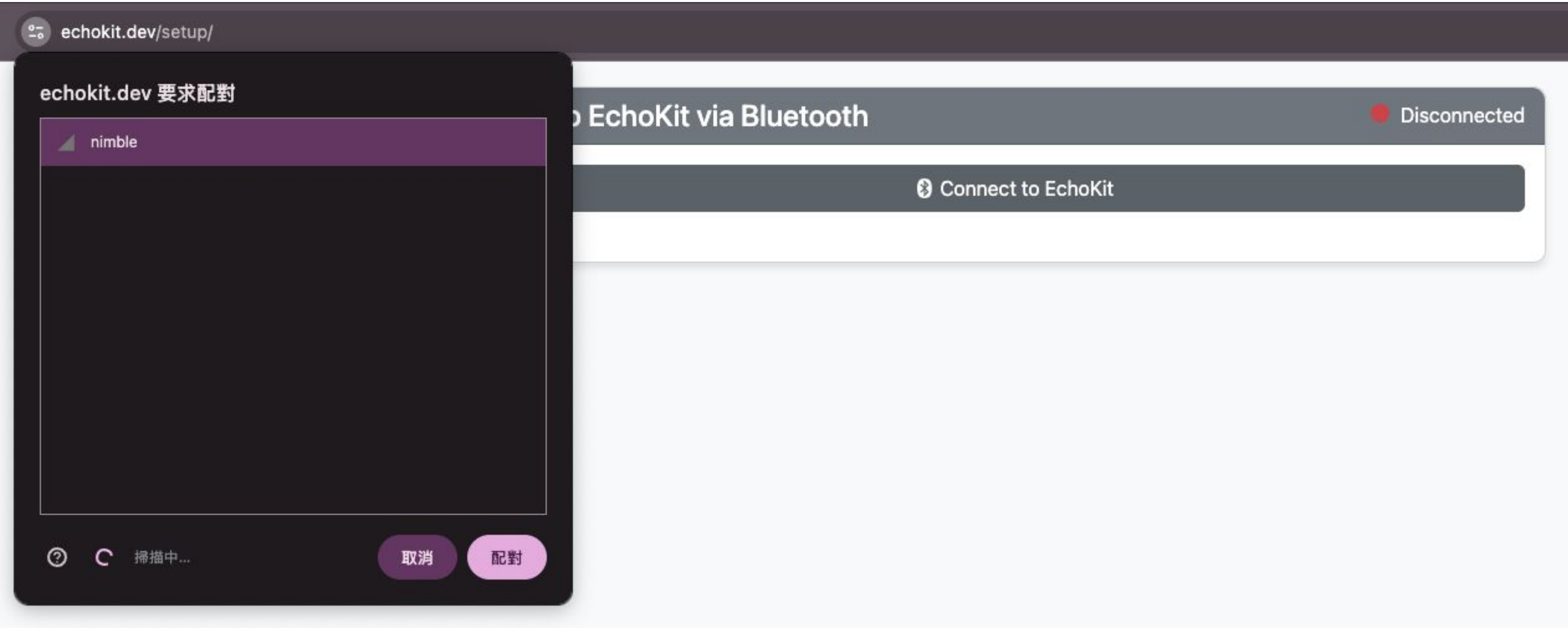
https://github.com/second-state/echokit_box
https://echokit.dev/setup/

Demo: Connect to echokit

Click “connect to echokit



Choose the ble devices



echokit.dev/setup/

echokit.dev 要求配對

nimble

掃描中...

取消

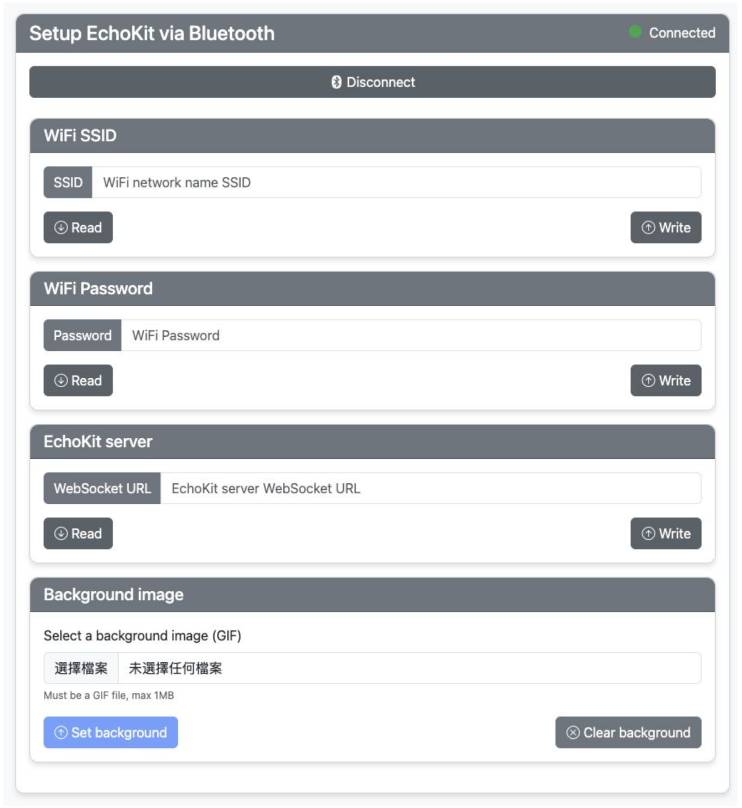
配對

EchoKit via Bluetooth

Disconnected

Connect to EchoKit

The setup UI


Setup EchoKit via Bluetooth
Connected
Disconnect
WiFi SSID
SSID
WiFi network name SSID
Read
Write
WiFi Password
Password
WiFi Password
Read
Write
EchoKit server
WebSocket URL
EchoKit server WebSocket URL
Read
Write
Background image
Select a background image (GIF)
選擇檔案
未選擇任何檔案
Must be a GIF file, max 1MB
Set background
Clear background

setup/index.html
https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API

Dig into UI

UUIDs

UUID should be the same as the constants on the devices (more on this later).

```
// UUIDs

const SERVICE_ID = "623fa3e2-631b-4f8f-a6e7-a7b09c03e7e0";

const SSID_ID = "1fda4d6e-2f14-42b0-96fa-453bed238375";

const PASS_ID = "a987ab18-a940-421a-a1d7-b94ee22bccbe";

const SERVER_URL_ID = "cef520a9-bcb5-4fc6-87f7-82804eee2b20";

const BACKGROUND_IMAGE_ID = "d1f3b2c4-5e6f-4a7b-8c9d-0e1f2a3b4c5d";
```

Connect to a BLE devices

```
async function connectToDevice() {
    try {
        device = await navigator.bluetooth.requestDevice({
            filters: [{ services: [SERVICE_ID] }],
            optionalServices: [SERVICE_ID]
        });

        // connect to GATT(Generic Attribute Profile)
        server = await device.gatt.connect();
        service = await server. getPrimaryService(SERVICE_ID);
        ...
        // Process the disconnect event
        device.addEventListener(
            'gattserverdisconnected', handleDisconnection);

    } catch (error) {...}
}
```

Disconnect from a BLE device

```
async function disconnectFromDevice() {
    if (device && device.gatt.connected) {
        try {
            await device.gatt.disconnect();
            ...
        } catch (error) {...}
    }
}
```

Read Characteristic from a BLE device

```
async function readCharacteristic(characteristicId, inputElement) {
    if (!isConnected || !service) { return; }

    try {
        const characteristic =
            await service.getCharacteristic(characteristicId);
        const value = await characteristic.readValue();

        const decoder = new TextDecoder();
        const stringValue = decoder.decode(value);
        ...
    } catch (error) {...}
}
```

Write Characteristic to a BLE device

```
async function writeCharacteristic(characteristicId, inputValue) {
    if (!isConnected || !service) { return; }
    if (!inputValue) { return; }

    try {
        const characteristic =
            await service.getCharacteristic(characteristicId);
        const encoder = new TextEncoder();
        const data = encoder.encode(inputValue);
        await characteristic.writeValue(data);
        ...
    } catch (error) { ... }
}
```

Write an image (GIF) to a BLE device

```
async function writeBackgroundImage() {
    if (!isConnected || !service) { return; }
    if (!selectedBackgroundFile) { return; }

    try {
        const characteristic =
            await service.getCharacteristic(BACKGROUND_IMAGE_ID);

        const arrayBuffer =
            await selectedBackgroundFile.arrayBuffer();
        const totalSize = arrayBuffer.byteLength;
        const chunkSize = 512; // BLE limit
        const totalChunks = Math.ceil(totalSize / chunkSize);

        for (let i = 0; i < totalChunks; i++) {
            ...
            await characteristic.writeValue(packet);
            // small delay to avoid overloading the BLE stack
            await new Promise(resolve => setTimeout(resolve, 50));
        }
        ...
    } catch (error) { ... }
}
```

src/bt.rs

BLE

BLE - NimBLE sdkconfig.defaults

To use this SDK, we must enable some config in sdkconfig.defaults (in root folder)

```
CONFIG_BT_ENABLED=y
CONFIG_BT_BLE_ENABLED=y
CONFIG_BT_BLUEDROID_ENABLED=n
CONFIG_BT_NIMBLE_ENABLED=y
```

Increasing esp-ble task stack size for heavier compute loads
Normally, it should vary between 4096 and 5120 [1]

```
CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE=7000
```

[1]:
https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/kconfig-reference.html

BLE - UUIDs

Using GATT, we must set the UUIDs for each service and characteristic. You can put any UUID you want.

```
const SERVICE_ID: BleUuid = uuid128!("623fa3e2-631b-4f8f-a6e7-a7b09c03e7e0");
const SSID_ID: BleUuid = uuid128!("1fda4d6e-2f14-42b0-96fa-453bed238375");
const PASS_ID: BleUuid = uuid128!("a987ab18-a940-421a-a1d7-b94ee22bccbe");
const SERVER_URL_ID: BleUuid = uuid128!("cef520a9-bcb5-4fc6-87f7-82804eee2b20");
const BACKGROUND_GIF_ID: BleUuid = uuid128!("d1f3b2c4-5e6f-4a7b-8c9d-0e1f2a3b4c5d");
```

How to create a service and a characteristic

```
let service = server.create_service(SERVICE_ID);
ssid_characteristic = service
        .lock()
        .create_characteristic(SSID_ID,
            NimbleProperties::READ | NimbleProperties::WRITE);
```

BLE - The ble_device

Everything we need for the ble device is from `esp32_nimble::BLEDevice`

```
let ble_device = esp32_nimble::BLEDevice::take();
```

We can retrieve these informations

```
let ble_addr = ble_device.get_addr()?.to_string();
let ble_advertising = ble_device.get_advertising();
let server = ble_device.get_server();
```

BLE - Server.on_connect

Use `on_connect` to handle a client

```
server.on_connect(|server, desc| {
    /// * `conn_handle`: The connection handle of the peer to send the request to.
    /// * `min_interval`: The minimum connection interval in 1.25ms units.
    /// * `max_interval`: The maximum connection interval in 1.25ms units.
    /// * `latency`: The number of packets allowed to skip (extends max interval).
    /// * `timeout`: The timeout time in 10ms units before disconnecting.
    server
        .update_conn_params(desc.conn_handle(), 24, 48, 0, 60)
        .unwrap();

    if server.connected_count() <
       (esp_idf_svc::sys::CONFIG_BT_NIMBLE_MAX_CONNECTIONS as _) {
        log::info!("Multi-connect support: start advertising");
        ble_advertising.lock().start().unwrap();
    }
});
```

BLE - Server.on_disconnect

Use `on_disconnect` to handle the event that the client is disconnect

```
server.on_disconnect(|_desc, reason| {
        /// reason: The reason code for the disconnection.
        log::info!("Client disconnected ({:?})", reason);
});
```

BLE - Server.create_characteristic

```
let service = server.create_service(SERVICE_ID);

let ssid_characteristic = service.lock()
    .create_characteristic(
         SSID_ID, NimbleProperties::READ | NimbleProperties::WRITE);

ssid_characteristic.lock()
    .on_read(move |c, _| {
        let setting = setting1.lock().unwrap();
        c.set_value(setting.0.ssid.as_bytes());
    })
    .on_write(move |args| {
        if let Ok(new_ssid) = String::from_utf8(args.recv_data().to_vec()) {
            let mut setting = setting2.lock().unwrap();
            if let Err(e) = setting.1.set_str("ssid", &new_ssid) {...}
            else {setting.0.ssid = new_ssid;}
        } else {...}
    });
```

BLE - Image Characteristic

```
let background_gif_characteristic = service
    .lock()
    .create_characteristic(
         BACKGROUND_GIF_ID, NimbleProperties::WRITE);

background_gif_characteristic.lock().on_write(move |args| {
    let gif_chunk = args.recv_data();

    if gif_chunk.len() <= 1024 * 1024 && gif_chunk.len() > 0 {
        let mut setting = setting_gif.lock().unwrap();
        setting.0.background_gif.0.extend_from_slice(gif_chunk);
        if gif_chunk.len() < 512 {
            setting.0.background_gif.1 = true; // Mark as valid
        }
    } else {
        log::error!("Failed to parse new background GIF from bytes.");
    }
});
```

BLE - Advertising

```
ble_advertising.lock().set_data(
        BLEAdvertisementData::new()
            .name(&format!("EchoKit-{}", ble_addr))
            .add_service_uuid(SERVICE_ID),
    )?;
ble_advertising.lock().start()?;
```

src/network.rs

WiFi

WiFi - Function Signature

```
pub fn wifi(
    ssid: &str,
    pass: &str,
    modem:
      impl peripheral::Peripheral
         <P = esp_idf_svc::hal::modem::Modem> + 'static,
    sysloop: EspSystemEventLoop,
) -> anyhow::Result<Box<EspWifi<'static>>> {}
```

WiFi - Authentication

```
let mut auth_method = AuthMethod::WPA2Personal;
if ssid.is_empty() {
    anyhow::bail!("Missing WiFi name")
}
if pass.is_empty() {
    auth_method = AuthMethod::None;
    info!("Wifi password is empty");
}
```

WiFi - ESPWiFi Configuration

```rust
let mut esp_wifi = EspWifi::new(modem, sysloop.clone(), None)?;
let mut wifi = BlockingWifi::wrap(&mut esp_wifi, sysloop)?;

wifi.set_configuration(&esp_idf_svc::wifi::Configuration::Client(
    esp_idf_svc::wifi::ClientConfiguration {
        ssid: ssid
            .try_into()
            .expect("Could not parse the given SSID into WiFi config"),
        password: pass
            .try_into()
            .expect("Could not parse the given password into WiFi config"),
        auth_method,
        ..Default::default()
    },
))?;
```

WiFi - Start WiFi

```
wifi.start()?;
wifi.connect()?; // Connect to WiFi
wifi.wait_netif_up()?; // Wait for DHCP
let ip_info = wifi.wifi().sta_netif().get_ip_info()?;
let mac = wifi.ap_netif().get_mac()?;
Ok(Box::new(esp_wifi)) // The final return value of wifi()
```

Reference

Reference

- esp32-nimble
  - https://taks.github.io/esp32-nimble/esp32_nimble/index.html
- ESP-IDF WiFi Guide
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/api-guides/wifi.html
- Web Bluetooth Spec
  - https://webbluetoothcg.github.io/web-bluetooth/
  - https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API

Until next time!