МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Центральноукраїнський національний технічний університет

Смірнов В.В., Смірнова Н.В.

Програмування пристроїв Internet Of Things на базі мікроконтролера ESP32. Протоколи мережевих модулів

Навчальний посібник

Кропивницький - 2024

УДК: 004.41
ББК 32.97
С50

Рекомендовано Вченою радою Центральноукраїнського національного технічного університету до друку та використанню підручника у навчальному процесі здобувачами вищої освіти очної та заочної форми навчання.
Протокол № від 2024 року

Рецензенти:

Дідик Олександр Костянтинович, кандидат технічних наук, завідувач кафедри автоматизації виробничих процесів Центральноукраїнського національного технічного університету.

Баранюк Олександр Филимонович, кандидат технічних наук, доцент кафедри інформатики та інформаційних технологій Центральноукраїнського державного університету імені Володимира Винниченка.

Смірнов В.В., Смірнова Н.В.

С50 Програмування пристроїв Internet Of Things на базі мікроконтролера ESP32. Протоколи мережевих модулів: навчальний посібник. – Кропивницький : ЦНТУ, 2024. – 400 с.

У навчальному посібнику с наведено технічні характеристики мікроконтролера ESP32. Наведено опис функцій API для програмування сетевых модулів мікроконтролера ESP32 таких, як MQTT, HTTP-сервера. HTTP - клієнта, HTTP-сервера, WEBSOCKET-сервера, WEBSOCKET - клієнта, служби MDNS. Представлено опис функцій API протоколу MODBUS для різних виробників.

Представлено опис компонента локального керування ESP32 у ESP-IDF який забезпечує можливість керування пристроєм через HTTPS або Bluetooth® Low Energy.

Представлено опис бездротової локальної мережі «Easy Net Everywhere» на базі якої мікроконтролери ESP8266 за допомогою інтерфейсу UART на локальному рівні об'єднуються для вирішення різноманітних завдань.

Описано конфігурацію різних моделей побудови мережі, їхні особливості та практичну реалізацію. Описано систему адресації вузлів мережі для простої та кластерної моделі мережі. Описано процес конфігурування, тестування та перевірки працездатності мережі.

Навчальний посібник призначений для здобувачів вищої освіти очної та заочної форми навчання за спеціальностями:

122 «Комп’ютерні науки»;
123 «Комп’ютерна інженерія»;
172 «Електронні комунікації та радіотехніка»;
151 «Автоматизація та комп'ютерно-інтегровані технології».

Навчальне електронне видання комбінованого використання.
Можна використовувати в локальному та мережному режимах

УДК: 004.41
ББК 32.97

ЗМІСТ

ВСТУП

На сьогодні розробка та програмування устройств «Internet Of Things» є актуальним завданням.

Технологія "Internet Of Things" у загальному випадку включає в себе:

- глобальну мережу Internet;
- мережеві протоколи Internet;
- мікроконтролери з вбудованими апаратними та мережевими програмними модулями;
- програмне забезпечення (API) для програмування вбудованих апаратних і програмних мережевих модулів;
- локальну бездротову мережу для організації взаємодії мікроконтролерів на локальному рівні.

Згідно з дослідженням Strategy Analytics, у 2024 році в світі налічувалося близько 32 млрд IoT пристроїв. За прогнозом компанії, в 2025 р. кількість IoT пристроїв складе 38,6 млрд, а у 2030 р. перевищить понад 50 млрд.

З цього випливає, що технології IoT і IIoT є дуже перспективними і затребуваними практично у всіх областях діяльності людини.

Ці технології вимагають великої кількості кваліфікованих і високооплачуваних фахівців з грунтовними знаннями електроніки, схемотехніки, мікропроцесорної техніки та програмування.

З огляду на величезний попит на таких фахівців, багато провідних закордонних університетів здійснюють підготовку фахівців в області IoT і IIoT.

Тому у навчальному посібнику подано опис функцій API для програмування мережевих модулів мікроконтролера ESP32, надани базові відомості з протоколів мережевих модулів та надано опис бездротової локальної мережі «Easy Net Everywhere».

Концепція Internet Of Things для контролера ESP32 представлена на рис.1.

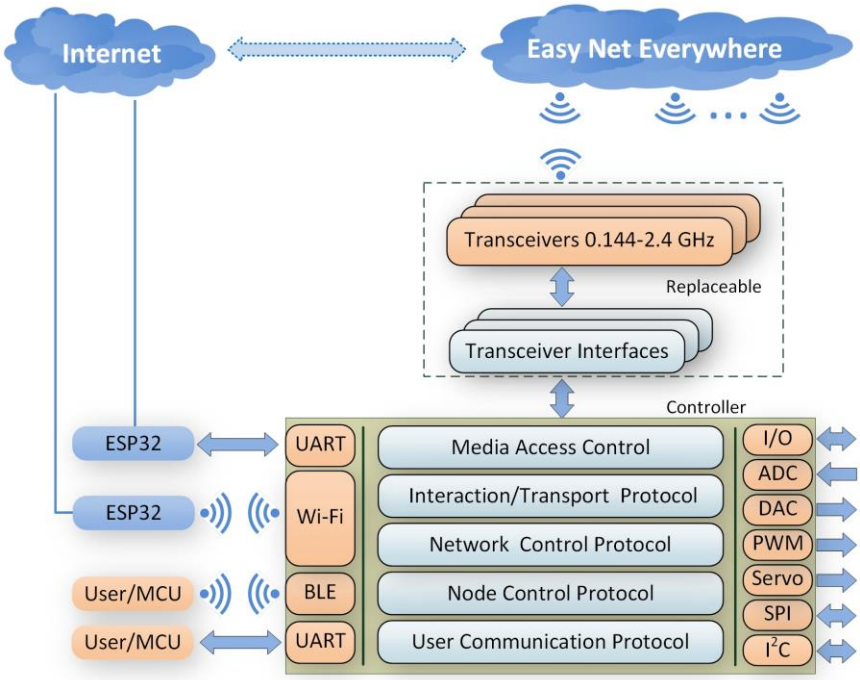


Рисунок 1 - Концепція Internet Of Things для контролера ESP32

Завданням навчального посібника є навчання студентів навичкам роботи з мережевими модулями, протоколами та API для мікроконтролера ESP32 спільно з бездротовою локальною мережею «Easy Net Everywhere», а також для підготовки програмістів в області розробки та управління територіально-розподіленими системами, роботами, об'єктами і комплексами.

Навчальний посібник призначений для здобувачів вищої освіти очної та заочної форми навчання за спеціальностями:

122 «Комп’ютерні науки»;

123 «Комп’ютерна інженерія»;

172 «Електронні комунікації та радіотехніка»;

151 «Автоматизація та комп'ютерно-інтегровані технології».

1. ОПИС МІКРОКОНТРОЛЕРА ESP32

ESP32 - серія недорогих мікроконтролерів з малим енергоспоживанням китайської компанії Espressif Systems.

Мікроконтролери ESP32 розроблені спеціально для створення пристроїв Internet Of Things і являють собою систему на кристалі з інтегрованим контролерами радіозв'язку Wi-Fi і Bluetooth (BLE).

У серіях ESP32 і ESP32-S використовуються процесорні ядра з архітектурою компанії Tensilica, а в серіях ESP32-C, ESP32-H, ESP32-P застосовуються ядра з відкритою архітектурою RISC-V.

У мікроконтролер інтегровано радіочастотний тракт: симетризуючий трансформатор, вбудовані антенні комутатори, радіочастотні компоненти, малошумний підсилювач, підсилювач потужності, фільтри та модулі керування живленням.

Ядро

- Tensilica Xtensa LX6 двоядерний / одноядерний 32-розрядний процесор, з тактовою частотою 160 або 240 МГц і продуктивністю до 600 DMIPS.
- Співпроцесор з ультранизьким енергоспоживанням - Пам'ять: 520 КБ пам'яті SRAM.

Бездротовий зв'язок:

- Wi-Fi: 802.11 b / g / N.
- Bluetooth: v4.2 BR/EDR and BLE.

Периферійні інтерфейси:

- 2-розрядний АЦП до 18 каналів.
- 2 × 8 біт ЦАП. - 10 × портів для підключення ємнісних датчиків.
- 4 × SPI майстер-інтерфейси (провідні пристрої).
- 2 × I²S майстер-інтерфейси.

- 2 × I²C майстер-інтерфейси.
- 3 × UART інтерфейси. - SD/SDIO/CE-ATA/MMC/ eMMC хост-контролер. - SDIO/SPI слейв-контролери (ведені пристрої).
- Ethernet MAC interface з виділеним DMA
- CAN bus 2.0.
- IR дистанційне керування (передавач/приймач, до 8 каналів).
- PWM. - Датчик Холла.
- Аналоговий передпідсилювач низького енергоспоживання.

Безпека:

- Підтримуються всі функції безпеки стандарту IEEE 802.11, зокрема WFA, WPA/WPA2 і WAPI.
- Безпечне завантаження.
- Шифрування флеш-диска.
- 1024-бітний ключ, до 768 біт для клієнтів.
- Криптографічне апаратне прискорення: AES, SHA-2, RSA, криптографії на основі еліптичних кривих (ECC), апаратний генератор випадкових чисел за увімкненого WiFi або Bluetooth, інакше використовується генератор псевдовипадкових чисел.

Керування живленням:

- Лінійний регулятор із низьким рівнем падіння напруги. - Індивідуальне живлення для RTC.
- Споживання 5-2,5 мкА в режимі "глибокий сон". - Пробудження за перериванням від GPIO, таймера, вимірювання АЦП, переривання ємнісного сенсорного датчика.
- Робоча напруга від 2,2-3,6 В.
- Робоча температура від -40 °C до +125 °C.

- Максимальна швидкість передавання даних 150 Мбіт/с за умови 11n HT40, 72 Мбіт/с за умови 11n HT20, 54 Мбіт/с @ 11g, та 11 Мбіт/с за умови 11b.
- Максимальна потужність передавання 19,5 дБм.
- Мінімальна чутливість приймача: 98 дБм. - Стійка пропускна здатність UDP 135 Мбіт/с.

Wi-fi

- 802.11 n (2.4 GHz), up to 150 Mbps
- 802.11 e: QoS for wireless multimedia technology
- WMM-PS, UAPSD
- A-MPDU and A-MSDU aggregation
- Block ACK
- Fragmentation and defragmentation
- Automatic Beacon monitoring/scanning
- 802.11 i security features: pre-authentication and TSN
- Wi-Fi Protected Access (WPA)/WPA2/WPA2-Enterprise/Wi-Fi Protected Setup (WPS)
- Infrastructure BSS Station mode/SoftAP mode
- Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode and P2P Power Management
- UMA compliant and certified
- Antenna diversity and selection
- Compliant with Bluetooth v4.2 BR/EDR and BLE specification

Bluetooth

- Class-1, class-2 and class-3 transmitter without external power amplifier
- Enhanced power control
- +10 dBm transmitting power
- NZIF receiver with −98 dBm sensitivity
- Adaptive Frequency Hopping (AFH)
- Standard HCI based on SDIO/SPI/UART
- High speed UART HCI, up to 4 Mbps
- BT 4.2 controller and host stack
- Service Discover Protocol (SDP)
- General Access Profile (GAP)
- Security Manage Protocol (SMP)
- Bluetooth Low Energy (BLE)
- ATT/GATT
- HID
- All GATT-based profile supported
- SPP-Like GATT-based profile
- BLE Beacon
- A2DP/AVRCP/SPP, HSP/HFP, RFCOMM
- CVSD and SBC for audio codec
- Bluetooth Piconet and Scatternet

Зовнішня FLASH і SRAM

ESP32 підтримує:

- До чотирьох банків 16-Мб зовнішньої flash QSPI і SRAM з апаратним шифруванням на основі AES із захистом користувацьких програм і даних.

- До 16 Мб зовнішньої флеш-пам'яті, співставлених із кодовим простором ЦП, що підтримують 8, 16 і 32-бітний доступ. Підтримується виконання коду.
- До 8 Мб зовнішньої flash/SRAM карти пам'яті на ЦП простору даних, підтримка 8, 16 і 32-біт доступу. Читання даних підтримується на флеш-пам'яті та SRAM. Запис даних підтримується на SRAM.
- ESP32-WROVER інтегрує 4-16 Мб зовнішньої SPI flash. 4-мб SPI flash може бути карта пам'яті на процесорний простір, що підтримують 8, 16 і 32 біт доступу. Підтримується виконання коду.

Кварцові генератори

- Мікропрограма ESP32 Wi-Fi/BT може підтримувати тільки кварцовий генератор 40 і більше МГц.

Корпус

- ESP32 випускається в планарному корпусі QFN.

Режими живлення/Power modes

- Active mode / Активний режим: чип радіо ввімкнено. Чип може отримувати, передавати або слухати.
- Modem-sleep mode / Режим сну модему: ЦП працює і годинник налаштовується. Базова смуга Wi-Fi/Bluetooth I радіо вимкнено.
- Light-sleep mode / Режим сну: ЦП призупинено. Пам'ять RTC і периферійні пристрої RTC, а також ULP Сопроцесор працює.
- Deep-sleep mode/Режим глибокого сну: Лише пам'ять RTC і периферійні пристрої RTC увімкнено. Wi-Fi і Bluetooth дані з'єднання зберігаються в пам'яті RTC. Співпроцесор ULP може працювати.

- Hibernation mode / Режим глибокого сну: внутрішній 8 МГц осцилятор і со-процесор ULP вимкнено. RTC відновлення пам'яті вимкнено. Тільки один таймер RTC на повільному годиннику і деякі GPIOs RTC активні.

Мови програмування, платформи та середовища, що використовуються для програмування ESP32:

- Arduino IDE с ESP32 Arduino Core
- Espressif IoT Development Framework - Официальная Espressif разработка для ESP32.
- Espruino - JavaScript SDK, эмулятор Node.js.
- Lua RTOS.
- Mongoose OS - Операційна система для переносної електроніки, рекомендована Espressif Systems, AWS IoT, Google Cloud IoT.
- PlatformIO Ecosystem и IDE
- Pymakr IDE
- Simba Embedded Programming Platform
- Whitecat Ecosystem Blockly основана на Web IDE
- MicroPython
- Zerynth - Python для IoT і мікроконтролерів, включно з ESP32.
- OWLOS - мережева операційна система з відкритим вихідним кодом для управління IoT пристроями.
- uLisp - Lisp для мікроконтролерів.
- ECS - Eigen Compiler Suite (C/C++/Oberon-2).

2. ПРОТОКОЛИ МЕРЕЖЕВИХ МОДУЛІВ ESP32

КЛІЄНТ ПРОТОКОЛУ ESP-MQTT

Огляд

ESP-MQTT - це реалізація клієнта протоколу MQTT , який є полегшеним протоколом для публікації/підписки. Тепер ESP-MQTT підтримує MQTT v5.0 .

Особливості

Підтримка MQTT через TCP, SSL з Mbed TLS, MQTT через WebSocket і MQTT через WebSocket Secure.

Підтримка підписки, публікації, автентифікації, останніх повідомлень, підтримки ping і всіх 3 рівнів якості обслуговування (QoS) (це має бути повнофункціональний клієнт)

Передача повідомлень MQTT

Нове повідомлення MQTT створюється викликом **esp_mqtt_client_publish** або його неблокуючим аналогом **esp_mqtt_client_enqueue**.

Повідомлення з QoS 0 надсилаються лише один раз. QoS 1 і 2 мають різну поведінку, оскільки протокол вимагає додаткових кроків для завершення процесу.

Бібліотека ESP-MQTT вирішує завжди повторно передавати непідтверджені повідомлення про публікацію QoS 1 і 2, щоб уникнути втрат через несправні з’єднання, навіть якщо специфікація MQTT вимагає повторної передачі лише після повторного з’єднання з прапорцем чистого сеансу, встановленим на 0.

Повідомлення QoS 1 і 2, які можуть потребувати повторної передачі, завжди ставляться в чергу, але перша спроба передачі відбувається негайно, якщо esp_mqtt_client_publish використовується.

Повторна спроба передачі для непідтверджених повідомлень відбудеться після message_retransmit_timeout.

Після CONFIG_MQTT_OUTBOX_EXPIRED_TIMEOUT_MS повідомлення закінчуються та їх видаляють.

Якщо встановлено CONFIG_MQTT_REPORT_DELETED_MESSAGES, буде надіслано подію для сповіщення користувача.

Конфігурація

Конфігурація виконується встановленням полів у **esp_mqtt_client_config_t** структурі. Структура конфігурації має наступні підструктури для налаштування різних аспектів роботи клієнта.

- **esp_mqtt_client_config_t::broker_t-** Дозволяє встановити адресу та перевірку безпеки.
- **esp_mqtt_client_config_t::credentials_t-** Облікові дані клієнта для автентифікації.
- **esp_mqtt_client_config_t::session_t-** Конфігурація для аспектів сесії MQTT.
- **esp_mqtt_client_config_t::network_t-** Конфігурація мережі.
- **esp_mqtt_client_config_t::task_t-** Дозволяє налаштувати завдання FreeRTOS.
- **esp_mqtt_client_config_t::buffer_t-** Розмір буфера для введення та виведення.

БРОКЕР MQTT

Адреса

Адресу брокера можна встановити за допомогою **address** структури. Конфігурацію можна виконати за допомогою **uri** поля або комбінації **hostname**, **transport** і **port**. Необов'язково, це **path** поле може бути встановлено, це поле є корисним у підключеннях WebSocket.

Поле **uri** використовується у форматі scheme://hostname:port/path.

Наразі підтримуються схеми mqtt, mqtts, ws, wss

- Зразки MQTT через TCP:
  - mqtt://mqtt.eclipseprojects.io: MQTT через TCP, стандартний порт 1883
  - mqtt://mqtt.eclipseprojects.io:1884: MQTT через TCP, порт 1884
  - mqtt://username:password@mqtt.eclipseprojects.io:1884: MQTT через TCP, порт 1884, з іменем користувача та паролем
- Приклади MQTT через SSL:
  - mqtts://mqtt.eclipseprojects.io: MQTT через SSL, порт 8883
  - mqtts://mqtt.eclipseprojects.io:8884: MQTT через SSL, порт 8884
- Приклади MQTT через WebSocket:
  - ws://mqtt.eclipseprojects.io:80/mqtt
- Приклади MQTT через WebSocket Secure:
  - wss://mqtt.eclipseprojects.io:443/mqtt

- Мінімальні комплектації:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .broker.address.uri = "mqtt://mqtt.eclipseprojects.io",
};
esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID,
mqtt_event_handler, client);
esp_mqtt_client_start(client);
```

За замовчуванням клієнт MQTT використовує бібліотеку циклів подій для публікації пов'язаних подій MQTT (підключення, підписка, публікація тощо).

Перевірка

Для безпечних з'єднань із використанням TLS і для гарантування ідентичності посередника **verification** необхідно встановити структуру. Сертифікат брокера може бути встановлений у форматі PEM або DER. Щоб вибрати PEM, **certificate_len** необхідно встановити еквівалентне поле. В іншому випадку в поле слід надати рядок із нульовим символом у форматі PEM **certificate**.

Отримати сертифікат із сервера, наприклад: **mqtt.eclipseprojects.io**

```
openssl s_client -showcerts -connect mqtt.eclipseprojects.io:8883 <
/dev/null \
2> /dev/null | openssl x509 -outform PEM > mqtt_eclipse_org.pem
```

- Перевірте приклад програми: protocols/mqtt/ssl
- Конфігурація:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .broker = {
      .address.uri = "mqtts://mqtt.eclipseprojects.io:8883",
      .verification.certificate = (const char
*)mqtt_eclipse_org_pem_start,
    },
};
```

Облікові дані клієнта

Усі пов’язані з клієнтом облікові дані знаходяться під цим **credentials** полем.

- **username**: покажчик на ім’я користувача, яке використовується для підключення до брокера, також може бути встановлено URI
- **client_id**: вказівник на ідентифікатор клієнта, за замовчуванням ESP32_%CHIPID% – %CHIPID% останні 3 байти MAC-адреси в шістнадцятковому форматі

Аутентифікація

- Через поле можна задати параметри аутентифікації **authentication**. Клієнт підтримує такі методи автентифікації:
- **password**: використовувати пароль, встановивши
- **certificate** і **key**: взаємна автентифікація за допомогою TLS, і обидва можуть бути надані у форматі PEM або DER
- **use_secure_element**: використовуйте захищений елемент (ATECC608A), підключений до ESP32
- **ds_data**: використовуйте периферійний пристрій цифрового підпису, доступний у деяких пристроях Espressif

СЕСІЯ MQTT

Для конфігурацій, пов’язаних із сеансом MQTT, **session** слід використовувати поля.

Остання воля і заповіт

MQTT дозволяє надсилати повідомлення про останню волю та заповіт (LWT), щоб сповіщати інших клієнтів, коли клієнт невигідно від’єднується. Це налаштовується за допомогою наступних полів у **last_will** структурі.

- **topic** : покажчик на тему повідомлення LWT
- **msg** : покажчик на повідомлення LWT
- **msg_len** : довжина повідомлення LWT, необхідна, якщо **msg** воно не завершується нулем
- **qos** : якість обслуговування для повідомлення LWT
- **retain** : вказує прапор збереження повідомлення LWT

Параметри в меню конфігурації проекту

Доступні такі налаштування:

- CONFIG_MQTT_PROTOCOL_311 : увімкнути версію 3.1.1 протоколу MQTT
- CONFIG_MQTT_TRANSPORT_SSL і CONFIG_MQTT_TRANSPORT_ WEBSOCKET : увімкнути певний транспортний рівень MQTT, наприклад SSL, WEBSOCKET і WEBSOCKET_SECURE
- CONFIG_MQTT_CUSTOM_OUTBOX : вимкнути реалізацію за замовчуванням mqtt_outbox, щоб можна було надати конкретну реалізацію

Події

Клієнт MQTT може публікувати такі події:

- `MQTT_EVENT_BEFORE_CONNECT`: Клієнт ініціалізовано та збирається почати підключення до брокера.
- `MQTT_EVENT_CONNECTED`: Клієнт успішно встановив з'єднання з брокером. Тепер клієнт готовий надсилати й отримувати дані.
- `MQTT_EVENT_DISCONNECTED`: Клієнт перервав з’єднання через те, що не може прочитати або записати дані, наприклад, через те, що сервер недоступний.
- `MQTT_EVENT_SUBSCRIBED`: Брокер підтвердив запит клієнта на підписку. Дані події містять ідентифікатор повідомлення про підписку.
- `MQTT_EVENT_UNSUBSCRIBED`: Брокер підтвердив запит клієнта на відмову від підписки. Дані події містять ідентифікатор повідомлення про відмову від підписки.
- `MQTT_EVENT_PUBLISHED`: брокер підтвердив публікацію повідомлення клієнта. Це публікується лише для QoS рівнів 1 і 2, оскільки рівень 0 не використовує підтвердження. Дані події містять ідентифікатор повідомлення публікації.
- `MQTT_EVENT_DATA`: Клієнт отримав повідомлення про публікацію. Дані події містять: ідентифікатор повідомлення, назву теми, у якій воно було опубліковано, отримані дані та їх довжину. Для даних, які перевищують внутрішній буфер, `MQTT_EVENT_DATA` публікується кілька подій, **`current_data_offset`** а **`total_data_len`** дані подій оновлюються, щоб відстежувати фрагментоване повідомлення.
- `MQTT_EVENT_ERROR`: Клієнт зіткнувся з помилкою. Поле **`error_handle`** в даних події містить **`error_type`** дані.

ДОВІДНИК API

Файл заголовка

```
#include "mqtt_client.h"
```

Цей файл заголовка є частиною API, що надається компонентом `mqtt`. Щоб оголосити, що ваш компонент залежить від `mqtt`, додайте наступне до свого CMakeLists.txt:

REQUIRES mqtt

або

PRIV_REQUIRES mqtt

Функції

esp_mqtt_client_handle_t esp_mqtt_client_init (const esp_mqtt_client_config_t * config)

Створює дескриптор клієнта *MQTT* на основі конфігурації.

Параметри

config -- структура конфігурації *MQTT*

Повернення

mqtt_client_handle у разі успішного створення, NULL у разі помилки

esp_err_t esp_mqtt_client_set_uri (клієнт esp_mqtt_client_handle_t , const char * uri)

Встановлює URI підключення *MQTT* . Цей API зазвичай використовується для заміни URI, налаштованого в esp_mqtt_client_init.

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **uri** --

Повернення

ESP_FAIL, якщо помилка синтаксичного аналізу URI, ESP_OK в разі успіху

esp_err_t esp_mqtt_client_start (клієнт esp_mqtt_client_handle_t)

Запускає клієнт *MQTT* із уже створеним дескриптором клієнта.

Параметри

клієнт -- дескриптор клієнта *MQTT*

Повернення

ESP_OK в разі успіху ESP_ERR_INVALID_ARG в разі неправильної ініціалізації ESP_FAIL в разі іншої помилки

esp_err_t esp_mqtt_client_reconnect (клієнт esp_mqtt_client_handle_t)

Цей API зазвичай використовується для примусового повторного підключення після певної події.

Параметри

клієнт -- дескриптор клієнта *MQTT*

Повернення

ESP_OK у разі успіху ESP_ERR_INVALID_ARG у разі неправильної ініціалізації ESP_FAIL, якщо клієнт перебуває в недійсному стані

esp_err_t esp_mqtt_client_disconnect (клієнт esp_mqtt_client_handle_t)

Цей API зазвичай використовується для примусового відключення від брокера.

Параметри

клієнт -- дескриптор клієнта *MQTT*

Повернення

ESP_OK у разі успіху ESP_ERR_INVALID_ARG у разі неправильної ініціалізації

esp_err_t esp_mqtt_client_stop (клієнт esp_mqtt_client_handle_t)

Зупиняє клієнтські завдання *MQTT* .

- Примітки:
- Неможливо викликати з обробника подій *MQTT*

Параметри

клієнт -- дескриптор клієнта *MQTT*

Повернення

ESP_OK у разі успіху ESP_ERR_INVALID_ARG у разі неправильної ініціалізації ESP_FAIL, якщо клієнт перебуває в недійсному стані

int esp_mqtt_client_subscribe_single (клієнт esp_mqtt_client_handle_t , const char * тема , int qos)

Підпишіться клієнта на визначену тему з визначеним qos.

Примітки:

- Щоб надіслати повідомлення про підписку, клієнт повинен бути підключений
- Цей API можна виконати із завдання користувача або зворотного виклику події *MQTT , тобто внутрішнього завдання MQTT* (API захищено внутрішнім м’ютексом, тому він може заблокуватися, якщо триває триваліша операція отримання даних.
- esp_mqtt_client_subscribe можна використовувати для виклику цієї функції.

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **тема** -- фільтр тем для підписки
- **qos** -- максимальний рівень qos підписки

Повернення

message_id повідомлення про підписку про успіх -1 про невдачу -2 у разі повної вихідної папки.

int esp_mqtt_client_subscribe_multiple (клієнт esp_mqtt_client_handle_t , const esp_mqtt_topic_t * topic_list , int size)

Підпишіться клієнта на список визначених тем із визначеними qos.

Примітки:

- Щоб надіслати повідомлення про підписку, клієнт повинен бути підключений
- Цей API можна виконати із завдання користувача або зворотного виклику події *MQTT , тобто внутрішнього завдання MQTT* (API захищено внутрішнім м'ютексом, тому він може заблокуватися, якщо триває триваліша операція отримання даних.
- esp_mqtt_client_subscribe можна використовувати для виклику цієї функції.

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **topic_list** -- Список тем для підписки
- **розмір** -- розмір списку_тем

Повернення

message_id повідомлення про підписку про успіх -1 про невдачу -2 у разі повної вихідної папки.

int esp_mqtt_client_unsubscribe (клієнт esp_mqtt_client_handle_t , const char * тема)

Скасувати підписку клієнта на визначену тему.

Примітки:

- Щоб надіслати повідомлення про відмову від підписки, необхідно підключити клієнта
- Це потоково безпечно, будь ласка, зверніться до esp_mqtt_client_subscribe_single деталей

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **тема** --

Повернення

message_id повідомлення про підписку в разі успіху -1 у разі невдачі

int esp_mqtt_client_publish (esp_mqtt_client_handle_t клієнт , const char * тема , const char * дані , int len , int qos , int retain)

Клієнт надсилає повідомлення про публікацію брокеру.

Примітки:

- Цей API може заблокуватися на кілька секунд або через тайм-аут мережі (10 с), або якщо публікація корисних даних перевищує внутрішній буфер (через фрагментацію повідомлення)
- Клієнт не повинен бути підключений, щоб цей API працював, ставлячи повідомлення в чергу з qos>1 (повертаючи -1 для всіх повідомлень qos=0, якщо відключено). Якщо MQTT_SKIP_PUBLISH_IF_ DISCONNECTED увімкнено, цей API не намагатиметься опублікувати, коли клієнт не підключено, і завжди повертатиме -1.
- Це потоково безпечно, будь ласка, зверніться до esp_mqtt_client_subscribe деталей

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **topic** -- рядок теми
- **дані** -- рядок корисного навантаження (встановлено на NULL, надсилання порожнього повідомлення корисного навантаження)
- **len** -- довжина даних, якщо встановлено значення 0, довжина обчислюється з рядка корисного навантаження
- **qos** -- QoS опублікованого повідомлення
- **retain** -- прапор збереження

Повернення

message_id повідомлення публікації (для QoS 0 message_id завжди дорівнюватиме нулю) у разі успіху. -1 у разі невдачі, -2 у разі повної вихідної папки.

int esp_mqtt_client_enqueue (клієнт esp_mqtt_client_handle_t , const char * тема , const char * дані , int len , int qos , int retain , bool store)

Додає повідомлення до папки «Вихідні», щоб надіслати його пізніше. Зазвичай використовується для повідомлень із qos>0, але також може використовуватися для повідомлень qos=0, якщо store=true.

Цей API генерує та зберігає повідомлення публікації у внутрішній вихідній папці, а фактичне надсилання в мережу виконується в контексті завдання mqtt (на відміну від esp_mqtt_client_publish(), який надсилає повідомлення публікації негайно в контексті завдання користувача).

Таким чином, його можна використовувати як неблокуючу версію esp_mqtt_client_publish().

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **topic** -- рядок теми
- **дані** -- рядок корисного навантаження (встановлено на NULL, надсилання порожнього повідомлення корисного навантаження)
- **len** -- довжина даних, якщо встановлено значення 0, довжина обчислюється з рядка корисного навантаження
- **qos** -- QoS опублікованого повідомлення
- **retain** -- прапор збереження
- **store** -- якщо істина, усі повідомлення ставляться в чергу; інакше лише QoS 1 і QoS 2 ставляться в чергу

Повернення

message_id, якщо поставлено в чергу успішно, -1 у разі помилки, -2 у разі повної вихідної папки.

esp_err_t esp_mqtt_client_destroy (клієнт esp_mqtt_client_handle_t)

Знищує дескриптор клієнта.

Примітки:

- Неможливо викликати з обробника подій *MQTT*

Параметри

клієнт -- дескриптор клієнта *MQTT*

Повернення

ESP_OK ESP_ERR_INVALID_ARG через неправильну ініціалізацію

esp_err_t esp_mqtt_set_config (клієнт esp_mqtt_client_handle_t , const esp_mqtt_client_config_t * конфігурація)

Встановіть структуру конфігурації, яка зазвичай використовується під час оновлення конфігурації (тобто під час події "before_connect".

Примітки:

- Під час виклику цієї функції переконайтеся, що встановлено всі заплановані конфігурації, інакше буде встановлено значення за замовчуванням.

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **config** -- структура конфігурації *MQTT*

Повернення

ESP_ERR_NO_MEM, якщо не вдалося виділити ESP_ERR_INVALID_ARG, якщо конфліктує конфігурація транспорту. ESP_OK в разі успіху

esp_err_t esp_mqtt_client_register_event (esp_mqtt_client_handle_t клієнт , esp_mqtt_event_id_t подія , esp_event_handler_t event_handler , void * event_handler_arg)

Реєструє подію *MQTT* .

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **event** -- тип події
- **event_handler** -- зворотний виклик обробника
- **event_handler_arg** -- контекст обробників

Повернення

ESP_ERR_NO_MEM, якщо не вдалося виділити ESP_ERR_INVALID_ARG у разі неправильної ініціалізації ESP_OK у разі успіху

esp_err_t esp_mqtt_client_unregister_event (клієнт esp_mqtt_client_handle_t , подія esp_mqtt_event_id_t , обробник події esp_event_handler_t)

Скасовує реєстрацію події mqtt.

Параметри

- **клієнт** -- дескриптор клієнта mqtt
- **подія** -- ID події
- **event_handler** -- обробник для скасування реєстрації

Повернення

ESP_ERR_NO_MEM, якщо не вдалося виділити ESP_ERR_INVALID_ARG для недійсного ідентифікатора події ESP_OK, якщо вдалося

int esp_mqtt_client_get_outbox_size (клієнт esp_mqtt_client_handle_t)

Отримати розмір вихідної папки.

Параметри

клієнт -- дескриптор клієнта *MQTT*

Повернення

вихідний розмір 0 при неправильній ініціалізації

esp_err_t esp_mqtt_dispatch_custom_event (клієнт esp_mqtt_client_handle_t , подія esp_mqtt_event_t *)

Надсилати подію користувача у внутрішній цикл подій mqtt.

Параметри

- **клієнт** -- дескриптор клієнта *MQTT*
- **подія** -- структура опису події *MQTT*

Повернення

ESP_OK у разі успіху ESP_ERR_TIMEOUT, якщо подію не вдалося поставити в чергу (див. також CONFIG_MQTT_EVENT_QUEUE_SIZE)

Структури

struct esp_mqtt_error_codes

Структура коду помилки MQTT , яка передається як контекстна інформація в подію ERROR

Важливо: ця структура розширює esp_tls_last_error структуру помилок і є зворотно сумісною з нею (тому може бути приведена до нижньої версії та розглядатися як esp_tls_last_error помилка, але рекомендовано оновлювати програми, якщо раніше використовувалося таким чином)

Використовуйте цю структуру, спочатку безпосередньо перевіряючи error_type, а потім відповідний код помилки залежно від джерела помилки:

MQTT_ERROR_TYPE_TCP_

Про помилку повідомляє MQTT_ERROR_TYPE_CONNECTION_REFUSED

|

Public члени

esp_err_t esp_tls_last_esp_err

останній код esp_err, отриманий від компонента esp-tls

int esp_tls_stack_err

Спеціальний код помилки tls повідомляється з базового стека tls

int esp_tls_cert_verify_flags

позначки tls, отримані з основного стеку tls під час перевірки сертифіката

esp_mqtt_error_type_t error_type

тип помилки з посиланням на джерело помилки

esp_mqtt_connect_return_code_t connect_return_code

У з'єднанні відмовлено, код помилки повідомляє брокер MQTT* під час з'єднання

int esp_transport_sock_errno

errno з базового сокета

struct esp_mqtt_event_t

Структура конфігурації події *MQTT*

Public члени

esp_mqtt_event_id_t ідентифікатор події

Тип події *MQTT*

esp_mqtt_client_handle_t клієнт

Дескриптор клієнта MQTT для цієї події

char * дані

Дані, пов'язані з цією подією

int data_len

Довжина даних для цієї події

int total_data_len

Загальна довжина даних (довші дані надаються з кількома подіями)

int current_data_offset

Фактичний зсув для даних, пов'язаних із цією подією

char * тема

Тема, пов'язана з цією подією

int topic_len

Тривалість теми для цієї події, пов'язаної з цією подією

int msg_id

Ідентифікатор повідомлення *MQTT*

int session_present

Прапор MQTT session_present для події підключення

esp_mqtt_error_codes_t * error_handle

дескриптор помилки esp-mqtt, включаючи помилки esp-tls, а також внутрішні помилки *MQTT*

bool зберегти

Зберігається позначка повідомлення, пов'язаного з цією подією

int qos

QoS повідомлень, пов'язаних із цією подією

bool dup

прапор дублювання повідомлення, пов'язаного з цією подією

esp_mqtt_protocol_ver_t protocol_ver

Версія протоколу MQTT, яка використовується для з'єднання, за замовчуванням значення з menuconfig

struct esp_mqtt_client_config_t

Структура конфігурації клієнта *MQTT*

- Значення за замовчуванням можна встановити через menuconfig
- Усі сертифікати та ключові дані можуть бути передані у форматі PEM або DER. Формат PEM повинен містити кінцевий символ NULL і пов'язане поле len має значення 0. Формат DER вимагає пов'язаного поля len із правильною довжиною.

Public члени

***struct* esp_mqtt_client_config_t :: broker_t брокер**

Адреса брокера та перевірка безпеки

***struct* esp_mqtt_client_config_t :: credentials_t облікові дані**

Облікові дані користувача для брокера

***struct* esp_mqtt_client_config_t :: session_t session**

Конфігурація сесії *MQTT* .

***struct* esp_mqtt_client_config_t :: network_t мережа**

Конфігурація мережі

***struct* esp_mqtt_client_config_t :: task_t завдання**

Конфігурація завдання FreeRTOS.

***struct* esp_mqtt_client_config_t :: buffer_t буфер**

Конфігурація розміру буфера.

***struct* esp_mqtt_client_config_t :: outbox_config_t outbox**

Конфігурація вихідної скриньки.

***struct* broker_t**

Конфігурація, пов'язана з брокером

Public члени

***struct* esp_mqtt_client_config_t :: broker_t :: address_t адреса**

Конфігурація адреси брокера

***struct* esp_mqtt_client_config_t :: broker_t :: verification_t перевірка**

Перевірка безпеки брокера

***struct* address_t**

Адреса брокера

- uri мають пріоритет над іншими полями
- Якщо uri не встановлено, принаймні ім’я хоста, транспорт і порт мають бути встановлені.

Public члени

***const* char * uri**

Повний URI брокера *MQTT*

***const* char * ім'я хоста**

Ім’я хосту, щоб встановити ipv4, передайте його як рядок)

esp_mqtt_transport_t транспорт

Вибирає транспорт

***const* char * шлях**

Шлях в URI

порт uint32_t

Порт сервера *MQTT*

***struct* verification_t**

Підтвердження особи брокера

Якщо поля не налаштовані, особу брокера не перевірено. Рекомендується встановити параметри в цій структурі з міркувань безпеки.

Public члени

bool use_global_ca_store

Використовуйте глобальний ca_store, подробиці дивіться в документації esp-tls.

esp_err_t (* crt_bundle_attach) (void * conf)

Покажчик на функцію прикріплення ESP x509 Certificate Bundle для використання пакетів сертифікатів. Клієнт лише прикріплює комплект, очищення має виконувати користувач.

***const* char * сертифікат**

Дані сертифіката, за умовчанням NULL. Він не скопійований і не звільнений клієнтом, користувачеві потрібно очистити.

size_t сертифікат_len

Довжина буфера, на яку вказує сертифікат.

***const struct* psk_key_hint * psk_hint_key**

Покажчик на структуру PSK, визначену в esp_tls.h, щоб увімкнути автентифікацію PSK (як альтернативу перевірці сертифіката). PSK вмикається, лише якщо немає інших способів верифікації брокера. Він не скопійований і не звільнений клієнтом, користувачеві потрібно очистити.

bool skip_cert_common_name_check

Пропустіть будь-яку перевірку поля CN сертифіката сервера, це знижує безпеку TLS і робить клієнт *MQTT* сприйнятливим до атак MITM

***const* char * * alpn_protos**

Список підтримуваних протоколів додатків для використання для ALPN із закінченням NULL.

***const* char * загальне_ім'я**

Покажчик на рядок із загальною назвою сертифіката сервера. Якщо не NULL, сертифікат CN сервера має відповідати цьому імені. Якщо NULL, сертифікат CN сервера має відповідати імені хоста. Це ігнорується, якщо skip_cert_common_name_check=true. Він не скопійований і не звільнений клієнтом, користувачеві потрібно очистити.

***struct* buffer_t**

Конфігурація розміру буфера клієнта

Клієнт має два буфери для введення та виведення відповідно.

Public члени

внутрішній розмір

розмір буфера надсилання/отримання *MQTT*

int out_size

розмір вихідного буфера *MQTT* . Якщо не визначено, за замовчуванням використовується розмір, визначений buffer_size

***struct* credentials_t**

Облікові дані клієнта для автентифікації.

Public члени

***const* char * ім'я користувача**

Ім'я користувача *MQTT*

***const* char * client_id**

Встановити ідентифікатор клієнта *MQTT* . Ігнорується, якщо set_null_client_id == true Якщо NULL, установіть ідентифікатор клієнта за умовчанням. Ідентифікатор клієнта за замовчуванням – це ESP32_CHIPID% останні CHIPID% 3 байти MAC-адреси в шістнадцятковому форматі

bool set_null_client_id

Вибирає NULL ідентифікатор клієнта

***struct* esp_mqtt_client_config_t :: credentials_t :: authentication_t автентифікація**

Аутентифікація клієнта

***struct* authentication_t**

Аутентифікація клієнта

Поля, пов’язані з автентифікацією клієнта брокером

Для взаємної автентифікації за допомогою TLS користувач може вибрати сертифікат і ключ, захищений елемент або периферійний пристрій цифрового підпису, якщо доступний.

Public члени

***const* char * пароль**

Пароль *MQTT*

***const* char * сертифікат**

Сертифікат для взаємної автентифікації SSL, не потрібен, якщо взаємна автентифікація не потрібна. Має бути забезпечено `key`. Він не скопійований і не звільнений клієнтом, користувачеві потрібно очистити.

size_t сертифікат_len

Довжина буфера, на яку вказує сертифікат.

***const* char * ключ**

Закритий ключ для взаємної автентифікації SSL, не потрібен, якщо взаємна автентифікація не потрібна. Якщо він не NULL, його також `certificate` потрібно вказати. Він не скопійований і не звільнений клієнтом, користувачеві потрібно очистити.

size_t key_len

Довжина буфера, на яку вказує ключ.

***const* char * ключ_пароль**

Пароль дешифрування ключа клієнта, а не PEM і DER, якщо він надається, `key_password_len` має бути встановлено правильно.

> **int key_password_len**

Довжина пароля, на який вказує `key_password`

> **bool use_secure_element**

Увімкніть безпечний елемент, доступний у ESP32-ROOM-32SE, для підключення SSL

> **void * ds_data**

Носій ручки для параметрів цифрового підпису, периферійний пристрій цифрового підпису доступний у деяких пристроях Espressif. Він не скопійований і не звільнений клієнтом, користувачеві потрібно очистити.

> ***struct* network_t**

Конфігурація мережі

Public члени

> **int reconnect_timeout_ms**

Повторне підключення до посередника після цього значення в мілісекундах, якщо автоматичне повторне підключення не вимкнено (за замовчуванням 10 с)

> **int timeout_ms**

Перервати мережеву операцію, якщо вона не завершена після цього значення в мілісекундах (за замовчуванням 10 с).

> **int refresh_connection_after_ms**

Оновити з’єднання після цього значення (у мілісекундах)

> **bool disable_auto_reconnect**

Клієнт повторно підключиться до сервера (у разі помилок/відключення). Встановіть `disable_auto_reconnect=true` для відключення

> **esp_transport_handle_t транспорт**

Попередження: транспорт має бути дійсним протягом життя клієнта та знищується під час виклику esp_mqtt_client_destroy.

> ***struct* ifreq * if_name**

Назва інтерфейсу для проходження даних. Використовуйте стандартний інтерфейс без налаштування

> ***struct* outbox_config_t**

Параметри конфігурації вихідної скриньки клієнта.

Public члени

> **обмеження uint64_t**

Обмеження розміру вихідної папки в байтах.

> ***struct* session_t**

Конфігурація, пов’язана з сеансом *MQTT*

Public члени

> ***struct* esp_mqtt_client_config_t :: session_t :: last_will_t last_will**

Остання конфігурація

> **bool disable_clean_session**

Чистий сеанс *MQTT , за умовчанням clean_session має значення true*

> **int keepalive**

MQTT keepalive, за замовчуванням - 120 секунд. Налаштовуючи це значення, майте на увазі, що клієнт намагається зв’язатися з посередником через половину інтервалу, який фактично встановлений. Цей консервативний підхід дозволяє робити більше спроб до того, як настане тайм-аут брокера

> **bool disable_keepalive**

Встановлено `disable_keepalive=true` для вимкнення механізму підтримки активності, функція підтримки активності активна за замовчуванням. : встановлення значення конфігурації `keepalive` не `0` вимикає функцію підтримки активності, але використовує стандартний період підтримки активності

> **esp_mqtt_protocol_ver_t protocol_ver**

Версія протоколу MQTT, яка використовується для підключення.

int message_retransmit_timeout

тайм-аут для повторної передачі невдалого пакета

***struct* last_will_t**

Конфігурація повідомлення Last Will and Testament.

Public члени

***const* char * тема**

Тема повідомлення LWT (Last Will and Testament).

***const* char * msg**

Повідомлення LWT, може завершуватися значенням NULL

int msg_len

Довжина повідомлення LWT, якщо повідомлення не завершується NULL, має мати правильну довжину

int qos

QoS повідомлення LWT

int retain

Прапор збереженого повідомлення LWT

***struct* task_t**

Конфігурація завдання клієнта

Public члени

int пріоритет

Пріоритет завдання *MQTT*

int stack_size

Розмір стека завдань *MQTT*

struct topic_t

Структура визначення теми

Public члени

***const* char * фільтр**

Фільтр тем для підписки

int qos

Максимальний рівень QoS підписки

Макроси

MQTT_ERROR_TYPE_ESP_TLS

Тип помилки MQTT_ERROR_TYPE_TCP_TRANSPORT містить усі види помилок транспортного рівня, включаючи помилку ESP-TLS, але раніше повідомлялося лише про помилки рівня MQTT_ERROR_TYPE_ESP_TLS, тому тип помилки ESP-TLS перевизначено тут для зворотної сумісності

esp_mqtt_client_subscribe (client_handle , topic_type , qos_or_size)

Зручний макрос для вибору функції підписки для використання.

Примітки:

- Використання `esp_mqtt_client_subscribe_single` так само, як і попередній esp_mqtt_client_subscribe, зверніться до нього для отримання детальної інформації.

Параметри

- **client_handle** -- дескриптор клієнта *MQTT*
- **topic_type** -- має бути char* для однієї підписки або `esp_mqtt_topic_t` для кількох тем
- **qos_or_size** – це або qos під час підписки на одну тему, або розмір масиву підписки під час підписки на кілька тем.

Повернення

message_id повідомлення про підписку про успіх -1 про невдачу -2 у разі повної вихідної папки.

Визначення типів

typedef struct esp_mqtt_client * esp_mqtt_client_handle_t

typedef enum esp_mqtt_event_id_t esp_mqtt_event_id_t

Типи подій *MQTT* .

Обробник подій користувача отримує дані контексту у esp_mqtt_event_t структурі з

- клієнт - дескриптор клієнта *MQTT*
- різні інші дані залежно від типу події

typedef enum esp_mqtt_connect_return_code_t esp_mqtt_connect_return_code_t

Коди помилок підключення *MQTT поширюються через подію ERROR*

typedef enum esp_mqtt_error_type_t esp_mqtt_error_type_t

Коди помилок підключення *MQTT поширюються через подію ERROR*

typedef enum esp_mqtt_transport_t esp_mqtt_transport_t

typedef enum esp_mqtt_protocol_ver_t esp_mqtt_protocol_ver_t

Версія протоколу MQTT , яка використовується для підключення

typedef struct esp_mqtt_error_codes esp_mqtt_error_codes_t

Структура коду помилки MQTT, яка передається як контекстна інформація в подію ERROR

Важливо: ця структура розширює esp_tls_last_error структуру помилок і є зворотно сумісною з нею (тому може бути приведена до нижньої версії та розглядатися як esp_tls_last_error помилка, але рекомендовано оновлювати програми, якщо раніше використовувалося таким чином)

Використовуйте цю структуру, спочатку безпосередньо перевіряючи error_type, а потім відповідний код помилки залежно від джерела помилки:

тип_помилки	пов'язані змінні члена		
MQTT_ERROR_TYPE_TCP_TRANSPORT	esp_tls_last_esp_err, esp_tls_stack_err, esp_tls_cert_verify_flags, sock_errno	Про помилку повідомляє tcp_transport/esp-tls	
MQTT_ERROR_TYPE_CONNECTION_REFUSED	підключення_повернений_код	Повідомлення про внутрішню помилку брокера *MQTT* під час підключення	

typedef struct esp_mqtt_event_t esp_mqtt_event_t

Структура конфігурації події *MQTT*

typedef esp_mqtt_event_t * esp_mqtt_event_handle_t

typedef struct esp_mqtt_client_config_t esp_mqtt_client_config_t

Структура конфігурації клієнта *MQTT*

- Значення за замовчуванням можна встановити через menuconfig
- Усі сертифікати та ключові дані можуть бути передані у форматі PEM або DER. Формат PEM повинен містити кінцевий символ NULL і пов’язане поле len має значення 0. Формат DER вимагає пов’язаного поля len із правильною довжиною.

typedef struct topic_t esp_mqtt_topic_t

Структура визначення теми

Перерахування

enum esp_mqtt_event_id_t

Типи подій *MQTT* .

Обробник подій користувача отримує дані контексту в `esp_mqtt_event_t` структурі з

- клієнт - дескриптор клієнта *MQTT*
- різні інші дані залежно від типу події

Значення:

> ***enumerator* MQTT_EVENT_ANY**
>
> ***enumerator* MQTT_EVENT_ERROR**

у разі помилки, додатковий контекст: код повернення з'єднання, опис помилки з esp_tls (якщо підтримується)

> ***enumerator* MQTT_EVENT_CONNECTED**

підключена подія, додатковий контекст: прапорець session_present

> ***enumerator* MQTT_EVENT_DISCONNECTED**

відключена подія

> ***enumerator* MQTT_EVENT_SUBSCRIBED**

підписана подія, додатковий контекст:

- msg_id ідентифікатор повідомлення
- error_handle `error_type` на випадок помилки підписки
- покажчик даних на відповідь брокера, перевірка на наявність помилок.
- data_len довжина даних для цієї події

> ***enumerator* MQTT_EVENT_UNSUBSCRIBED**

подія про відмову від підписки, додатковий контекст: msg_id

> ***enumerator* MQTT_EVENT_PUBLISHED**

опублікована подія, додатковий контекст: msg_id

> ***enumerator* MQTT_EVENT_DATA**

подія даних, додатковий контекст:

- msg_id ідентифікатор повідомлення
- покажчик теми на отриману тему
- topic_len довжина теми
- покажчик даних на отримані дані

- data_len довжина даних для цієї події
- current_data_offset зміщення поточних даних для цієї події
- total_data_len загальна довжина отриманих даних
- зберегти прапорець збереження повідомлення
- qos Рівень QoS повідомлення
- dup dup прапор повідомлення. : кілька MQTT_EVENT_DATA можуть бути запущені для одного повідомлення, якщо воно довше внутрішнього буфера. У цьому випадку лише перша подія містить вказівник теми та довжину, інші містять лише дані з поточною довжиною даних і поточним оновленням зміщення даних.

> ***enumerator* MQTT_EVENT_BEFORE_CONNECT**

Подія відбувається перед підключенням

> ***enumerator* MQTT_EVENT_DELETED**

Сповіщення про видалення одного повідомлення з внутрішньої папки вихідних, якщо повідомлення не можна було надіслати та підтвердити до закінчення терміну, визначеного в OUTBOX_EXPIRED_TIMEOUT_MS. (події не публікуються після видалення успішно підтверджених повідомлень)

- Цей ідентифікатор події публікується, лише якщо MQTT_REPORT_DELETED_MESSAGES==1
- Додатковий контекст: msg_id (ідентифікатор видаленого повідомлення).

> ***enumerator* MQTT_USER_EVENT**

Спеціальна подія, яка використовується для постановки завдань у чергу в обробнику подій mqtt. Усі поля типу esp_mqtt_event_t можна використовувати для передачі додаткових даних контексту обробнику.

enum esp_mqtt_connect_return_code_t

Коди помилок підключення *MQTT поширюються через подію ERROR*

Значення:

***enumerator* MQTT_CONNECTION_ACCEPTED**

Підключення прийнято

***enumerator* MQTT_CONNECTION_REFUSE_PROTOCOL**

Причина відмови у підключенні MQTT : неправильний протокол

***enumerator* MQTT_CONNECTION_REFUSE_ID_REJECTED**

Причина відмови у підключенні MQTT : ID відхилено

***enumerator* MQTT_CONNECTION_REFUSE_SERVER_UNAVAILABLE**

Причина відмови в підключенні *MQTT : сервер недоступний*

***enumerator* MQTT_CONNECTION_REFUSE_BAD_USERNAME**

Причина відмови у підключенні *MQTT : не той користувач*

***enumerator* MQTT_CONNECTION_REFUSE_NOT_AUTHORIZED**

Причина відмови у з'єднанні *MQTT : неправильне ім'я користувача або пароль*

enum esp_mqtt_error_type_t

Коди помилок підключення *MQTT поширюються через подію ERROR*

Значення:

***enumerator* MQTT_ERROR_TYPE_NONE**

***enumerator* MQTT_ERROR_TYPE_TCP_TRANSPORT**

***enumerator* MQTT_ERROR_TYPE_CONNECTION_REFUSED**

***enumerator* MQTT_ERROR_TYPE_SUBSCRIBE_FAILED**

enum esp_mqtt_transport_t

Значення:

***enumerator* MQTT_TRANSPORT_UNKNOWN**

***enumerator* MQTT_TRANSPORT_OVER_TCP**

MQTT через TCP, використовуючи схему: MQTT

***enumerator* MQTT_TRANSPORT_OVER_SSL**

MQTT через SSL, використовуючи схему: MQTTS

> ***enumerator*** **MQTT_TRANSPORT_OVER_WS**

MQTT через Websocket, використовуючи схему:: ws

> ***enumerator*** **MQTT_TRANSPORT_OVER_WSS**

MQTT через Websocket Secure, використовуючи схему: wss

enum esp_mqtt_protocol_ver_t

Версія протоколу MQTT , яка використовується для підключення

Значення:

> ***enumerator*** **MQTT_PROTOCOL_UNDEFINED**
>
> ***enumerator*** **MQTT_PROTOCOL_V_3_1**
>
> ***enumerator*** **MQTT_PROTOCOL_V_3_1_1**
>
> ***enumerator*** **MQTT_PROTOCOL_V_5**

СЕРВЕР HTTP

Огляд

Компонент HTTP Server надає можливість запускати легкий веб-сервер на ESP32. Нижче наведено докладні кроки для використання API, доступного сервером HTTP:

- **httpd_start()**: створює екземпляр HTTP-сервера, виділяє для нього пам’ять/ресурси залежно від зазначеної конфігурації та виводить дескриптор екземпляру сервера. Сервер має як сокет прослуховування (TCP) для HTTP-трафіку, так і сокет керування (UDP) для сигналів керування, які вибираються циклічним способом у циклі завдань сервера. Пріоритет завдання та розмір стека можна налаштувати під час створення екземпляра сервера шляхом передачі httpd_config_tструктури в httpd_start(). TCP-трафік аналізується як HTTP-запити, і, залежно від запитаного URI, викликаються зареєстровані користувачем обробники, які мають надсилати назад пакети відповіді HTTP.
- **httpd_stop()**: Це зупиняє сервер із наданим дескриптором і звільняє будь-яку пов’язану пам’ять/ресурси. Це функція блокування, яка спочатку сигналізує про зупинку завдання сервера, а потім чекає завершення завдання. Під час зупинки завдання закриває всі відкриті підключення, видаляє зареєстровані обробники URI та скидає всі дані контексту сеансу до порожніх.
- **httpd_register_uri_handler()**: Обробник URI реєструється шляхом передачі об’єкта httpd_uri_t структури типу, який має члени, включаючи uri ім’я, method тип (наприклад, HTTPD_GET/HTTPD_POST/HTTPD_PUT тощо), покажчик функції типу та покажчик на дані контексту користувача.

Приклад застосування

```
* Our URI handler function to be called during GET /uri request */
esp_err_t get_handler(httpd_req_t *req)
{
    /* Send a simple response */
    const char resp[] = "URI GET Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
```

```
    return ESP_OK;
}

/* Our URI handler function to be called during POST /uri request */
esp_err_t post_handler(httpd_req_t *req)
{
    /* Destination buffer for content of HTTP POST request.
     * httpd_req_recv() accepts char* only, but content could
     * as well be any binary data (needs type casting).
     * In case of string data, null termination will be absent, and
     * content length would give length of string */
    char content[100];

    /* Truncate if content length larger than the buffer */
    size_t recv_size = MIN(req->content_len, sizeof(content));

    int ret = httpd_req_recv(req, content, recv_size);
    if (ret <= 0) {  /* 0 return value indicates connection closed */
        /* Check if timeout occurred */
        if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
            /* In case of timeout one can choose to retry calling
             * httpd_req_recv(), but to keep it simple, here we
             * respond with an HTTP 408 (Request Timeout) error */
            httpd_resp_send_408(req);
        }
        /* In case of error, returning ESP_FAIL will
         * ensure that the underlying socket is closed */
        return ESP_FAIL;
    }

    /* Send a simple response */
    const char resp[] = "URI POST Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

/* URI handler structure for GET /uri */
httpd_uri_t uri_get = {
    .uri      = "/uri",
    .method   = HTTP_GET,
    .handler  = get_handler,
    .user_ctx = NULL
};

/* URI handler structure for POST /uri */
httpd_uri_t uri_post = {
    .uri      = "/uri",
    .method   = HTTP_POST,
    .handler  = post_handler,
    .user_ctx = NULL
};
```

```
/* Function for starting the webserver */
httpd_handle_t start_webserver(void)
{
    /* Generate default configuration */
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    /* Empty handle to esp_http_server */
    httpd_handle_t server = NULL;

    /* Start the httpd server */
    if (httpd_start(&server, &config) == ESP_OK) {
        /* Register URI handlers */
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }
    /* If server failed to start, handle will be NULL */
    return server;
}

/* Function for stopping the webserver */
void stop_webserver(httpd_handle_t server)
{
    if (server) {
        /* Stop the httpd server */
        httpd_stop(server);
    }
}
```

Постійні підключення

HTTP-сервер має постійні з’єднання, що дозволяє повторно використовувати те саме з’єднання (сеанс) для кількох передач, зберігаючи при цьому контекстно-залежні дані для сеансу. Дані контексту можуть динамічно розподілятися обробником, і в цьому випадку може знадобитися вказати спеціальну функцію для звільнення цих даних, коли з’єднання/сеанс закрито.

Приклад постійних підключень

```
/* Custom function to free context */
void free_ctx_func(void *ctx)
{
    /* Could be something other than free */
    free(ctx);
}

esp_err_t adder_post_handler(httpd_req_t *req)
{
    /* Create session's context if not already available */
    if (! req->sess_ctx) {
        req->sess_ctx = malloc(sizeof(ANY_DATA_TYPE));   /*!< Pointer to
context data */
        req->free_ctx = free_ctx_func;                   /*!< Function to
free context data */
    }

    /* Access context data */
    ANY_DATA_TYPE *ctx_data = (ANY_DATA_TYPE *)req->sess_ctx;

    /* Respond */
    ..............
    ..............
    ..............

    return ESP_OK;
}
```

СЕРВЕР WEBSOCKET

Компонент HTTP-сервера забезпечує підтримку websocket. Функцію websocket можна ввімкнути в menuconfig за допомогою параметра CONFIG_HTTPD_WS_SUPPORT .

Обробка подій

HTTP-сервер ESP має різні події, для яких бібліотека циклу подій може запускати обробник, коли відбувається певна подія. Обробник має бути зареєстрований за допомогою esp_event_handler_register(). Це допомагає в обробці подій для HTTP-сервера ESP.

esp_http_server_event_id_tмістить усі події, які можуть статися для HTTP-сервера ESP.

Очікуваний тип даних для різних подій HTTP-сервера ESP у циклі подій:

- HTTP_SERVER_EVENT_ERROR : `httpd_err_code_t`
- HTTP_SERVER_EVENT_START : `NULL`
- HTTP_SERVER_EVENT_ON_CONNECTED : `int`
- HTTP_SERVER_EVENT_ON_HEADER : `int`
- HTTP_SERVER_EVENT_HEADERS_SENT : `int`
- HTTP_SERVER_EVENT_ON_DATA : `esp_http_server_event_data`
- HTTP_SERVER_EVENT_SENT_DATA : `esp_http_server_event_data`
- HTTP_SERVER_EVENT_DISCONNECTED : `int`
- HTTP_SERVER_EVENT_STOP : `NULL`

ДОВІДНИК API

Файл заголовка

components/esp_http_server/include/esp_http_server.h

Цей файл заголовка можна включити до:

```
#include "esp_http_server.h"
```

Цей файл заголовка є частиною API, що надається компонентом esp_http_server.

Щоб оголосити, що ваш компонент залежить від esp_http_server, додайте наступне до свого CMakeLists.txt:

```
REQUIRES esp_http_server
```

або

```
PRIV_REQUIRES esp_http_server
```

Функції

esp_err_t httpd_start (httpd_handle_t * handle , const httpd_config_t * config)

Запускає веб-сервер.

Створіть екземпляр HTTP-сервера та виділіть для нього пам’ять/ресурси залежно від зазначеної конфігурації.

Приклад використання:

```
//Function for starting the webserver
httpd_handle_t start_webserver(void)
{
// Generate default configuration
httpd_config_t config = HTTPD_DEFAULT_CONFIG();

// Empty handle to http_server
     httpd_handle_t server = NULL;

// Start the httpd server
if (httpd_start(&server, &config) == ESP_OK) {
// Register URI handlers
```

```
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }
// If server failed to start, handle will be NULL
return server;
}
```

Параметри

- **config -- [in]** Конфігурація для нового екземпляра сервера
- **handle -- [out]** дескриптор новоствореного екземпляра сервера. NULL у разі помилки

Повернення

- ESP_OK : екземпляр створено успішно
- ESP_ERR_INVALID_ARG : нульовий аргумент(и)
- ESP_ERR_HTTPD_ALLOC_MEM: не вдалося виділити пам’ять, наприклад
- ESP_ERR_HTTPD_TASK: Не вдалося запустити завдання сервера

esp_err_t httpd_stop (httpd_handle_t дескриптор)

Зупиняє веб-сервер.

Вивільняє пам’ять/ресурси, які використовуються примірником HTTP-сервера, і видаляє їх. Після видалення дескриптор більше не можна використовувати для доступу до екземпляра.

Приклад використання:

```
// Function for stopping the webserver
void stop_webserver(httpd_handle_t server)
{
// Ensure handle is non NULL
if (server != NULL) {
// Stop the httpd server
        httpd_stop(server);
    }
}
```

Параметри

дескриптор -- [in] дескриптор сервера, повернутий httpd_start

Повернення

- ESP_OK : Сервер успішно зупинено
- ESP_ERR_INVALID_ARG : Аргумент дескриптора має значення Null

esp_err_t httpd_register_uri_handler (httpd_handle_t дескриптор , const h ttpd_uri_t * uri_handler)

Реєструє обробник URI.

Приклад використання:

```
esp_err_t my_uri_handler(httpd_req_t* req)
{
// Recv , Process and Send
    ....
    ....
    ....

// Fail condition
if (....) {
// Return fail to close session //
return ESP_FAIL;
    }

// On success
return ESP_OK;
}

// URI handler structure
httpd_uri_t my_uri {
    .uri      = "/my_uri/path/xyz",
    .method   = HTTPD_GET,
    .handler  = my_uri_handler,
    .user_ctx = NULL
};

// Register handler
if (httpd_register_uri_handler(server_handle, &my_uri) != ESP_OK) {
// If failed to register handler
   ....
}
```

Обробники URI можна реєструвати в режимі реального часу, доки дескриптор сервера є дійсним.

Параметри

- **дескриптор** -- **[in]** дескриптор екземпляра сервера HTTPD
- **uri_handler** -- **[in]** покажчик на обробник, який потрібно зареєструвати

Повернення

- ESP_OK : після успішної реєстрації обробника
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_HANDLERS_FULL : якщо не залишилося слотів для нового обробника
- ESP_ERR_HTTPD_HANDLER_EXISTS : якщо обробник із тим самим URI та методом уже зареєстровано

esp_err_t httpd_unregister_uri_handler (httpd_handle_t дескриптор , cons t char * uri , метод httpd_method_t)

Скасувати реєстрацію обробника URI.

Параметри

- **дескриптор** -- **[in]** дескриптор екземпляра сервера HTTPD
- **uri** -- **[в]** рядок URI
- **метод** -- **[в]** метод HTTP

Повернення

- ESP_OK : після успішного скасування реєстрації обробника
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_NOT_FOUND : обробник із зазначеним URI та методом не знайдено

esp_err_t httpd_unregister_uri (httpd_handle_t дескриптор , const char * uri)

Скасувати реєстрацію всіх обробників URI із зазначеним рядком uri.

Параметри

- **дескриптор -- [in]** дескриптор екземпляра сервера HTTPD
- **uri -- [in]** рядок uri, що визначає всі обробники, які потрібно скасувати з реєстрації

Повернення

- ESP_OK : після успішного скасування реєстрації всіх таких обробників
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_NOT_FOUND : Немає зареєстрованого обробника з указаним рядком uri

esp_err_t httpd_sess_set_recv_override (httpd_handle_t hd , int sockfd , ht tpd_recv_func_t recv_func)

Перевизначити функцію отримання веб-сервера (за допомогою FD сеансу) Ця функція замінює функцію отримання веб-сервера. Ця ж функція використовується для читання пакетів запитів HTTP.

Передбачається, що цей API викликається з контекстуAPI сесії http, де sockfd є дійсним параметромобробник URI, де sockfd отримано за допомогою httpd_req_to_sockfd()

Параметри

- **hd -- [in]** дескриптор екземпляра HTTPD
- **sockfd -- [in]** FD сеансового сокета
- **recv_func -- [in]** Функція отримання, яка буде встановлена для цього сеансу

Повернення

- ESP_OK : Перевизначення після успішної реєстрації
- ESP_ERR_INVALID_ARG : Нульові аргументи

esp_err_t httpd_sess_set_send_override (httpd_handle_t hd , int sockfd , ht tpd_send_func_t send_func)

Перевизначити функцію надсилання веб-сервера (за допомогою FD сеансу)

Ця функція замінює функцію надсилання веб-сервера. Ця ж функція використовується для надсилання відповіді на будь-який запит HTTP.

Передбачається, що цей API викликається з контекстуAPI сесії http, де sockfd є дійсним параметромобробник URI, де sockfd отримано за допомогою httpd_req_to_sockfd()

Параметри

- **hd** -- **[in]** дескриптор екземпляра HTTPD
- **sockfd** -- **[in]** FD сеансового сокета
- **send_func** -- **[in]** Функція надсилання, яка буде встановлена для цього сеансу

Повернення

- ESP_OK : Перевизначення після успішної реєстрації
- ESP_ERR_INVALID_ARG : Нульові аргументи

esp_err_t httpd_sess_set_pending_override (httpd_handle_t hd , int sockf, httpd_pending_func_t pending_func)

Перевизначити функцію очікування веб-сервера (за допомогою FD сеансу)

Ця функція замінює функцію очікування веб-сервера. Ця функція використовується для перевірки наявності незавершених байтів у сокеті.

Параметри

- **hd** -- **[in]** дескриптор екземпляра HTTPD
- **sockfd** -- **[in]** FD сеансового сокета
- **pending_func** -- **[in]** Функція отримання, яка буде встановлена для цього сеансу

Повернення

- ESP_OK : Перевизначення після успішної реєстрації
- ESP_ERR_INVALID_ARG : Нульові аргументи

esp_err_t httpd_req_async_handler_begin (httpd_req_t * r , httpd_req_t * * вихід)

Почніть асинхронний запит. Цю функцію можна викликати в обробнику запиту, щоб отримати копію запиту, яку можна використовувати в асинхронному потоці.

Ця функція необхідна для одночасної обробки кількох запитів. Перегляньте examples/async_requests для прикладів використання.

Ви повинні викликати httpd_req_async_handler_complete(), коли завершите роботу над запитом.

Параметри

- **r -- [in]** Запит на створення асинхронної копії
- **out -- [out]** Щойно виділений запит, який можна використовувати в асинхронному потоці

Повернення

- ESP_OK : об’єкт асинхронного запиту створено

esp_err_t httpd_req_async_handler_complete (httpd_req_t * r)

Позначити асинхронний запит як виконаний.

- звільнити пам'ять запиту
- відмовитися від права власності на основний сокет, щоб його можна було використовувати повторно.
- дозволити http-серверу закрити наш сокет за потреби (lru_purge_enable)

Якщо асинхронні запити не позначено як завершені, зрештою сервер більше не прийматиме вхідні з’єднання. Якщо це станеться, сервер зареєструє повідомлення "httpd_accept_conn: помилка в accept (23)".

Параметри

r -- [in] Запит позначити асинхронну роботу як завершену

Повернення

- ESP_OK: асинхронний запит було позначено як виконане

int httpd_req_to_sockfd (httpd_req_t * r)

Отримає дескриптор сокета з HTTP-запиту.

Цей API поверне дескриптор сокета сеансу, для якого було виконано обробник URI після отримання запиту HTTP. Це корисно, коли користувач хоче викликати функції, які вимагають fd сеансового сокета, з обробника URI, тобто. : httpd_sess_get_ctx(), httpd_sess_trigger_close(), httpd_sess_update_lru_counter().

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Параметри

r -- [in] Запит, дескриптор сокета якого потрібно знайти

Повернення

- Дескриптор сокета: дескриптор сокета для цього запиту
- -1: void/NULL покажчик запиту

int httpd_req_recv (httpd_req_t * r , char * buf , size_t buf_len)

API для читання даних вмісту із запиту HTTP.

Цей API читатиме дані вмісту HTTP із запиту HTTP у наданий буфер. Використовуйте content_len, наданий у структурі httpd_req_t, щоб дізнатися довжину даних, які потрібно отримати.

Якщо content_len завеликий для буфера, тоді користувачеві, можливо, доведеться зробити кілька викликів цієї функції, щоразу вибираючи 'buf_len' кількість байтів, тоді як вказівник на дані вмісту внутрішньо збільшується на те саме число.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Якщо повертається помилка, обробник URI повинен повертати помилку. Це гарантує, що помилковий сокет буде закрито та очищено веб-сервером.

Наразі фрагментоване кодування не підтримується

Параметри

- **r -- [in]** Запит, на який відповідає
- **buf -- [in]** Покажчик на буфер, у який будуть зчитані дані
- **buf_len -- [in]** Довжина буфера

Повернення

- Байти: кількість байтів, успішно зчитаних у буфер
- 0: параметр довжини буфера дорівнює нулю / з'єднання закрито одноранговим вузлом
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Час очікування/перерва під час виклику socket recv()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket recv()

size_t httpd_req_get_hdr_value_len (httpd_req_t * r , поле const char *)

Знайти поле в заголовках запиту та отримати довжину рядка його значення.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику API httpd_resp_send() усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Параметри

- **r -- [in]** Запит, на який відповідає
- **field -- [in]** Поле заголовка, у якому буде здійснюватися пошук у запиті

Повернення

- Довжина : якщо поле знайдено в URL-адресі запиту
- Нуль: поле не знайдено / Void запит / Нульові аргументи

esp_err_t httpd_req_get_hdr_value_str (httpd_req_t * r , const char * пол, char * val , size_t val_size)

Отримати рядок значення поля із заголовків запиту.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику API httpd_resp_send() усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Якщо вихідний розмір більший за вхідний, значення скорочується, що супроводжується помилкою скорочення як значення, що повертається.

Використовуйте httpd_req_get_hdr_value_len(), щоб дізнатися правильну довжину буфера

Параметри

- **r -- [in]** Запит, на який відповідає
- **field -- [in]** Поле для пошуку в заголовку
- **val -- [out]** Покажчик на буфер, у який буде скопійовано значення, якщо поле знайдено
- **val_size -- [in]** Розмір буфера користувача "val"

Повернення

- ESP_OK : знайдено поле в заголовку запиту та скопійовано рядок значення
- ESP_ERR_NOT_FOUND : ключ не знайдено
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту HTTP
- ESP_ERR_HTTPD_RESULT_TRUNC : рядок значення скорочено

size_t httpd_req_get_url_query_len (httpd_req_t * r)

Отримайте довжину рядка запиту з URL-адреси запиту.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний

Параметри

r -- [in] Запит, на який відповідає

Повернення

- Довжина : запит міститься в URL-адресі запиту
- Нуль: Запит не знайдено / Нульові аргументи / Void запит

esp_err_t httpd_req_get_url_query_str (httpd_req_t * r , char * buf , size_t buf_len)

Отримати рядок запиту з URL-адреси запиту.

Наразі користувач може отримати повний рядок запиту URL-адреси, але декодування має виконати користувач. Заголовки запитів можна прочитати за допомогою httpd_req_get_hdr_value_str(), щоб дізнатися «тип вмісту» (наприклад, Content-Type: application/x-www-form-urlencoded), а потім потрібно застосувати відповідний алгоритм декодування.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Якщо вихідний розмір більший за вхідний, значення скорочується, що супроводжується помилкою усічення як повертається значення.

Перед викликом цієї функції можна використати httpd_req_get_url_ query_len(), щоб дізнатися довжину рядка запиту заздалегідь і, отже, виділити буфер потрібного розміру (зазвичай довжина рядка запиту + 1 для нульового завершення) для зберігання рядка запиту

Параметри

- **r -- [in]** Запит, на який відповідає
- **buf -- [out]** Покажчик на буфер, у який буде скопійовано рядок запиту (якщо знайдено)
- **buf_len -- [in]** Довжина вихідного буфера

Повернення

- ESP_OK: Запит знайдено в URL-адресі запиту та скопійовано в буфер
- ESP_ERR_NOT_FOUND : Запит не знайдено
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту HTTP
- ESP_ERR_HTTPD_RESULT_TRUNC : рядок запиту скорочено

esp_err_t httpd_query_key_value (const char * qry , const char * ключ , char * val , size_t val_size)

Допоміжна функція для отримання тегу URL-запиту з рядка запиту типу param1=val1 m2=val2.

Компоненти рядка запиту URL-адреси (ключі та значення) не декодуються. Користувач повинен перевірити наявність поля «Content-Type» у заголовках запиту, а потім, залежно від указаного кодування (URLencoded чи іншого), застосувати відповідний алгоритм декодування.

Якщо фактичний розмір значення більший за val_size, тоді значення скорочується, що супроводжується помилкою скорочення як значення, що повертається.

Параметри

- **qry -- [in]** Покажчик на рядок запиту
- **key -- [in]** Ключ, який потрібно шукати в рядку запиту
- **val -- [out]** Покажчик на буфер, у який буде скопійовано значення, якщо ключ знайдено
- **val_size -- [in]** Розмір буфера користувача "val"

Повернення

- ESP_OK : ключ знайдено в рядку запиту URL-адреси та скопійовано в буфер
- ESP_ERR_NOT_FOUND : ключ не знайдено
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESULT_TRUNC : рядок значення скорочено

esp_err_t httpd_req_get_cookie_val (httpd_req_t * req , const char * cookie_name , char * val , size_t * val_size)

Отримайте рядок значення значення cookie із заголовків запиту "Cookie" за назвою cookie.

Параметри

- **req** -- **[in]** Покажчик на запит HTTP
- **cookie_name** -- **[in]** Ім'я cookie, яке буде шукатися в запиті
- **val** -- **[out]** Покажчик на буфер, у який буде скопійовано значення cookie, якщо файл cookie знайдено
- **val_size** -- **[inout]** Покажчик на розмір буфера користувача "val". Ця змінна міститиме довжину файлу cookie, якщо повернено ESP_OK, і необхідну довжину буфера, якщо повернено ESP_ERR_HTTPD_RESULT_TRUNC.

Повернення

- ESP_OK : ключ знайдено в рядку cookie та скопійовано в буфер
- ESP_ERR_NOT_FOUND : ключ не знайдено
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESULT_TRUNC : рядок значення скорочено
- ESP_ERR_NO_MEM : Помилка розподілу пам'яті

bool httpd_uri_match_wildcard (const char * uri_template , const char * uri _to_match , size_t match_upto)

Перевірте, чи відповідає URI заданому шаблону підстановки.

Шаблон може закінчуватися на "?" щоб зробити попередній символ необов'язковим (зазвичай скісною рискою), «*» для відповідності символу узагальнення та «?*», щоб зробити попередній символ необов'язковим, і, якщо він є, дозволити будь-чому, що слідує за ним.

приклад:

- * відповідає всьому
- /foo/? відповідає /foo і /foo/
- /foo/* (без зворотної косої риски) відповідає /foo/ і /foo/bar, але не /foo або /fo
- /foo/?* або /foo/*? (без зворотної косої риски) відповідає /foo/, /foo/bar, а також /foo, але не /foox або /fo

Спеціальні символи "?" і "*" в будь-якому іншому місці шаблону буде сприйматися буквально.

Параметри

- **uri_template** -- **[in]** шаблон URI (шаблон)
- **uri_to_match** -- **[in]** URI для відповідності
- **match_upto** -- **[in]** скільки символів буфера URI для перевірки (може бути кінцевий рядок запиту тощо)

Повернення

true, якщо збіг знайдено

esp_err_t httpd_resp_send (httpd_req_t * r , const char * buf , ssize_t buf)

API для надсилання повної відповіді HTTP.

Цей API надсилатиме дані як відповідь HTTP на запит. Це передбачає, що у вас є готова вся відповідь в одному буфері. Якщо ви бажаєте надіслати відповідь поетапними фрагментами, замість цього використовуйте httpd_resp_send_chunk().

Якщо код статусу та тип вмісту не встановлено, за замовчуванням буде надіслано код стану 200 OK і тип вмісту як текст/html.

Ви можете викликати такі функції перед цим API, щоб налаштувати заголовки відповіді: httpd_resp_set_status() – для встановлення рядка статусу HTTP, httpd_resp_set_type() – для встановлення типу вмісту, httpd_resp_set_hdr() – для додавання будь-яких додаткових записів значень поля у відповідь заголовок

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику цього API на запит отримано відповідь.

Після цього додаткові дані не можуть бути надіслані для запиту.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Параметри

- **r -- [in]** Запит, на який відповідає
- **buf -- [in]** Буфер, звідки має бути отримано вміст
- **buf_len -- [in]** Довжина буфера, HTTPD_RESP_USE_STRLEN для використання strlen()

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульовий покажчик запиту
- ESP_ERR_HTTPD_RESP_HDR : Основні заголовки завеликі для внутрішнього буфера
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void запит

esp_err_t httpd_resp_send_chunk (httpd_req_t * r , const char * buf , ssize_t buf_len)

API для надсилання одного блоку HTTP.

Цей API надсилатиме дані як відповідь HTTP на запит. Цей API використовуватиме фрагментоване кодування та надсилатиме відповідь у формі фрагментів. Якщо у вас є вся відповідь, що міститься в одному буфері, натомість використовуйте httpd_resp_send().

Якщо код статусу та тип вмісту не встановлено, за замовчуванням буде надіслано код стану 200 OK і тип вмісту як текст/html. Ви можете викликати такі функції перед цим API, щоб налаштувати заголовки відповідей httpd_resp_set_status() – для встановлення рядка статусу HTTP, httpd_resp_set_type() – для встановлення типу вмісту, httpd_resp_set_hdr() – для додавання будь-яких додаткових записів значень поля в заголовку відповіді

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Коли ви закінчите надсилати всі ваші фрагменти, ви повинні викликати цю функцію з buf_len як 0.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Параметри

- **r -- [in]** Запит, на який відповідає
- **buf -- [in]** Покажчик на буфер, який зберігає дані
- **buf_len -- [in]** Довжина буфера, HTTPD_RESP_USE_STRLEN для використання strlen()

Повернення

- ESP_OK : після успішного надсилання фрагменту пакета відповіді
- ESP_ERR_INVALID_ARG : Нульовий покажчик запиту
- ESP_ERR_HTTPD_RESP_HDR : Основні заголовки завеликі для внутрішнього буфера
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

static inline esp_err_t httpd_resp_sendstr (httpd_req_t * r , const char * str)

API для надсилання повного рядка як відповіді HTTP.

Цей API просто викликає http_resp_send із довжиною буфера, встановленою на довжину рядка, припускаючи, що буфер містить рядок із нульовим завершенням

Параметри

- **r -- [in]** Запит, на який відповідає
- **str -- [in]** Рядок, який буде надіслано як тіло відповіді

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульовий покажчик запиту
- ESP_ERR_HTTPD_RESP_HDR : Основні заголовки завеликі для внутрішнього буфера
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void запит

static inline esp_err_t httpd_resp_sendstr_chunk (httpd_req_t * r , const char * str)

API для надсилання рядка як блоку відповіді HTTP.

Цей API просто викликає http_resp_send_chunk із довжиною буфера, встановленою на довжину рядка, припускаючи, що буфер містить рядок із нульовим завершенням

Параметри

- **r -- [in]** Запит, на який відповідає
- **str -- [in]** Рядок, який буде надіслано як тіло відповіді (NULL для завершення пакета відповіді)

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульовий покажчик запиту
- ESP_ERR_HTTPD_RESP_HDR : Основні заголовки завеликі для внутрішнього буфера
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void запит

esp_err_t httpd_resp_set_status (httpd_req_t * r , const char * status)

API для встановлення коду статусу HTTP.

Цей API встановлює статус відповіді HTTP на вказане значення. За умовчанням як відповідь надсилається відповідь «200 OK».

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Цей API встановлює лише це значення статусу. Статус не надсилається, доки не буде виконано будь-який із API надсилання.

Переконайтеся, що термін дії рядка стану дійсний до виклику функції надсилання.

Параметри

- **r -- [in]** Запит, на який відповідає
- **status -- [in]** Код статусу HTTP цієї відповіді

Повернення

- ESP_OK : У разі успіху
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

esp_err_t httpd_resp_set_type (httpd_req_t * r , const char * тип)

API для встановлення типу вмісту HTTP.

Цей API встановлює поле "Тип вмісту" відповіді. Тип вмісту за замовчуванням – «text/html».

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Цей API лише встановлює це значення для типу вмісту. Тип не надсилається, доки не буде виконано будь-який із API надсилання.

Переконайтеся, що термін життя рядка типу дійсний до виклику функції надсилання.

Параметри

- **r -- [in]** Запит, на який відповідає
- **type -- [in]** Тип вмісту відповіді

Повернення

- ESP_OK : У разі успіху
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

esp_err_t httpd_resp_set_hdr (httpd_req_t * r , const char * поле , const char * значення)

API для додавання додаткових заголовків.

Цей API встановлює будь-які додаткові поля заголовка, які потрібно надіслати у відповіді.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Заголовок не надсилається, доки не буде виконано будь-який із API надсилання.

Максимально дозволена кількість додаткових заголовків обмежена значенням max_resp_headers у структурі конфігурації.

Переконайтеся, що строки значення поля дійсні до моменту виклику функції надсилання.

Параметри

- **r -- [in]** Запит, на який відповідає
- **field -- [in]** Назва поля заголовка HTTP
- **value -- [in]** Значення цього заголовка HTTP

Повернення

- ESP_OK : Після успішного додавання нового заголовка
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESP_HDR : Загальна кількість додаткових заголовків перевищує максимально дозволений
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

esp_err_t httpd_resp_send_err (httpd_req_t * req , httpd_err_code_t помилка , const char * msg)

Для надсилання коду помилки у відповідь на запит HTTP.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Якщо ви бажаєте надіслати додаткові дані в тілі відповіді, використовуйте функції нижчого рівня безпосередньо.

Параметри

- **req -- [in]** Покажчик на HTTP-запит, для якого потрібно надіслати відповідь
- **error -- [in]** Тип помилки для надсилання
- **msg -- [in]** Рядок повідомлення про помилку (передайте NULL для повідомлення за замовчуванням)

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

esp_err_t httpd_resp_send_custom_err (httpd_req_t * req , const char * status , const char * msg)

Для надсилання спеціального коду помилки у відповідь на запит HTTP.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Якщо ви бажаєте надіслати додаткові дані в тілі відповіді, використовуйте функції нижчого рівня безпосередньо.

Параметри

- **req -- [in]** Покажчик на HTTP-запит, для якого потрібно надіслати відповідь
- **status -- [in]** Статус помилки для надсилання
- **msg -- [in]** Рядок повідомлення про помилку

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

static int esp_err_t httpd_resp_send_404 (httpd_req_t * r)

Допоміжна функція для HTTP 404.

Надіслати повідомлення HTTP 404. Якщо ви бажаєте надіслати додаткові дані в тілі відповіді, використовуйте функції нижчого рівня безпосередньо.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Параметри

r -- [in] Запит, на який відповідає

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

static int esp_err_t httpd_resp_send_408 (httpd_req_t * r)

Допоміжна функція для HTTP 408.

Надіслати повідомлення HTTP 408. Якщо ви бажаєте надіслати додаткові дані в тілі відповіді, використовуйте функції нижчого рівня безпосередньо.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Параметри

r -- [in] Запит, на який відповідає

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

static int esp_err_t httpd_resp_send_500 (httpd_req_t * r)

Допоміжна функція для HTTP 500.

Надіслати повідомлення HTTP 500. Якщо ви бажаєте надіслати додаткові дані в тілі відповіді, використовуйте функції нижчого рівня безпосередньо.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Після виклику цього API усі заголовки запитів очищаються, тому заголовки запитів потрібно скопіювати в окремі буфери, якщо вони знадобляться пізніше.

Параметри

r -- [in] Запит, на який відповідає

Повернення

- ESP_OK : після успішного надсилання пакета відповіді
- ESP_ERR_INVALID_ARG : Нульові аргументи
- ESP_ERR_HTTPD_RESP_SEND : Помилка під час надсилання необроблених даних
- ESP_ERR_HTTPD_INVALID_REQ : Void покажчик запиту

int httpd_send (httpd_req_t * r , const char * buf , size_t buf_len)

Необроблений HTTP-надсилання.

Викличте цей API, якщо ви бажаєте створити власний пакет відповіді. Під час використання цього всі важливі заголовки, наприклад.

Версію HTTP, код статусу, тип і довжину вмісту, кодування тощо потрібно буде створити вручну, а роздільники HTTP (CRLF) потрібно буде правильно розмістити для розділення підрозділів пакета відповіді HTTP.

Якщо встановлено функцію перевизначення надсилання, цей API зрештою викличе цю функцію для надсилання даних.

Передбачається, що цей API викликається лише з контексту обробника URI, де вказівник запиту httpd_req_t* дійсний.

Якщо відповідь не має правильної структури HTTP (яку тепер повинен переконатися користувач), не гарантується, що вона буде розпізнана клієнтом. У більшості випадків вам не потрібно буде викликати цей API, а краще використовувати будь-який із: httpd_resp_send(), httpd_resp_send_chunk()

Параметри

- **r** -- **[in]** Запит, на який відповідає
- **buf** -- **[in]** Буфер, з якого буде зчитано повністю створений пакет
- **buf_len** -- **[in]** Довжина буфера

Повернення

- Байти: кількість успішно надісланих байтів
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Час очікування/перерва під час виклику socket send()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket send()

int httpd_socket_send (httpd_handle_t hd , int sockfd , const char * buf , size_t buf_len , int flags)

API низького рівня для надсилання даних у певний сокет

Це внутрішньо викликає функцію надсилання за замовчуванням або функцію, зареєстровану httpd_sess_set_send_override().

Цей API не рекомендується використовувати в жодному обробнику запитів. Використовуйте це лише для розширених випадків використання, коли деякі асинхронні дані мають надсилатися через сокет.

Параметри

- **hd** -- **[in]** примірник сервера
- **sockfd** -- **[в]** дескриптор файлу сокета сеансу
- **buf** -- **[в]** буфері з байтами для надсилання
- **buf_len** -- **[in]** розмір даних
- **flags** -- **[in]** прапорці для функції send().

Повернення

- Байти: кількість успішно надісланих байтів
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Час очікування/перерва під час виклику socket send()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket send()

int httpd_socket_recv (httpd_handle_t hd , int sockfd , char * buf , size_t buf_len , int flags)

API низького рівня для отримання даних із певного сокета

Це внутрішньо викликає функцію recv за замовчуванням або функцію, зареєстровану за допомогою httpd_sess_set_recv_override().

Цей API не рекомендується використовувати в жодному обробнику запитів. Використовуйте це лише для розширених випадків використання, де потрібне деяке асинхронне спілкування.

Параметри

- **hd** -- **[in]** примірник сервера
- **sockfd** -- **[в]** дескриптор файлу сокета сеансу
- **buf** -- **[в]** буфері з байтами для надсилання
- **buf_len** -- **[in]** розмір даних
- **flags** -- **[in]** прапорці для функції send().

Повернення

- Байти: кількість успішно отриманих байтів
- 0: параметр довжини буфера дорівнює нулю / з'єднання закрито одноранговим вузлом
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Час очікування/перерва під час виклику socket recv()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket recv()

esp_err_t httpd_register_err_handler (httpd_handle_t дескриптор , httpd_err_code_t помилка , httpd_err_handler_func_t handler_fn)

Функція для реєстрації обробників помилок HTTP.

Параметри

- **дескриптор** -- **[в]** дескриптор сервера HTTP
- **error** -- **[in]** Тип помилки
- **handler_fn** -- **[in]** Функція обробника, реалізована користувачем (передайте NULL, щоб скасувати будь-який попередньо встановлений обробник)

Повернення

- ESP_OK : обробник успішно зареєстрований
- ESP_ERR_INVALID_ARG : void код помилки або дескриптор сервера

esp_err_t httpd_queue_work (httpd_handle_t дескриптор , httpd_work_fn _t робота , void * arg)

Виконання функції в черзі в контексті HTTPD.

Цей API ставить робочу функцію в чергу для асинхронного виконання

Деякі протоколи вимагають, щоб веб-сервер генерував деякі асинхронні дані та надсилав їх до постійно відкритого з'єднання. Ця функція призначена для використання такими протоколами.

Параметри

- **дескриптор** -- **[in]** дескриптор сервера, повернутий httpd_start
- **work** -- **[in]** Покажчик на функцію, яку потрібно виконати в контексті HTTPD
- **arg** -- **[in]** Покажчик на аргументи, які слід передати цій функції

Повернення

- ESP_OK : після успішного розміщення роботи в черзі
- ESP_FAIL: Помилка в сокеті ctrl
- ESP_ERR_INVALID_ARG : Нульові аргументи

void * httpd_sess_get_ctx (httpd_handle_t handle , int sockfd)

Отримати контекст сеансу з дескриптора сокета.

Як правило, якщо контекст сеансу створюється, він доступний для обробників URI через структуру httpd_req_t. Але є випадки, коли функції надсилання/отримання веб-сервера можуть потребувати контексту (наприклад, для доступу до інформації про ключ тощо).

Оскільки функції надсилання/отримання мають у своєму розпорядженні лише дескриптор сокета, цей API надає їм спосіб отримати контекст сеансу.

Параметри

- **дескриптор** -- **[in]** дескриптор сервера, повернутий httpd_start
- **sockfd** -- **[in]** Дескриптор сокета, для якого потрібно видобути контекст.

Повернення

- void* : покажчик на контекст, пов’язаний із цим сеансом
- NULL: Void контекст/Void дескриптор/Void fd сокета

void httpd_sess_set_ctx (httpd_handle_t handle , int sockfd , void * ctx , htt pd_free_ctx_fn_t free_fn)

Встановити контекст сеансу за допомогою дескриптора сокета.

Параметри

- **дескриптор** -- **[in]** дескриптор сервера, повернутий httpd_start
- **sockfd** -- **[in]** Дескриптор сокета, для якого потрібно видобути контекст.
- **ctx** -- **[in]** Об’єкт контексту для призначення сеансу
- **free_fn** -- **[in]** Функція, яку слід викликати для звільнення контексту

void * httpd_sess_get_transport_ctx (httpd_handle_t handle , int sockfd)

Отримати контекст «транспорту» сесії за допомогою дескриптора сокета. Цей контекст використовується функціями надсилання/отримання, наприклад, для керування контекстом SSL.

Параметри

- **дескриптор** -- **[in]** дескриптор сервера, повернутий httpd_start
- **sockfd** -- **[in]** Дескриптор сокета, для якого потрібно видобути контекст.

Повернення

- void* : покажчик на транспортний контекст, пов’язаний із цим сеансом
- NULL: Void контекст/Void дескриптор/Void fd сокета

void httpd_sess_set_transport_ctx (httpd_handle_t handle , int sockfd , voi d * ctx , httpd_free_ctx_fn_t free_fn)

Встановіть контекст «транспорту» сесії за допомогою дескриптора сокета.

Параметри

- **дескриптор** -- **[in]** дескриптор сервера, повернутий httpd_start
- **sockfd** -- **[in]** Дескриптор сокета, для якого потрібно видобути контекст.
- **ctx** -- **[in]** Об'єкт транспортного контексту для призначення сеансу
- **free_fn** -- **[in]** Функція, яку слід викликати, щоб звільнити транспортний контекст

void * httpd_get_global_user_ctx (httpd_handle_t дескриптор)

Отримати глобальний контекст користувача HTTPD (його було встановлено в структурі конфігурації сервера)

Параметри

дескриптор -- **[in]** дескриптор сервера, повернутий httpd_start

Повернення

глобальний контекст користувача

void * httpd_get_global_transport_ctx (httpd_handle_t дескриптор)

Отримати глобальний транспортний контекст HTTPD (його було встановлено в структурі конфігурації сервера)

Параметри

дескриптор -- **[in]** дескриптор сервера, повернутий httpd_start

Повернення

глобальний транспортний контекст

esp_err_t httpd_sess_trigger_close(httpd_handle_t дескриптор, int sockfd)

Ініціювати зовнішнє закриття сеансу httpd.

Виклик цього API потрібний лише в особливих обставинах, коли деяка програма вимагає асинхронно закрити сеанс клієнта httpd.

Параметри

- **дескриптор -- [in]** дескриптор сервера, повернутий httpd_start
- **sockfd -- [in]** Дескриптор сокета сеансу, який буде закрито

Повернення

- ESP_OK : після успішного запуску закриття
- ESP_FAIL : Помилка роботи в черзі
- ESP_ERR_NOT_FOUND : Сокет fd не знайдено
- ESP_ERR_INVALID_ARG : Нульові аргументи

esp_err_t httpd_sess_update_lru_counter (httpd_handle_t дескриптор , in t sockfd)

Оновити лічильник LRU для певного сокета.

Лічильники LRU внутрішньо пов’язані з кожним сеансом, щоб відстежувати, як нещодавно сеанс обмінювався трафіком.

Коли очищення LRU увімкнено, якщо клієнт запитує з’єднання, але досягнуто максимальної кількості сокетів/сеансів, сеанс із першим лічильником LRU закривається автоматично.

Оновлення лічильника LRU вручну запобігає очищенню сокета через логіку найменшого використання (LRU), навіть якщо він міг не отримувати трафік протягом деякого часу.

Це корисно, коли всі відкриті сокети/сеанси часто обмінюються трафіком, але користувач конкретно хоче, щоб один із сеансів залишався відкритим, незалежно від того, коли він востаннє обмінювався пакетом.

Виклик цього API необхідний, лише якщо ввімкнено параметр LRU Purge Enable.

Параметри

- **дескриптор -- [in]** дескриптор сервера, повернутий httpd_start
- **sockfd -- [in]** Дескриптор сокета сеансу, для якого потрібно оновити лічильник LRU

Повернення

- ESP_OK: Сокет знайдено та лічильник LRU оновлено
- ESP_ERR_NOT_FOUND : Сокет не знайдено
- ESP_ERR_INVALID_ARG : Нульові аргументи

esp_err_t httpd_get_client_list (httpd_handle_t handle , size_t * fds , int * client_fds)

Повертає список поточних дескрипторів сокетів активних сеансів.

Розмір наданого масиву має дорівнювати або перевищувати максимальну кількість відкритих сокетів, налаштовану під час ініціалізації за допомогою поля max_open_sockets у структурі httpd_config_t.

Параметри

- **дескриптор -- [in]** дескриптор сервера, повернутий httpd_start
- **fds -- [inout]** In: Розмір наданого масиву client_fds Out: кількість дійсних fds клієнта, повернутих у client_fds,
- **client_fds -- [out]** Масив клієнтських fds

Повернення

- ESP_OK : Успішно отримано список сеансів
- ESP_ERR_INVALID_ARG : Неправильні аргументи або список довший за наданий масив

Структури

struct esp_http_server_event_data

Структура аргументів для подій HTTP_SERVER_EVENT_ON_DATA та HTTP_SERVER_EVENT_SENT_DATA

Public члени

int fd

Дескриптор файлу сокета сесії

int data_len

Довжина даних

struct httpd_config

Структура конфігурації сервера HTTP.

Public члени

unsigned task_priority

Пріоритет завдання FreeRTOS, яке запускає сервер

розмір_t стек_розмір

Максимальний розмір стека, дозволений для завдання сервера

BaseType_t core_id

Ядро, на якому виконуватиметься завдання HTTP-сервера

uint32_t task_caps

Можливості пам’яті для використання під час розподілу стека завдання сервера HTTP

uint16_t порт_сервера

Номер порту TCP для отримання та передачі трафіку HTTP

uint16_t ctrl_port

Номер порту UDP для асинхронного обміну керуючими сигналами між різними компонентами сервера

uint16_t max_open_sockets

Максимальна кількість сокетів/клієнтів, підключених у будь-який час (3 сокети зарезервовано для внутрішньої роботи HTTP-сервера)

> **uint16_t max_uri_handlers**

Максимально дозволена кількість обробників uri

> **uint16_t max_resp_headers**

Максимально допустима кількість додаткових заголовків у відповіді HTTP

> **uint16_t backlog_conn**

Кількість невиконаних підключень

> **bool lru_purge_enable**

Очистити підключення «Найменш використане».

> **uint16_t recv_wait_timeout**

Час очікування для функції recv (у секундах)

> **uint16_t send_wait_timeout**

Час очікування функції надсилання (у секундах)

> **void * global_user_ctx**

Глобальний контекст користувача.

Це поле можна використовувати для зберігання довільних даних користувача в контексті сервера. Значення можна отримати за допомогою дескриптора сервера, доступного, наприклад, у структурі httpd_req_t.

Під час завершення роботи сервер звільняє контекст користувача, викликаючи free() у полі global_user_ctx. Якщо ви бажаєте використовувати спеціальну функцію для звільнення глобального контексту користувача, вкажіть це тут.

> **httpd_free_ctx_fn_t global_user_ctx_free_fn**

Безкоштовна функція для глобального контексту користувача

> **void * global_transport_ctx**

Глобальний транспортний контекст.

Подібно до global_user_ctx, але використовується для кодування або шифрування сеансу (наприклад, для зберігання контексту SSL). Його буде

звільнено за допомогою free(), якщо не вказано global_transport_ctx_free_fn.

> **httpd_free_ctx_fn_t global_transport_ctx_free_fn**

Безкоштовна функція для глобального транспортного контексту

> **логічний параметр enable_so_linger**

bool, щоб увімкнути/вимкнути затримку

> **int linger_timeout**

час очікування (у секундах)

> **bool keep_alive_enable**

Увімкнути тайм-аут підтримки активності

> **int keep_alive_idle**

Час простою Keep-alive. За замовчуванням 5 (секунда)

> **int keep_alive_interval**

Час інтервалу підтримки активності. За замовчуванням 5 (секунда)

> **int keep_alive_count**

Кількість повторних спроб надсилання пакетів Keep-alive. За замовчуванням 3 рахунки

httpd_open_func_t open_fn

Зворотний виклик відкриття спеціального сеансу.

Викликається в новому сокеті сеансу одразу після accept(), але перед читанням будь-яких даних.

Це можливість налаштувати, наприклад, шифрування SSL за допомогою global_transport_ctx і перевизначати сеанс надсилання/отримання/очікування.

Якщо потрібно підтримувати контекст між цими функціями, збережіть його в сеансі за допомогою httpd_sess_set_transport_ctx() і отримайте пізніше за допомогою httpd_sess_get_transport_ctx()

Повернення значення, відмінного від ESP_OK, негайно закриє новий сокет.

httpd_close_func_t close_fn

Зворотний виклик закриття спеціального сеансу.

Викликається, коли сеанс видаляється, перед звільненням контекстів користувача та транспорту та перед закриттям сокета. Це місце для спеціального коду деініціалізації, загального для всіх сокетів.

Сервер закриє сокет, лише якщо не встановлено зворотний виклик для закриття сеансу. Якщо використовується настроюваний зворотний виклик, `close(sockfd)` у більшості випадків слід викликати тут.

Встановіть для контексту користувача або транспорту значення NULL, якщо його було звільнено тут, щоб сервер не намагався звільнити його знову.

Ця функція виконується для всіх завершених сеансів, включаючи сеанси, у яких сокет було закрито мережевим стеком, тобто дескриптор файлу може бути недійсним.

httpd_uri_match_func_t uri_match_fn

Функція збігу URI.

Викликається під час пошуку відповідного URI:

1) чий обробник запитів має бути виконано одразу після успішного аналізу HTTP-запиту

2) щоб запобігти дублюванню під час реєстрації нового обробника URI за допомогою httpd_register_uri_handler()

Доступні варіанти:

1) NULL : внутрішнє виконання базової відповідності за допомогою strncmp()

2) httpd_uri_match_wildcard(): збіг символів підстановки URI

Користувачі можуть реалізувати власні функції відповідності

struct httpd_req

Структура даних запиту HTTP.

Public члени

httpd_handle_t дескриптор

Дескриптор екземпляра сервера

метод int

Тип запиту HTTP, -1, якщо метод не підтримується, HTTP_ANY для методу підстановки для підтримки кожного методу

***const* char uri [HTTPD_MAX_URI_LEN + 1]**

URI цього запиту (1 байт додатково для нульового завершення)

size_t content_len

Довжина тіла запиту

void * доп

Внутрішньо використовувані члени

void * user_ctx

Покажчик контексту користувача, переданий під час реєстрації URI.

void * sess_ctx

Покажчик контексту сеансу

Контекст сеансу. Контексти зберігаються через «сеанси» для заданого відкритого TCP-з'єднання. Один сеанс може мати кілька відповідей на запит. Веб-сервер забезпечить збереження контексту в усіх цих запитах і відповідях.

За замовчуванням це NULL. Обробники URI можуть встановити будь-яке значуще значення.

Якщо основний сокет закривається, а цей вказівник не NULL, веб-сервер звільнить контекст, викликавши free(), якщо не встановлено функцію free_ctx.

httpd_free_ctx_fn_t free_ctx

Покажчик на хук вільного контексту

Функція звільнення контексту сеансу

Якщо сокет веб-сервера закривається, контекст сеансу звільняється шляхом виклику free() для члена sess_ctx. Якщо ви бажаєте використовувати спеціальну функцію для звільнення контексту сеансу, вкажіть це тут.

bool ignore_sess_ctx_changes

Прапорець, що вказує, чи слід ігнорувати зміни контексту сеансу

За замовчуванням, якщо ви зміните sess_ctx у якомусь обробнику URI, http-сервер внутрішньо звільнить попередній контекст (якщо не NULL), після повернення обробника URI.

struct httpd_uri

Структура обробника URI.

Public члени

> ***const* char * uri**

URI для обробки

> **Метод httpd_method_t**

Метод підтримується URI, HTTP_ANY для методу підстановки для підтримки всіх методів

> **esp_err_t (* обробник) (httpd_req_t * r)**

Обробник для виклику підтримуваного методу запиту. Це має повернути ESP_OK, інакше основний сокет буде закрито.

> **void * user_ctx**

Покажчик на дані контексту користувача, які будуть доступні обробнику

Макроси

HTTP_ANY
HTTPD_MAX_REQ_HDR_LEN
HTTPD_MAX_URI_LEN
HTTPD_SOCK_ERR_FAIL
HTTPD_SOCK_ERR_INVALID
HTTPD_SOCK_ERR_TIMEOUT
HTTPD_200
Відповідь HTTP 200

HTTPD_204

Відповідь HTTP 204

HTTPD_207

Відповідь HTTP 207

HTTPD_400

Відповідь HTTP 400

HTTPD_404

Відповідь HTTP 404

HTTPD_408

Відповідь HTTP 408

HTTPD_500

Відповідь HTTP 500

HTTPD_TYPE_JSON

Тип вмісту HTTP JSON

HTTPD_TYPE_TEXT

Тип вмісту HTTP text/HTML

HTTPD_TYPE_OCTET

Тип вмісту HTTP octext-stream

ESP_HTTPD_DEF_CTRL_PORT

Порт керування сокетом сервера HTTP

HTTPD_DEFAULT_CONFIG ()

ESP_ERR_HTTPD_BASE

Початкова кількість кодів помилок HTTPD

ESP_ERR_HTTPD_HANDLERS_FULL

Усі слоти для реєстрації обробників URI використано

ESP_ERR_HTTPD_HANDLER_EXISTS

Обробник URI з тим самим методом і цільовий URI вже зареєстрований

ESP_ERR_HTTPD_INVALID_REQ

Void покажчик запиту

ESP_ERR_HTTPD_RESULT_TRUNC

Рядок результату скорочено

ESP_ERR_HTTPD_RESP_HDR

Поле заголовка відповіді більше ніж підтримується

ESP_ERR_HTTPD_RESP_SEND

Під час надсилання пакета відповіді сталася помилка

ESP_ERR_HTTPD_ALLOC_MEM

Не вдалося динамічно виділити пам’ять для ресурсу

ESP_ERR_HTTPD_TASK

Не вдалося запустити завдання/потік сервера

HTTPD_RESP_USE_STRLEN

Визначення типів

typedef void * httpd_handle_t

Дескриптор екземпляра сервера HTTP.

Кожен екземпляр сервера матиме унікальний дескриптор.

typedef enum http_method httpd_method_t

HTTP Method Type оболонка над "enum http_method", доступна в бібліотеці "http_parser".

typedef void (* httpd_free_ctx_fn_t) (void * ctx)

Прототип для звільнення контекстних даних (якщо є)

Параметр ctx

[в] об’єкт для звільнення

typedef esp_err_t (* httpd_open_func_t) (httpd_handle_t hd , int sockfd)

Прототип функції для відкриття сесії.

Викликається відразу після відкриття сокета для налаштування функцій надсилання/отримання та інших параметрів сокета.

Параметр hd

[в] примірнику сервера

Параметр sockfd

[в] дескриптор файлу сокета сесії

Повернення

- ESP_OK : У разі успіху
- Будь-яке значення, відмінне від ESP_OK, сигналізує серверу негайно закрити сокет

typedef void (* httpd_close_func_t) (httpd_handle_t hd , int sockfd)

Прототип функції для закриття сесії.

Можливо, на даний момент дескриптор сокета void, функція викликається для всіх завершених сеансів. Забезпечте належну обробку кодів повернення.

Параметр hd

[в] примірнику сервера

Параметр sockfd

[в] дескриптор файлу сокета сесії

typedef bool (* httpd_uri_match_func_t) (const char * reference_uri , cons t char * uri_to_match , size_t match_upto)

Прототип функції для відповідності URI.

Параметр reference_uri

[in] URI/шаблон, щодо якого збігається інший URI

Параметр uri_to_match

[in] URI/шаблон зіставляється з еталонним URI/шаблоном

Параметр match_upto

[in] Для вказівки фактичної довжини, `uri_to_match` до якої має застосовуватися алгоритм відповідності (максимальне значення становить `strlen(uri_to_match)`, незалежно від довжини `reference_uri`)

Повернення

typedef struct httpd_config httpd_config_t

Структура конфігурації сервера HTTP.

Використовує HTTPD_DEFAULT_CONFIG(), щоб ініціалізувати конфігурацію до значення за замовчуванням, а потім змінює лише ті поля, які конкретно визначені варіантом використання.

typedef struct httpd_req httpd_req_t

Структура даних запиту HTTP.

typedef struct httpd_uri httpd_uri_t

Структура обробника URI.

typedef int (* httpd_send_func_t) (httpd_handle_t hd , int sockfd , const char * buf , size_t buf_len , int flags)

Прототип функції надсилання низького рівня HTTPD.

Зазначена користувачем функція надсилання повинна обробляти помилки внутрішньо, залежно від установленого значення errno, і повертати певні коди HTTPD_SOCK_ERR_, які зрештою будуть передані як значення, що повертається функцією httpd_send().

Параметр hd

[в] примірнику сервера

Параметр sockfd

[в] дескриптор файлу сокета сесії

Параметр buf

[in] буфер із байтами для надсилання

Параметр buf_len

[in] розмір даних

Прапори параметрів

прапорці [in] для функції send().

Повернення

- Байти: кількість успішно надісланих байтів
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Час очікування/перерва під час виклику socket send()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket send()

typedef int (* httpd_recv_func_t) (httpd_handle_t hd , int sockfd , char * buf , size_t buf_len , int flags)

Прототип для низькорівневої функції recv HTTPD.

Зазначена користувачем функція recv повинна внутрішньо обробляти помилки, залежно від установленого значення errno, і повертати певні коди HTTPD_SOCK_ERR_, які зрештою будуть передані як значення, що повертається функцією httpd_req_recv().

Параметр hd

[в] примірнику сервера

Параметр sockfd

[в] дескриптор файлу сокета сесії

Параметр buf

[in] буфер із байтами для надсилання

Параметр buf_len

[in] розмір даних

Прапори параметрів

прапорці [in] для функції send().

Повернення

- Байти: кількість успішно отриманих байтів

- 0: параметр довжини буфера дорівнює нулю / з’єднання закрито одноранговим вузлом
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Час очікування/перерва під час виклику socket recv()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket recv()

typedef int (* httpd_pending_func_t) (httpd_handle_t hd , int sockfd)

Прототип низькорівневої функції «отримати байти в очікуванні» для HTTPD.

Зазначена користувачем функція очікування повинна обробляти помилки внутрішньо, залежно від встановленого значення errno, і повертати певні коди HTTPD_SOCK_ERR_, які відповідно оброблятимуться в завданні сервера.

Параметр hd

[в] примірнику сервера

Параметр sockfd

[в] дескриптор файлу сокета сесії

Повернення

- Байти : кількість байтів, які очікують на отримання
- HTTPD_SOCK_ERR_INVALID : void аргументи
- HTTPD_SOCK_ERR_TIMEOUT : Тайм-аут/перервано під час виклику socket pending()
- HTTPD_SOCK_ERR_FAIL : Невиправна помилка під час виклику socket pending()

typedef esp_err_t (* httpd_err_handler_func_t) (httpd_req_t * req , помилка httpd_err_code_t)

Прототип функції для обробки помилок HTTP.

Ця функція виконується після помилок HTTP, які виникають під час внутрішньої обробки HTTP-запиту. Це використовується для заміни поведінки за замовчуванням у разі помилки, тобто надсилання відповіді на помилку HTTP та закриття основного сокета.

Якщо це реалізовано, сервер не буде автоматично надсилати коди відповіді на помилку HTTP, тому httpd_resp_send_err() має бути викликано всередині цієї функції, якщо користувач хоче генерувати відповіді на помилку HTTP.

Під час виклику дійсність полів `uri`, `method` і `content_len` параметра `user_ctx` httpd_req_t не гарантується, оскільки HTTP-запит може бути частково отримано/проаналізовано.

Функція має повертати ESP_OK, якщо основний сокет потрібно залишати відкритим. Будь-яке інше значення гарантує, що сокет закрито.

Повернене значення ігнорується, якщо помилка має тип `HTTPD_500_INTERNAL_SERVER_ERROR` і сокет все одно закрито.

Param req

[in] HTTP-запит, для якого потрібно обробити помилку

Помилка параметра

[in] Тип помилки

Повернення

- ESP_OK: помилка оброблена успішно
- ESP_FAIL: помилка вказує на те, що основний сокет потрібно закрити

typedef void (* httpd_work_fn_t) (void * arg)

Прототип робочої функції HTTPD Будь ласка, зверніться до httpd_queue_work() для отримання додаткової інформації.

Параметр арг

[in] Аргументи для цієї робочої функції

Перерахування

enum httpd_err_code_t

Коди помилок, які надсилаються як HTTP-відповідь у разі виникнення помилок під час обробки HTTP-запиту.

Значення:

> ***enumerator* HTTPD_500_INTERNAL_SERVER_ERROR**
>
> ***enumerator* HTTPD_501_METHOD_NOT_IMPLEMENTED**
>
> ***enumerator* HTTPD_505_VERSION_NOT_SUPPORTED**
>
> ***enumerator* HTTPD_400_BAD_REQUEST**
>
> ***enumerator* HTTPD_401_UNAUTHORIZED**
>
> ***enumerator* HTTPD_403_FORBIDDEN**
>
> ***enumerator* HTTPD_404_NOT_FOUND**
>
> ***enumerator* HTTPD_405_METHOD_NOT_ALLOWED**
>
> ***enumerator* HTTPD_408_REQ_TIMEOUT**
>
> ***enumerator* HTTPD_411_LENGTH_REQUIRED**
>
> ***enumerator* HTTPD_414_URI_TOO_LONG**
>
> ***enumerator* HTTPD_431_REQ_HDR_FIELDS_TOO_LARGE**
>
> ***enumerator* HTTPD_ERR_CODE_MAX**

enum esp_http_server_event_id_t

Ідентифікатор подій сервера HTTP.

Значення:

> ***enumerator* HTTP_SERVER_EVENT_ERROR**

Ця подія виникає, коли під час виконання виникають будь-які помилки

> ***enumerator* HTTP_SERVER_EVENT_START**

Ця подія виникає під час запуску сервера HTTP

> ***enumerator* HTTP_SERVER_EVENT_ON_CONNECTED**

Після підключення HTTP-сервера до клієнта обмін даними не здійснювався

> ***enumerator* HTTP_SERVER_EVENT_ON_HEADER**

Відбувається під час отримання кожного заголовка, надісланого від клієнта

> ***enumerator* HTTP_SERVER_EVENT_HEADERS_SENT**

Після відправки всіх заголовків клієнту

> ***enumerator* HTTP_SERVER_EVENT_ON_DATA**

Виникає при отриманні даних від клієнта

> ***enumerator* HTTP_SERVER_EVENT_SENT_DATA**

Відбувається після завершення сеансу HTTP-сервера ESP

> ***enumerator* HTTP_SERVER_EVENT_DISCONNECTED**

З'єднання було розірвано

> ***enumerator* HTTP_SERVER_EVENT_STOP**

Ця подія виникає, коли сервер HTTP зупинено

КЛІЄНТ ESP WEBSOCKET

Огляд

Клієнт ESP WebSocket є реалізацією клієнта протоколу WebSocket для ESP32

Особливості

- Підтримує WebSocket через TCP, TLS з mbedtls
- Легко налаштувати за допомогою URI
- Кілька екземплярів (кілька клієнтів в одній програмі)

Конфігурація

Мінімальні комплектації:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://echo.websocket.org",
};
```

Клієнт WebSocket підтримує використання як шляху, так і запиту в URI.

Зразок:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://echo.websocket.org/connectionhandler?id=104",
};
```

Якщо є будь-які параметри, пов’язані з URI в **esp_websocket_client_config_t**, параметр, визначений URI, буде замінено. Зразок:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://echo.websocket.org:123",
    .port = 4567,
};
//WebSocket client will connect to websocket.org using port 4567
```

TLS

Конфігурація:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "wss://echo.websocket.org",
    .cert_pem = (const char *)websocket_org_pem_start,
};
```

Якщо ви хочете перевірити сервер, вам потрібно надати сертифікат у форматі PEM і надати cert_pemв websocket_client_config_t. Якщо сертифікат не надано, з'єднання TLS за замовчуванням не вимагає перевірки.

Сертифікат PEM для цього прикладу можна отримати з команди openssl s_client , яка підключається до websocket.org.

Якщо в операційній системі хоста інстальовано пакунки openssl і sed , можна виконати таку команду, щоб завантажити та зберегти кореневий або проміжний кореневий сертифікат у файл

Підпротокол

Поле підпротоколу в структурі конфігурації можна використовувати для запиту підпротоколу

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://websocket.org",
    .subprotocol = "soap",
};
```

Щоб отримати додаткові параметри щодо **esp_websocket_client_config_t**, зверніться до посилання на API нижче

Події

- *WEBSOCKET_EVENT_CONNECTED* : Клієнт успішно встановив з'єднання із сервером. Тепер клієнт готовий надсилати й отримувати дані. Не містить даних про події.

- *WEBSOCKET_EVENT_DISCONNECTED* : Клієнт перервав з'єднання через те, що транспортний рівень не зміг прочитати дані, наприклад через те, що сервер недоступний. Не містить даних про події.
- *WEBSOCKET_EVENT_DATA* : Клієнт успішно отримав і проаналізував кадр WebSocket. Дані події містять покажчик на дані корисного навантаження, довжину даних корисного навантаження, а також код операції отриманого кадру. Повідомлення може бути фрагментоване на кілька подій, якщо довжина перевищує розмір буфера. Ця подія також буде опублікована для кадрів без корисного навантаження, наприклад для кадрів понг або закриття з'єднання.
- *WEBSOCKET_EVENT_ERROR* : не використовується в поточній реалізації клієнта.

Якщо дескриптор клієнта потрібен в обробнику подій, до нього можна отримати доступ через покажчик, переданий обробнику подій:

```
esp_websocket_client_handle_t client =
(esp_websocket_client_handle_t)handler_args;
```

Обмеження та відомі проблеми

Клієнт може подати запит на використання підпротоколу від сервера під час рукостискання, але не виконує жодних пов'язаних із підпротоколом перевірок відповіді від сервера.

Надсилання текстових даних

Клієнт WebSocket підтримує надсилання даних у вигляді текстового кадру даних, який інформує прикладний рівень, що дані корисного навантаження є текстовими даними, закодованими як UTF-8. приклад:

```
esp_websocket_client_send_text(client, data, len, portMAX_DELAY);
```

1ДОВІДНИК API

Файл заголовка

../components/esp_websocket_client/include/esp_websocket_client.h

ФУНКЦІЇ

esp_websocket_client_handle_t esp_websocket_client_init (const esp_websocket_client_config_t * config)

Розпочати сеанс Websocket Ця функція має бути першою функцією для виклику, і вона повертає esp_websocket_client_handle_t, який ви повинні використовувати як вхідні дані для інших функцій в інтерфейсі.

Цей виклик ПОВИНЕН мати відповідний виклик esp_websocket_client_destroy після завершення операції.

Параметри

config – **[in]** Конфігурація

Повернення

- esp_websocket_client_handle_t
- NULL, якщо є помилки

esp_err_t esp_websocket_client_set_uri (клієнт esp_websocket_client_handle_t , const char * uri)

Встановити URL-адресу для клієнта, під час виконання цієї поведінки параметри в URL-адресі замінять старі. Необхідно зупинити клієнт WebSocket перед встановленням URI, якщо клієнт підключено.

Параметри

- **клієнт** – **[в]** Клієнт
- **uri** – **[в]** uri

Повернення

esp_err_t

esp_err_t esp_websocket_client_set_headers (клієнт esp_websocket_client _handle_t , const char * заголовки)

Встановіть додаткові заголовки websocket для клієнта, під час виконання цієї поведінки заголовки замінять старі.

- Цей API слід використовувати після успішного підключення клієнта WebSocket (тобто після ініціалізації транспортного рівня).
- Якщо ви бажаєте встановити або додати заголовки до встановлення підключення клієнта WebSocket (до рукостискання), розгляньте такі варіанти:
  a. Введіть заголовки безпосередньо в параметри конфігурації, закінчуючи кожен елемент [CR][LF]. Цей підхід замінить усі попередні заголовки. Приклад: websocket_cfg.headers = "Sec-WebSocket-Key: my_key\r\nПароль: my_pass\r\n";
  b. Використовуйте esp_websocket_client_append_header API, щоб додати один заголовок до поточного набору.

попередньо

Необхідно зупинити клієнт WebSocket перед встановленням заголовків, якщо клієнт підключено

Параметри

- **client** – **[in]** Дескриптор клієнта WebSocket
- **заголовки** – **[in]** Додаткові рядки заголовків, кожен із яких закінчується [CR][LF]

Повернення

esp_err_t

esp_err_t esp_websocket_client_append_header (клієнт esp_websocket_client_handle_t , const char * ключ , const char * значення)

Додає нову пару ключ-значення до заголовків клієнта WebSocket.

Параметри

- **client** – **[in]** Дескриптор клієнта WebSocket
- **key** – **[in]** Ключ заголовка для додавання
- **value** – **[in]** Пов’язане значення для заданого ключа

попередньо

Переконайтеся, що ця функція викликається перед запуском клієнта WebSocket.

Повернення

esp_err_t

esp_err_t esp_websocket_client_start (клієнт esp_websocket_client_handle_t)

Відкрийте підключення WebSocket.

Параметри

клієнт – **[в]** Клієнт

Повернення

esp_err_t

esp_err_t esp_websocket_client_stop (клієнт esp_websocket_client_handle_t)

Зупиняє з’єднання WebSocket без рукостискання закриття websocket.

Цей API зупиняє ws-клієнт і закриває TCP-з’єднання безпосередньо без надсилання закритих кадрів. Рекомендується закрити з’єднання чистим способом за допомогою esp_websocket_client_close().

Примітки:

- Неможливо викликати з обробника подій websocket

Параметри

клієнт – **[в]** Клієнт

Повернення

esp_err_t

esp_err_t esp_websocket_client_destroy (клієнт esp_websocket_client_handle_t)

Зруйнує з’єднання WebSocket і звільніть усі ресурси. Ця функція має бути останньою функцією, яка викликається для сеансу.

Це протилежність функції esp_websocket_client_init і її потрібно викликати з тим самим дескриптором, що й вхідні дані, які повернув виклик esp_websocket_client_init.

Це може закрити всі з’єднання, які використовував цей маркер.

Примітки:

- Неможливо викликати з обробника подій websocket

Параметри

клієнт – **[в]** Клієнт

Повернення

esp_err_t

esp_err_t esp_websocket_client_destroy_on_exit (клієнт esp_websocket_client_handle_t)

Якщо цей API викликається, клієнт WebSocket знищить і звільнить усі ресурси в кінці циклу подій.

Примітки:

- Після завершення циклу подій дескриптор клієнта буде висіти, і його ніколи не слід використовувати

Параметри

клієнт – **[в]** Клієнт

Повернення

esp_err_t

int esp_websocket_client_send_bin(клієнт esp_websocket_client_handle_t, const char * data , int len , час очікування TickType_t)

Запис двійкових даних у з'єднання WebSocket (дані надсилаються з WS OPCODE=02, тобто двійкові)

Параметри

- **клієнт** – **[в]** Клієнт
- **дані** – **[в]** дані
- **len** – **[in]** Довжина
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

int esp_websocket_client_send_bin_partial (клієнт esp_websocket_client_ handle_t , const char * data , int len , час очікування TickType_t)

Запис двійкових даних у з'єднання WebSocket і надсилання без встановлення прапора FIN (дані надсилаються з WS OPCODE=02, тобто двійкові)

Примітки:

- Щоб надіслати фрейм продовження, ви повинні використовувати API 'esp_websocket_client_send_cont_msg(…)'.
- Щоб позначити кінець фрагментованих даних, слід використовувати API esp_websocket_client_send_fin(…)». Це надсилає кадр FIN.

Параметри

- **клієнт** – **[в]** Клієнт

- **дані** – **[в]** дані
- **len** – **[in]** Довжина
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

int esp_websocket_client_send_text (клієнт esp_websocket_client_handle_t , const char * data , int len , час очікування TickType_t)

Запис текстових даних у з'єднання WebSocket (дані надсилаються з WS OPCODE=01, тобто текст)

Параметри

- **клієнт** – **[в]** Клієнт
- **дані** – **[в]** дані
- **len** – **[in]** Довжина
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

int esp_websocket_client_send_text_partial (клієнт esp_websocket_client_handle_t , const char * data , int len , час очікування TickType_t)

Запис текстових даних у з'єднання WebSocket і надсилання їх без встановлення прапора FIN (дані надсилаються з WS OPCODE=01, тобто текст)

Примітки:

- Щоб надіслати фрейм продовження, слід використовувати API esp_websocket_client_send_cont_mgs(…)».
- Щоб позначити кінець фрагментованих даних, слід використовувати API esp_websocket_client_send_fin(…)». Це надсилає кадр FIN.

Параметри

- **клієнт** – **[в]** Клієнт
- **дані** – **[в]** дані
- **len** – **[in]** Довжина
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

int esp_websocket_client_send_cont_msg (клієнт esp_websocket_client_handle_t , const char * data , int len , час очікування TickType_t)

Запис текстових даних у з'єднання WebSocket і надсилання їх як кадр продовження (OPCODE=0x0)

Примітки:

- Кадри продовження мають код операції 0x0 і явно не вказують, чи вони продовжують текстове чи двійкове повідомлення.
- Ви визначаєте тип повідомлення (текстове чи двійкове), яке продовжується, дивлячись на код операції початкового кадру в послідовності фрагментованих кадрів.
- Щоб позначити кінець фрагментованих даних, слід використовувати API esp_websocket_client_send_fin(…)». Це надсилає кадр FIN.

Параметри

- **клієнт** – **[в]** Клієнт
- **дані** – **[в]** дані
- **len** – **[in]** Довжина
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

int esp_websocket_client_send_fin (клієнт esp_websocket_client_handle_t, час очікування TickType_t)

Надсилає кадр FIN.

Параметри

- **клієнт** – **[в]** Клієнт
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

int esp_websocket_client_send_with_opcode (esp_websocket_client_handle_t client , ws_transport_opcodes_t opcode , const uint8_t * data , int len, TickType_t timeout)

Запис даних коду операції до з’єднання WebSocket.

Примітки:

- Щоб надіслати нульове корисне навантаження, data та len мають бути встановлені на NULL/0
- Цей API встановлює біт FIN на останній фрагмент повідомлення

Параметри

- **клієнт** – **[в]** Клієнт
- **код операції** – **[in]** Код операції
- **дані** – **[в]** дані
- **len** – **[in]** Довжина
- **timeout** – **[in]** Час очікування запису даних у тиках RTOS

Повернення

- Кількість надісланих даних
- (-1) якщо є помилки

esp_err_t esp_websocket_client_close (клієнт esp_websocket_client_handle_t , час очікування TickType_t)

Закрийте підключення WebSocket чистим способом.

Послідовність чистого закриття, ініційованого клієнтом:

- Клієнт надсилає кадр CLOSE
- Клієнт чекає, доки сервер відобразить кадр CLOSE
- Клієнт чекає, поки сервер не закриє з'єднання
- Клієнт зупиняється так само, як і esp_websocket_client_stop()

Примітки:

- Неможливо викликати з обробника подій websocket

Параметри

- **клієнт** – **[в]** Клієнт
- **timeout** – **[in]** Тайм-аут в тиках RTOS для очікування

Повернення

esp_err_t

esp_err_t esp_websocket_client_close_with_code (клієнт esp_websocket_client_handle_t , int code , const char * data , int len , TickType_t timeout)

Закрийте з’єднання WebSocket чистим способом за допомогою спеціального коду/даних. Послідовність закриття така ж, як і для esp_websocket_client_close()

Примітки:

- Неможливо викликати з обробника подій websocket

Параметри

- **клієнт** – **[в]** Клієнт
- **код** – **[in]** Закрити код статусу, як визначено в розділі 7.4 RFC6455
- **data** – **[in]** Додаткові дані до кінцевого повідомлення
- **len** – **[in]** Довжина додаткових даних
- **timeout** – **[in]** Тайм-аут в тиках RTOS для очікування

Повернення

esp_err_t

bool esp_websocket_client_is_connected (клієнт esp_websocket_client_handle_t)

Перевірте стан підключення клієнта WebSocket.

Параметри

client – **[in]** Керівник клієнта

Повернення

- правда
- помилковий

size_t esp_websocket_client_get_ping_interval_sec (клієнт esp_websocket_client_handle_t)

Отримайте інтервал ping для клієнта в секундах.

Параметри

клієнт – **[в]** Клієнт

Повернення

Інтервал ping в сек

esp_err_t esp_websocket_client_set_ping_interval_sec (клієнт esp_websocket_client_handle_t , size_t ping_interval_sec)

Встановити новий інтервал ping для клієнта.

Параметри

- **клієнт** – **[в]** Клієнт
- **ping_interval_sec** – **[in]** Новий інтервал

Повернення

esp_err_t

esp_err_t esp_websocket_register_events (клієнт esp_websocket_client_handle_t , подія esp_websocket_event_id_t , обробник_події esp_event_handler_t , void * event_handler_arg)

Зареєструйте події Websocket.

Параметри

- **клієнт** – дескриптор клієнта
- **event** – ідентифікатор події
- **event_handler** – функція зворотного виклику
- **event_handler_arg** – контекст користувача

Повернення

esp_err_t

Структури

struct esp_websocket_error_codes_t

Структура коду помилки Websocket, яка передається як контекстна інформація в подію ERROR.

Public члени

esp_err_t esp_tls_last_esp_err

останній код esp_err, отриманий від компонента esp-tls

int esp_tls_stack_err

Спеціальний код помилки tls повідомляється з базового стека tls

int esp_tls_cert_verify_flags

позначки tls, отримані з основного стеку tls під час перевірки сертифіката

int esp_ws_handshake_status_code

Код статусу http рукостискання оновлення websocket

int esp_transport_sock_errno

errno з базового сокета

struct esp_websocket_event_data_t

Дані подій Websocket.

Public члени

> ***const* char * data_ptr**

Покажчик даних

> **int data_len**

Довжина даних

> **bool fin**

Плавник прапор

> **код операції uint8_t**

Отримано код операції

> **клієнт esp_websocket_client_handle_t**

контекст esp_websocket_client_handle_t

> **void * user_context**

контекст user_data, з esp_websocket_client_config_t user_data

> **int payload_len**

Загальна довжина корисного навантаження, корисне навантаження, що перевищує буфер, буде опубліковано через кілька подій

> **int payload_offset**

Фактичний зсув для даних, пов’язаних із цією подією

> **esp_websocket_error_codes_t error_handle**

дескриптор помилки esp-websocket, включаючи помилки esp-tls, а також внутрішні помилки websocket

struct esp_websocket_client_config_t

Конфігурація клієнта Websocket.

Public члени

> ***const* char * uri**

Websocket URI, інформація про URI може перевизначати інші поля нижче, якщо такі є

> ***const* char * хост**

Домен або IP як рядок

внутр . порт

Порт для підключення, за умовчанням залежить від esp_websocket_transport_t (80 або 443)

***const* char * ім'я користувача**

Використовується для автентифікації Http, зараз підтримується лише базова автентифікація

***const* char * пароль**

Використання для автентифікації Http

***const* char * шлях**

Шлях HTTP, якщо не встановлено, за замовчуванням /

bool disable_auto_reconnect

Вимкніть функцію автоматичного повторного підключення після відключення

void * user_context

Контекст даних користувача HTTP

int task_prio

Пріоритет завдання Websocket

***const* char * ім'я_завдання**

Назва завдання Websocket

int task_stack

Стек завдань Websocket

int розмір_буфера

Розмір буфера Websocket

***const* char * cert_pem**

Покажчик на дані сертифіката у форматі PEM або DER для перевірки сервера (з SSL), за умовчанням NULL, не потрібно для перевірки сервера. PEM-формат повинен мати кінцевий NULL-символ. Формат DER вимагає передачі довжини в cert_len.

size_t cert_len

Довжина буфера, на який вказує cert_pem. Може бути 0 для pem з нульовим закінченням

> ***const* char * client_cert**

Покажчик на дані сертифіката у форматі PEM або DER для взаємної автентифікації SSL, за умовчанням NULL, не потрібно, якщо взаємна автентифікація не потрібна. Якщо він не NULL, його також `client_key` потрібно вказати. PEM-формат повинен мати кінцевий NULL-символ. Формат DER вимагає передачі довжини в client_cert_len.

> **size_t client_cert_len**

Довжина буфера, на який вказує client_cert. Може бути 0 для pem з нульовим закінченням

> ***const* char * client_key**

Покажчик на дані приватного ключа у форматі PEM або DER для взаємної автентифікації SSL, за умовчанням NULL, не потрібно, якщо взаємна автентифікація не потрібна. Якщо він не NULL, його також `client_cert` потрібно вказати. PEM-формат повинен мати кінцевий NULL-символ. Формат DER вимагає передачі довжини в client_key_len

> **size_t client_key_len**

Довжина буфера, на який вказує client_key_pem. Може бути 0 для pem з нульовим закінченням

> **esp_websocket_transport_t транспорт**

Тип транспорту Websocket див. `esp_websocket_transport_t

> ***const* char * підпротокол**

Підпротокол Websocket

> ***const* char * user_agent**

Агент користувача Websocket

> ***const* char * заголовки**

Додаткові заголовки Websocket

> **int pingpong_timeout_sec**

Період до з’єднання переривається через відсутність повідомлень PONG

bool disable_pingpong_discon

Вимкнути автоматичне відключення, оскільки PONG не отримано протягом pingpong_timeout_sec

bool use_global_ca_store

Використовуйте глобальне ca_store для всіх з’єднань, у яких встановлено цей логічний параметр.

esp_err_t (* crt_bundle_attach) (void * conf)

Покажчик функції на esp_crt_bundle_attach. Вмикає використання пакета сертифікації для перевірки сервера, MBEDTLS_CERTIFICATE_BUNDLE має бути ввімкнено в menuconfig. Включіть esp_crt_bundle.h і використовуйте esp_crt_bundle_attach тут, щоб включити пакетні сертифікати ЦС.

bool skip_cert_common_name_check

Пропустіть будь-яке поле CN сертифіката сервера

bool keep_alive_enable

Увімкнути тайм-аут підтримки активності

int keep_alive_idle

Час простою Keep-alive. За замовчуванням 5 (секунда)

int keep_alive_interval

Час інтервалу підтримки активності. За замовчуванням 5 (секунда)

int keep_alive_count

Кількість повторних спроб надсилання пакетів Keep-alive. За замовчуванням 3 рахунки

int reconnect_timeout_ms

Повторне підключення після цього значення в мілісекундах, якщо disable_auto_reconnect не ввімкнено (за замовчуванням 10 с)

int network_timeout_ms

Перервати мережеву операцію, якщо вона не завершена після цього значення, у мілісекундах (за замовчуванням 10 с)

size_t ping_interval_sec

Інтервал ping Websocket, за замовчуванням 10 секунд, якщо не встановлено

***struct* ifreq * if_name**

Назва інтерфейсу для проходження даних. Використовуйте стандартний інтерфейс без налаштування

Визначення типів

typedef struct esp_websocket_client * esp_websocket_client_handle_t

Перерахування

перелік esp_websocket_event_id_t

Ідентифікатор подій клієнта Websocket.

Значення:

***enumerator* WEBSOCKET_EVENT_ANY**

***enumerator* WEBSOCKET_EVENT_ERROR**

Ця подія виникає, коли під час виконання виникають будь-які помилки

***enumerator* WEBSOCKET_EVENT_CONNECTED**

Після підключення Websocket до сервера обмін даними не здійснювався

***enumerator* WEBSOCKET_EVENT_DISCONNECTED**

З'єднання було розірвано

***enumerator* WEBSOCKET_EVENT_DATA**

При отриманні даних із сервера, можливо, кількох частин пакета

***enumerator* WEBSOCKET_EVENT_CLOSED**

З’єднання закрито чисто

***enumerator* WEBSOCKET_EVENT_BEFORE_CONNECT**

Подія відбувається перед підключенням

***enumerator* WEBSOCKET_EVENT_MAX**

перелік esp_websocket_error_type_t

Коди помилок з’єднання Websocket поширюються через подію ERROR.

Значення:

> ***enumerator* WEBSOCKET_ERROR_TYPE_NONE**
>
> ***enumerator* WEBSOCKET_ERROR_TYPE_TCP_TRANSPORT**
>
> ***enumerator* WEBSOCKET_ERROR_TYPE_PONG_TIMEOUT**
>
> ***enumerator* WEBSOCKET_ERROR_TYPE_HANDSHAKE**

перелік esp_websocket_transport_t

Транспорт клієнта Websocket.

Значення:

> ***enumerator* WEBSOCKET_TRANSPORT_UNKNOWN**

Транспорт невідомий

> ***enumerator* WEBSOCKET_TRANSPORT_OVER_TCP**

Транспорт через tcp

> ***enumerator* WEBSOCKET_TRANSPORT_OVER_SSL**

Транспорт через ssl

СЕРВЕР HTTPS

Огляд

Цей компонент створено на основі HTTP-сервера.

Сервер HTTPS використовує функції реєстрації гаків на звичайному сервері HTTP, щоб забезпечити функцію зворотного виклику для сеансу SSL.

Використані API

Наступні API HTTP-сервера не слід використовувати з HTTPS-сервером , оскільки вони використовуються внутрішньо для обробки безпечних сеансів і підтримки внутрішнього стану:

- Функції реєстрації зворотного виклику "надсилати", "отримувати" та "очікувати" - безпечна обробка сокетів
  - **httpd_sess_set_send_override()**
  - **httpd_sess_set_recv_override()**
  - **httpd_sess_set_pending_override()**
- «транспортний контекст» - як глобальний, так і сесійний
  - **httpd_sess_get_transport_ctx()** - повертає SSL, використаний для сесії
  - **httpd_sess_set_transport_ctx()**
  - **httpd_get_global_transport_ctx()** - повертає спільний контекст SSL
  - **httpd_config::global_transport_ctx**
  - **httpd_config::global_transport_ctx_free_fn**
  - **httpd_config::open_fn** - використовується для встановлення безпечних розеток

Все інше можна використовувати без обмежень.

Використання

Сервер можна запустити з SSL або без нього, змінивши позначку в структурі ініціалізації - **httpd_ssl_config::transport_mode**. Це можна використовувати, наприклад, для тестування або в надійних середовищах, де ви віддаєте перевагу швидкості над безпекою.

Продуктивність

Початкове налаштування сеансу може тривати приблизно дві секунди або більше з нижчою тактовою частотою або більш детальним журналюванням. Подальші запити через відкритий захищений сокет виконуються набагато швидше (до 100 мс).

Обробка подій

Сервер ESP HTTPS має різні події, для яких бібліотека циклу подій може запускати обробник, коли відбувається певна подія. Обробник має бути зареєстрований за допомогою **esp_event_handler_register()**. Це допомагає в обробці подій для сервера ESP HTTPS. **esp_https_server_event_id_t** містить усі події, які можуть статися для сервера ESP HTTPS.

Очікуваний тип даних для різних подій сервера ESP HTTPS у циклі подій:

- HTTPS_SERVER_EVENT_ERROR : esp_https_server_last_error_t
- HTTPS_SERVER_EVENT_START : NULL
- HTTPS_SERVER_EVENT_ON_CONNECTED : NULL
- HTTPS_SERVER_EVENT_ON_DATA : int
- HTTPS_SERVER_EVENT_SENT_DATA : NULL
- HTTPS_SERVER_EVENT_DISCONNECTED : NULL
- HTTPS_SERVER_EVENT_STOP : NULL

ДОВІДНИК API

Файл заголовка

components/esp_https_server/include/esp_https_server.h

Цей файл заголовка є частиною API, що надається компонентом `esp_https_server`.

Щоб оголосити, що ваш компонент залежить від `esp_https_server`, додайте наступне до свого CMakeLists.txt:

REQUIRES esp_https_server

або

PRIV_REQUIRES esp_https_server

ФУНКЦІЇ

esp_err_t httpd_ssl_start (httpd_handle_t * дескриптор , httpd_ssl_config_t * конфігурація)

Створює сервер HTTP з підтримкою SSL (захищений режим може бути вимкнено в конфігурації)

Параметри

- **config -- [inout]** - конфігурація сервера, не має бути const. Не має залишатися дійсним після виклику цієї функції.
- **handle -- [out]** - сховище для дескриптора сервера, має бути дійсним покажчиком

Повернення

успіх

esp_err_t httpd_ssl_stop (httpd_handle_t дескриптор)

Зупиняє сервер. Блокується до вимкнення сервера.

Параметри

ручка -- [в]

Повернення

- ESP_OK: Сервер успішно зупинено
- ESP_ERR_INVALID_ARG: void аргумент
- ESP_FAIL: Помилка завершення роботи сервера

Структури

struct esp_https_server_user_cb_arg

Структура даних зворотного виклику містить дескриптор з'єднання ESP-TLS і стан з'єднання, у якому виконується зворотний виклик.

Public члени

> **httpd_ssl_user_cb_state_t user_cb_state**

Стан зворотного виклику користувача

> **esp_tls_t * tls**

Ручка підключення ESP-TLS

struct httpd_ssl_config

Структура конфігурації сервера HTTPS

Щоб ініціалізувати його, використовуйте HTTPD_SSL_CONFIG_DEFAULT().

Public члени

> **httpd_config_t httpd**

Основна конфігурація сервера HTTPD

Тут можна налаштувати такі параметри, як розмір стека завдань і пріоритет.

> ***const* uint8_t * servercert**

Сертифікат сервера

> **size_t servercert_len**

Довжина сертифіката сервера в байтах

> ***const* uint8_t * cacert_pem**

Сертифікат ЦС ((ЦС використовується для підпису клієнтів або сам сертифікат клієнта)

size_t cacert_len

Довжина сертифіката CA в байтах

***const* uint8_t * prvtkey_pem**

Приватний ключ

size_t prvtkey_len

Довжина закритого ключа в байтах

bool use_ecdsa_peripheral

Для використання закритого ключа використовуйте периферійний пристрій ECDSA

uint8_t ecdsa_key_efuse_blk

Блок efuse, де зберігається ключ ECDSA

httpd_ssl_transport_mode_t транспортний_режим

Транспортний режим (за замовчуванням безпечний)

uint16_t port_secure

Порт, який використовується, коли транспортний режим безпечний (за замовчуванням 443)

uint16_t port_insecure

Порт, який використовується, коли транспортний режим незахищений (за замовчуванням 80)

bool session_tickets

Увімкнути квитки сесії tls

bool use_secure_element

Увімкнути безпечний елемент для сеансу сервера

esp_https_server_user_cb * user_cb

Зворотний виклик користувача для esp_https_server

void * ssl_userdata

Дані користувача для додавання до контексту ssl

esp_tls_handshake_callback cert_select_cb

Зворотний виклик вибору сертифіката для використання. Зворотний виклик застосовний, лише якщо CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK увімкнено в menuconfig

***const* char * * alpn_protos**

Протоколи програм, які підтримує сервер, у порядку переваги. Використовується для узгодження під час рукостискання TLS, спочатку вибирається той, який підтримує клієнт. Структура даних має існувати так само довго, як і сам сервер https

Макроси

HTTPD_SSL_CONFIG_DEFAULT ()

За замовчуванням config struct init Примітки:

- порт встановлюється під час запуску сервера, відповідно до 'transport_mode'
- один сокет використовує ~ 40 Кб оперативної пам’яті з SSL, ми зменшуємо кількість сокетів за замовчуванням до 4
- SSL-сокети зазвичай довговічні, закриття LRU запобігає виснаженню пулу DOS
- Залежно від програми користувача може знадобитися коригування розміру стека

Визначення типів

typedef struct esp_https_server_user_cb_arg esp_https_server_user_cb_arg_t

Структура даних зворотного виклику містить дескриптор з’єднання ESP-TLS і стан з’єднання, у якому виконується зворотний виклик.

typedef esp_tls_last_error_t esp_https_server_last_error_t

typedef void esp_https_server_user_cb (esp_https_server_user_cb_arg_t * user_cb)

Прототип функції зворотного виклику. Може використовуватися для отримання інформації про з'єднання або клієнта (контекст SSL). Наприклад, сертифікат клієнта, FD сокета, стан з'єднання тощо.

Параметр user_cb

Структура даних зворотного виклику

typedef struct httpd_ssl_config httpd_ssl_config_t

Перерахування

enum esp_https_server_event_id_t

Значення:

> ***enumerator* HTTPS_SERVER_EVENT_ERROR**

Ця подія виникає, коли під час виконання виникають будь-які помилки

> ***enumerator* HTTPS_SERVER_EVENT_START**

Ця подія виникає під час запуску сервера HTTPS

> ***enumerator* HTTPS_SERVER_EVENT_ON_CONNECTED**

Після підключення сервера HTTPS до клієнта

> ***enumerator* HTTPS_SERVER_EVENT_ON_DATA**

Виникає при отриманні даних від клієнта

> ***enumerator* HTTPS_SERVER_EVENT_SENT_DATA**

Виникає, коли сервер ESP HTTPS надсилає дані клієнту

> ***enumerator* HTTPS_SERVER_EVENT_DISCONNECTED**

З'єднання було розірвано

> ***enumerator* HTTPS_SERVER_EVENT_STOP**

Ця подія виникає, коли сервер HTTPS зупинено

enum httpd_ssl_transport_mode_t

Значення:

> ***enumerator* HTTPD_SSL_TRANSPORT_SECURE**
>
> ***enumerator* HTTPD_SSL_TRANSPORT_INSECURE**

enum httpd_ssl_user_cb_state_t

Вказує стан, у якому виконується зворотний виклик користувача, тобто під час створення або закриття сеансу.

Значення:

> ***enumerator* HTTPD_SSL_USER_CB_SESS_CREATE**
>
> ***enumerator* HTTPD_SSL_USER_CB_SESS_CLOSE**

HTTP-КЛІЄНТ ESP

Огляд

esp_http_client компонент надає набір API для створення запитів HTTP/S із програм ESP-IDF. Щоб використовувати ці API, виконайте наведені нижче дії.

- **esp_http_client_init()**: Створює **esp_http_client_handle_t** екземпляр, тобто дескриптор клієнта HTTP на основі заданої **esp_http_client_config_t** конфігурації. Ця функція має бути викликана першою; значення за замовчуванням приймаються для значень конфігурації, які явно не визначені користувачем.
- **esp_http_client_perform()**: Виконує всі операції esp_http_client - відкриття з'єднання, обмін даними та закриття з'єднання (за потреби), одночасно блокуючи поточне завдання до його завершення. Усі пов’язані події викликаються через обробник подій (як зазначено в **esp_http_client_config_t**).
- **esp_http_client_cleanup()**: закриває з’єднання (якщо є) і звільняє всю пам’ять, виділену екземпляру клієнта HTTP. Це має бути остання функція, яка викликається після завершення операцій.

Постійні підключення

Постійне з’єднання означає, що клієнт HTTP може повторно використовувати те саме з’єднання для кількох обмінів. Якщо сервер не запитує закриття з’єднання за допомогою заголовка , з’єднання не розривається, а залишається відкритим і використовується для подальших запитів. Connection: close

Щоб дозволити HTTP-клієнту ESP використовувати всі переваги постійних з’єднань, потрібно зробити якомога більше запитів, використовуючи той самий екземпляр дескриптора.

Використовуйте Secure Element (ATECC608) для TLS

Захищений елемент (ATECC608) також можна використовувати для основного з’єднання TLS у підключенні клієнта HTTP.

HTTP-клієнт можна налаштувати на використання безпечного елемента наступним чином:

```
esp_http_client_config_t cfg = {
    /* other configurations options */
    .use_secure_element = true,
};
```

Запит HTTPS

HTTP-клієнт ESP підтримує з’єднання SSL за допомогою **mbedTLS** із `url` конфігурацією, яка починається зі `https` схеми або `transport_type` має значення `HTTP_TRANSPORT_OVER_SSL`. Підтримку HTTPS можна налаштувати за допомогою CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS (увімкнено за замовчуванням).

Під час надсилання запитів HTTPS, якщо потрібна перевірка сервера, додатковий кореневий сертифікат (у форматі PEM) потрібно надати учаснику `cert_pem` в `esp_http_client_config_t` конфігурації.

Користувачі також можуть використовувати для перевірки сервера за допомогою члена конфігурації . `ESP x509 Certificate Bundlecrt_bundle_attachesp_http_client_config_t`

Потік HTTP

Деяким програмам необхідно відкрити з'єднання та активно контролювати обмін даними (потік даних). У таких випадках потік заявок відрізняється від звичайних запитів. Приклад потоку наведено нижче:

- **esp_http_client_init()** : Створення дескриптора клієнта HTTP.
- esp_http_client_set_* або esp_http_client_delete_* : Змініть параметри з'єднання HTTP (необов'язково).
- **esp_http_client_open()** : Відкрийте HTTP-з'єднання з write_len параметром (довжина вмісту, який потрібно записати на сервер), встановленим write_len=0 для з'єднання лише для читання.
- **esp_http_client_write()** : запис даних на сервер максимальною довжиною, що дорівнює write_len функції **esp_http_client_open()** ; немає необхідності викликати цю функцію для write_len=0 .
- **esp_http_client_fetch_headers()** : читає заголовки відповіді HTTP-сервера після надсилання заголовків запиту та даних сервера (якщо такі є). Повертає content-length з сервера і може бути наступним **esp_http_client_get_status_code()** для отримання статусу HTTP з'єднання.
- **esp_http_client_read()** : Читання потоку HTTP.
- **esp_http_client_close()** : Закрийте з'єднання.
- **esp_http_client_cleanup()** : Звільнити виділені ресурси.

Аутентифікація HTTP

HTTP-клієнт ESP підтримує як **базову** , так і **дайджест** -автентифікацію.

- Користувачі можуть вказати ім’я користувача та пароль у `url` або `username` членах `password` конфігурації `esp_http_client_config_t` .

  Для HTTP-клієнта потрібна лише одна операція для проходження процесу автентифікації. `auth_type = HTTP_AUTH_TYPE_BASIC`
- Якщо , але в конфігурації присутні поля та , HTTP-клієнт виконує дві операції. Клієнт отримає заголовок під час першої спроби підключитися до сервера. На основі цієї інформації він вирішує, який метод автентифікації вибрати, і виконує його під час другої операції. `auth_type = HTTP_AUTH_TYPE_NONEusernamepassword`

  `401 Unauthorized`
- Наразі дайджест-автентифікація підтримує лише алгоритми MD5 і SHA-256.

Приклади конфігурації автентифікації

Автентифікація за допомогою URI

```
esp_http_client_config_t config = {
    .url = "http://user:passwd@httpbin.org/basic-auth/user/passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

Аутентифікація за допомогою введення імені користувача та пароля

esp_http_client_config_t config = {

```
    .url = "http://httpbin.org/basic-auth/user/passwd",
    .username = "user",
    .password = "passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

Обробка подій

HTTP-клієнт ESP підтримує обробку подій, запускаючи обробник події, що відповідає події, яка має місце. **esp_http_client_event_id_t** містить усі події, які можуть статися під час виконання HTTP-запиту за допомогою HTTP-клієнта ESP.

Діагностична інформація клієнта ESP HTTP

Діагностична інформація може бути корисною для розуміння проблеми. У випадку HTTP-клієнта ESP діагностичну інформацію можна зібрати, зареєструвавши обробник подій у бібліотеці Event Loop .

Цю функцію було додано з урахуванням структури ESP Insights , яка збирає діагностичну інформацію.

Однак цю функцію також можна використовувати для діагностичних цілей без будь-якої залежності від структури ESP Insights. Обробник подій можна зареєструвати в циклі подій за допомогою **esp_event_handler_register()** функції. Очікувані типи даних для різних подій клієнта HTTP в циклі подій такі:

- HTTP_EVENT_ERROR : esp_http_client_handle_t
- HTTP_EVENT_ON_CONNECTED : esp_http_client_handle_t
- HTTP_EVENT_HEADERS_SENT : esp_http_client_handle_t
- HTTP_EVENT_ON_HEADER : esp_http_client_handle_t
- HTTP_EVENT_ON_DATA : esp_http_client_on_data_t
- HTTP_EVENT_ON_FINISH : esp_http_client_handle_t
- HTTP_EVENT_DISCONNECTED : esp_http_client_handle_t
- HTTP_EVENT_REDIRECT : esp_http_client_redirect_event_data_t

Отримані **esp_http_client_handle_t** разом із даними події будуть дійсними, доки **HTTP_EVENT_DISCONNECTED** не буде отримано. Цей дескриптор було надіслано в основному для розрізнення різних підключень клієнта, і його не можна використовувати з іншою метою, оскільки він може змінюватися залежно від стану підключення клієнта.

Версія протоколу TLS

Версію протоколу TLS, яка буде використовуватися для базового підключення TLS, можна встановити в **esp_http_client_config_t**

Версію протоколу TLS для клієнта HTTP можна налаштувати таким чином:

```
#include "esp_http_client.h"
esp_http_client_config_t config = {
    .tls_version = ESP_HTTP_CLIENT_TLS_VER_TLS_1_2,
};
```

ДОВІДНИК API

Функції

esp_http_client_handle_t esp_http_client_init (const esp_http_client_config_t * config)

Розпочати сеанс HTTP. Ця функція має бути першою функцією, яка викликається, і вона повертає esp_http_client_handle_t, який ви повинні використовувати як вхідні дані для інших функцій в інтерфейсі. Цей виклик ПОВИНЕН мати відповідний виклик esp_http_client_cleanup після завершення операції.

Параметри

config -- **[in]** Конфігурації див http_client_config_t

Повернення

- esp_http_client_handle_t
- NULL, якщо є помилки

esp_err_t esp_http_client_perform (клієнт esp_http_client_handle_t)

Викличте цю функцію після того esp_http_client_init, як буде зроблено всі виклики параметрів, і виконайте передачу, як описано в параметрах.

Його потрібно викликати з тим самим esp_http_client_handle_t як вхідні дані, що й виклик esp_http_client_init. esp_http_client_perform виконує весь запит блокуючим або неблокуючим способом.

За замовчуванням API виконує запит блокуючим способом і повертає запит, коли він виконаний, або якщо він не вдався, і неблокуючим способом, він повертає, якщо зустрічається EAGAIN/EWOULDBLOCK або EINPROGRESS, або якщо це не вдалося.

А у випадку неблокуючого запиту користувач може викликати цей API кілька разів, якщо запит і відповідь не завершені або не сталася помилка. Щоб увімкнути неблокуючий esp_http_client_perform(), потрібно

встановити is_async член esp_http_client_config_t під час виклику API esp_http_client_init().

Ви можете виконувати будь-яку кількість викликів esp_http_client_perform, використовуючи той самий esp_http_client_handle_t. Основне з'єднання може залишатися відкритим, якщо це дозволяє сервер.

Якщо ви збираєтеся передати більше одного файлу, вам навіть рекомендується це зробити.

Тоді esp_http_client спробує повторно використати те саме з'єднання для наступних передач, таким чином роблячи операції швидшими, менш навантажуючими ЦП і використовуючи менше мережевих ресурсів.

Ви ніколи не повинні викликати цю функцію одночасно з двох місць, використовуючи той самий дескриптор клієнта. Нехай функція спочатку повернеться, перш ніж викликати її в інший раз.

Якщо вам потрібні паралельні передачі, ви повинні використовувати кілька esp_http_client_handle_t.

Ця функція включає esp_http_client_open -> esp_http_client_write -> esp_http_client_fetch_headers -> esp_http_client_read (і опцію) esp_http_client_close.

Параметри

клієнт -- дескриптор esp_http_client

Повернення

- ESP_OK при успішному виконанні
- ESP_FAIL через помилку

esp_err_t esp_http_client_cancel_request (клієнт esp_http_client_handle_t)

Скасувати поточний запит HTTP. Цей API закриває поточний сокет і відкриває новий сокет із тим самим контекстом esp_http_client.

Параметри

клієнт -- дескриптор esp_http_client

Повернення

- ESP_OK при успішному виконанні
- ESP_FAIL через помилку
- ESP_ERR_INVALID_ARG
- ESP_ERR_INVALID_STATE

esp_err_t esp_http_client_set_url (клієнт esp_http_client_handle_t , const char * url)

Установіть URL-адресу для клієнта, під час виконання цієї дії параметри в URL-адресі замінять старі.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **url** -- **[in]** URL-адреса

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_set_post_field (esp_http_client_handle_t клієнт, const char * data , int len)

Встановити дані публікації, цю функцію потрібно викликати до esp_http_client_perform. . Параметр даних, переданий цій функції, є вказівником, і ця функція не копіюватиме дані.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **дані** -- **[в]** покажчик даних посту
- **len** -- **[in]** довжина повідомлення

Повернення

- ESP_OK
- ESP_FAIL

int esp_http_client_get_post_field (esp_http_client_handle_t клієнт , char * * дані)

Отримати поточну інформацію про поле публікації.

Параметри

- **клієнт** -- **[в]** Клієнт
- **дані** -- **[вихід]** Вказівник на розміщення даних

Повернення

Розмір даних публікації

esp_err_t esp_http_client_set_header (клієнт esp_http_client_handle_t , const char * ключ , const char * значення)

Установіть заголовок http-запиту, цю функцію потрібно викликати після esp_http_client_init і перед будь-якою функцією виконання.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **key** -- **[in]** Ключ заголовка
- **value** -- **[in]** Значення заголовка

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_get_header (клієнт esp_http_client_handle_t , const char * ключ , char * * значення)

Отримати заголовок запиту http. Параметр значення буде встановлено на NULL, якщо немає заголовка, який би збігався з указаним ключем, інакше адресу значення заголовка буде призначено параметру значення. Цю функцію потрібно викликати після `esp_http_client_init`.

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **key -- [in]** Ключ заголовка
- **value -- [out]** Значення заголовка

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_get_username (esp_http_client_handle_t клієнт, char * * значення)

Отримати ім'я користувача для запиту http. Параметру значення буде присвоєно адресу буфера імені користувача. Цю функцію потрібно викликати після `esp_http_client_init`.

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **value -- [out]** Значення імені користувача

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_set_username (esp_http_client_handle_t клієнт, const char * ім’я користувача)

Установити ім’я користувача для запиту http. Значення параметра імені користувача буде призначено буферу імені користувача. Якщо параметр імені користувача має значення NULL, буфер імені користувача буде звільнено.

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **ім'я користувача -- [in]** Значення імені користувача

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_get_password (клієнт esp_http_client_handle_t , значення char * *)

Отримайте пароль http запиту. Параметру значення буде присвоєно адресу буфера паролів. Цю функцію потрібно викликати після esp_http_client_init.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **value** -- **[out]** Значення пароля

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_set_password (клієнт esp_http_client_handle_t , const char * пароль)

Встановіть пароль для запиту http. Значення параметра пароля буде присвоєно буферу паролів. Якщо параметр пароля NULL, буфер пароля буде звільнено.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **пароль** -- **[in]** Значення пароля

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_set_authtype (esp_http_client_handle_t клієнт , esp_http_client_auth_type_t auth_type)

Встановити http запит auth_type.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **auth_type** -- **[in]** Тип автентифікації esp_http_client

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_get_user_data (клієнт esp_http_client_handle_t , void * * дані)

Отримати http запит user_data. Значення, збережене в esp_http_client_config_t, буде записано на адресу, передану в дані.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **data** -- **[out]** Покажчик на вказівник, який буде встановлено як user_data.

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_set_user_data (клієнт esp_http_client_handle_t, void * data)

Встановити http-запит user_data. Значення, передане в +data+, буде доступним під час зворотних викликів події. Управління пам'яттю від імені користувача не виконуватиметься.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **data** -- **[in]** Покажчик на дані користувача

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

int esp_http_client_get_errno (клієнт esp_http_client_handle_t)

Отримати сеанс клієнта HTTP errno.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

- (-1) якщо void аргумент
- помилково

esp_err_t esp_http_client_set_method (клієнт esp_http_client_handle_t, м етод esp_http_client_method_t)

Установити метод запиту http.

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **метод -- [в]** Метод

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_set_timeout_ms (клієнт esp_http_client_handle _t , int timeout_ms)

Установити тайм-аут запиту http.

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **timeout_ms -- [in]** Значення часу очікування

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_delete_header (клієнт esp_http_client_handle_t, const char * key)

Видалити заголовок http-запиту.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **ключ** -- **[в]** Ключ

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_open (esp_http_client_handle_t клієнт , int write_len)

Ця функція відкриє з’єднання, запише всі рядки заголовків і повернеться.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **write_len** -- **[in]** Довжина вмісту HTTP, яку потрібно записати на сервер

Повернення

- ESP_OK
- ESP_FAIL

int esp_http_client_write (esp_http_client_handle_t клієнт , const char * буфер , int len)

Ця функція записуватиме дані до HTTP-з’єднання, яке раніше було відкрито esp_http_client_open()

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **buffer** -- Буфер
- **len -- [in]** Це значення не має бути більшим за параметр write_len, наданий esp_http_client_open()

Повернення

- (-1) якщо є помилки
- Довжина записаних даних

int64_t esp_http_client_fetch_headers (esp_http_client_handle_t клієнт)

Цю функцію потрібно викликати після esp_http_client_open, вона читатиме з http-потоку, оброблятиме всі отримані заголовки.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

- (0), якщо потік не містить заголовка довжини вмісту або фрагментованого кодування (перевіряється esp_http_client_is_chunked відповіддю)
- (-1: ESP_FAIL), якщо є помилки
- (-ESP_ERR_HTTP_EAGAIN = -0x7007) якщо виклик минув перед тим, як будь-які дані були готові
- Довжина завантажуваних даних визначається заголовком content-length

bool esp_http_client_is_chunked_response (клієнт esp_http_client_handle_t)

Дані відповіді перевірки розбиті на частини.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

Правда чи неправда

int esp_http_client_read (esp_http_client_handle_t клієнт , char * буфер , int len)

Читати дані з http-потоку.

(-ESP_ERR_HTTP_EAGAIN = -0x7007) повертається, коли виклик минув перед тим, як будь-які дані були готові

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **buffer** -- Буфер
- **len** -- **[in]** Довжина

Повернення

- (-1) якщо є помилки
- Довжина даних була прочитана

int esp_http_client_get_status_code (клієнт esp_http_client_handle_t)

Отримати код статусу відповіді http, дійсне значення, якщо ця функція буде викликана після `esp_http_client_perform`

Параметри

клієнт -- **[in]** Дескриптор esp_http_client

Повернення

Код стану

int64_t esp_http_client_get_content_length (клієнт esp_http_client_handle_t)

Отримайте дійсне значення довжини вмісту відповіді http (із заголовка Content-Length), якщо ця функція викликається після `esp_http_client_perform`

Параметри

клієнт -- **[in]** Дескриптор esp_http_client

Повернення

- (-1) Часткова передача
- Значення Content-Length у байтах

esp_err_t esp_http_client_close (клієнт esp_http_client_handle_t)

Закрити http-з’єднання, усе ще зберігає всі ресурси http-запиту.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_cleanup (клієнт esp_http_client_handle_t)

Ця функція має бути останньою функцією, яка викликається для сеансу. Це протилежність функції esp_http_client_init і має викликатися з тим самим дескриптором, що й вхідні дані, які повернув виклик esp_http_client_init.

Це може закрити всі з’єднання, які використовував цей маркер і, можливо, залишав відкритими досі.

Не викликайте цю функцію, якщо ви маєте намір передати більше файлів, повторне використання дескрипторів є ключем до високої продуктивності з esp_http_client.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

- ESP_OK
- ESP_FAIL

esp_http_client_transport_t esp_http_client_get_transport_type (esp_http_client_handle_t клієнт)

Отримати вид транспорту.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

- HTTP_TRANSPORT_UNKNOWN
- HTTP_TRANSPORT_OVER_TCP
- HTTP_TRANSPORT_OVER_SSL

esp_err_t esp_http_client_set_redirection(клієнт esp_http_client_handle_t)

Установити URL-адресу перенаправлення. Отримавши код 30x від сервера, клієнт зберігає URL-адресу перенаправлення, надану сервером.

Ця функція встановить поточну URL-адресу для переспрямування, щоб клієнт міг виконати запит на переспрямування.

Якщо disable_auto_redirect встановлено, клієнт не буде викликати цю функцію, але подію HTTP_EVENT_REDIRECT буде відправлено, надаючи користувачеві контроль над подією перенаправлення.

Параметри

клієнт -- [in] Дескриптор esp_http_client

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_reset_redirect_counter (клієнт esp_http_client_handle_t)

Скинути лічильник перенаправлення. Це корисно для скидання лічильника перенаправлення у випадках, коли той самий дескриптор використовується для кількох запитів.

Параметри

клієнт -- **[in]** Дескриптор esp_http_client

Повернення

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t esp_http_client_set_auth_data (esp_http_client_handle_t клієнт, const char * auth_data , int len)

Після отримання спеціального заголовка автентифікації цей API можна викликати для встановлення інформації автентифікації із заголовка. Цей API можна викликати з обробника подій.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **auth_data** -- **[in]** Дані автентифікації, отримані в заголовку
- **len** -- **[in]** довжина auth_data.

Повернення

- ESP_ERR_INVALID_ARG
- ESP_OK

void esp_http_client_add_auth (клієнт esp_http_client_handle_t)

Після отримання коду статусу HTTP 401 цей API можна викликати для додавання інформації авторизації.

Існує можливість отримання основного повідомлення з кодами стану перенаправлення, тому переконайтеся, що після виклику цього API скинуто основні дані.

Параметри

клієнт -- **[in]** Дескриптор esp_http_client

bool esp_http_client_is_complete_data_received (клієнт esp_http_client_handle_t)

Перевіряє, чи всі дані у відповіді було прочитано без помилок.

Параметри

клієнт -- **[in]** Дескриптор esp_http_client

Повернення

- правда
- помилковий

int esp_http_client_read_response (esp_http_client_handle_t клієнт , char * буфер , int len)

Допоміжний API для читання великих фрагментів даних. Це допоміжний API, який внутрішньо викликає esp_http_client_read багато разів, доки не буде досягнуто кінця даних або доки не заповниться буфер.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **buffer** -- Буфер
- **len** -- **[in]** Довжина буфера

Повернення

- Довжина даних була прочитана

esp_err_t esp_http_client_flush_response (esp_http_client_handle_t клієнт, int * len)

Обробити всі дані відповіді, що залишилися. Це використовує внутрішній буфер для повторного отримання, аналізу та відхилення даних відповіді, доки дані не будуть оброблені повністю.

Оскільки додатковий буфер, який надає користувач, не потрібен, це може бути кращим esp_http_client_read_responsey ситуаціях, коли вміст відповіді може ігноруватися.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **len** -- Довжина видалених даних

Повернення

- ESP_OK У разі успіху len матиме відкинуту довжину
- ESP_FAIL Якщо не вдалося прочитати відповідь
- ESP_ERR_INVALID_ARG Якщо клієнт дорівнює NULL

esp_err_t esp_http_client_get_url (esp_http_client_handle_t клієнт , char * url , const int len)

Отримати URL від клієнта.

Параметри

- **клієнт** -- **[in]** Дескриптор esp_http_client
- **url** -- **[inout]** Буфер для збереження URL-адреси
- **len** -- **[in]** Довжина буфера

Повернення

- ESP_OK
- ESP_FAIL

esp_err_t esp_http_client_get_chunk_length (esp_http_client_handle_t клієнт , int * len)

Отримайте Chunk-Length від клієнта.

Параметри

- **клієнт -- [in]** Дескриптор esp_http_client
- **len -- [out]** Змінна для збереження довжини

Повернення

- ESP_OK У разі успіху len матиме довжину поточного фрагмента
- ESP_FAIL Якщо сервер не є сервером із фрагментами
- ESP_ERR_INVALID_ARG Якщо клієнт або len мають значення NULL

Структури

struct esp_http_client_event

Дані подій клієнта HTTP.

Public члени

esp_http_client_event_id_t event_id

event_id, щоб дізнатися причину події

esp_http_client_handle_t клієнт

контекст esp_http_client_handle_t

void * дані

дані події

int data_len

довжина даних

void * user_data

контекст user_data, з esp_http_client_config_t user_data

char * ключ_заголовка

Для HTTP_EVENT_ON_HEADER event_id це поточний ключ заголовка http

char * значення_заголовка

Для HTTP_EVENT_ON_HEADER event_id це зберігає поточне значення заголовка http

struct esp_http_client_on_data

Структура аргументу для події HTTP_EVENT_ON_DATA.

Public члени

> **esp_http_client_handle_t клієнт**

Ручка клієнта

> **int64_t data_process**

Загальна кількість оброблених даних

struct esp_http_client_redirect_event_data

Структура аргументу для події HTTP_EVENT_REDIRECT.

Public члени

> **esp_http_client_handle_t клієнт**

Ручка клієнта

> **int код_статусу**

Код статусу

struct esp_http_client_config_t

Конфігурація HTTP.

Public члени

> ***const* char * url**

URL-адреса HTTP, інформація про URL-адресу є найважливішою, вона перекриває інші поля нижче, якщо такі є

> ***const* char * хост**

Домен або IP як рядок

> **int . порт**

Порт для підключення, за умовчанням залежить від esp_http_client_transport_t (80 або 443)

> ***const* char * ім'я користувача**

Використання для автентифікації Http

> ***const* char * пароль**

Використання для автентифікації Http

> **esp_http_client_auth_type_t auth_type**

Тип автентифікації Http див esp_http_client_auth_type_t

> ***const* char * url**

Шлях HTTP, якщо не встановлено, за замовчуванням /

> ***const* char * запит**

HTTP-запит

> ***const* char * cert_pem**

Сертифікація сервера SSL, формат PEM як рядок, якщо клієнт вимагає перевірити сервер

> **size_t cert_len**

Довжина буфера, на який вказує cert_pem. Може бути 0 для pem з нульовим закінченням

> ***const* char * client_cert_pem**

Сертифікація клієнта SSL, формат PEM як рядок, якщо сервер вимагає перевірити клієнта

> **size_t client_cert_len**

Довжина буфера, на який вказує client_cert_pem. Може бути 0 для pem з нульовим закінченням

> ***const* char * client_key_pem**

Ключ клієнта SSL, формат PEM як рядок, якщо сервер вимагає перевірити клієнта

> **size_t client_key_len**

Довжина буфера, на який вказує client_key_pem. Може бути 0 для pem з нульовим закінченням

> ***const* char * client_key_password**

Рядок пароля розшифровки ключа клієнта

size_t client_key_password_len

Довжина рядка пароля, на який вказує client_key_password

esp_http_client_proto_ver_t tls_version

Версія протоколу TLS підключення, наприклад, TLS 1.2, TLS 1.3 (за замовчуванням – без переваг)

***const* char * user_agent**

Рядок агента користувача для надсилання з HTTP-запитами

method esp_http_client_method_t

Метод HTTP

int timeout_ms

Час очікування мережі в мілісекундах

bool disable_auto_redirect

Вимкніть автоматичне переспрямування HTTP

int max_redirection_count

Максимальна кількість перенаправлень після отримання коду статусу перенаправлення HTTP, якщо значення за замовчуванням дорівнює нулю

int max_authorization_retries

Максимальна кількість повторів підключення після отримання неавторизованого коду статусу HTTP, використовуючи значення за замовчуванням, якщо воно дорівнює нулю. Вимикає повторну спробу авторизації, якщо -1

http_event_handle_cb обробник подій

Дескриптор події HTTP

esp_http_client_transport_t транспортний_тип

Тип транспорту HTTP див esp_http_client_transport_t

int розмір_буфера

Розмір буфера отримання HTTP

int buffer_size_tx

Розмір буфера передачі HTTP

void * user_data

Контекст HTTP user_data

bool is_async

Встановити асинхронний режим, наразі підтримується лише HTTPS

bool use_global_ca_store

Використовує глобальне ca_store для всіх з’єднань, у яких встановлено цей логічний параметр.

bool skip_cert_common_name_check

Пропустіть будь-яке поле CN сертифіката сервера

***const* char * загальне_ім'я**

Покажчик на рядок із загальною назвою сертифіката сервера. Якщо не NULL, сертифікат CN сервера має відповідати цьому імені. Якщо NULL, сертифікат CN сервера має відповідати імені хоста.

esp_err_t (* crt_bundle_attach) (void * conf)

Покажчик функції на esp_crt_bundle_attach. Вмикає використання пакета сертифікації для перевірки сервера, має бути ввімкнено в меню конфігурації

bool keep_alive_enable

Увімкнути тайм-аут підтримки активності

int keep_alive_idle

Час простою Keep-alive. За замовчуванням 5 (секунда)

int keep_alive_interval

Час інтервалу підтримки активності. За замовчуванням 5 (секунда)

int keep_alive_count

Кількість повторних спроб надсилання пакетів Keep-alive. За замовчуванням 3 рахунки

***struct* ifreq * if_name**

Назва інтерфейсу для проходження даних. Використовуйте стандартний інтерфейс без налаштування

Макроси

DEFAULT_HTTP_BUF_SIZE

ESP_ERR_HTTP_BASE

Початкова кількість кодів помилок HTTP

ESP_ERR_HTTP_MAX_REDIRECT

Помилка перевищує кількість перенаправлень HTTP

ESP_ERR_HTTP_CONNECT

Помилка відкриття HTTP-з'єднання

ESP_ERR_HTTP_WRITE_DATA

Помилка запису даних HTTP

ESP_ERR_HTTP_FETCH_HEADER

Помилка читання заголовка HTTP із сервера

ESP_ERR_HTTP_INVALID_TRANSPORT

Транспортне забезпечення схеми введення відсутнє

ESP_ERR_HTTP_CONNECTING

З'єднання HTTP ще не встановлено

ESP_ERR_HTTP_EAGAIN

Відображення errno EAGAIN на esp_err_t

ESP_ERR_HTTP_CONNECTION_CLOSED

Прочитайте FIN з однорангового пристрою, і з’єднання закрито

Визначення типів

typedef struct esp_http_client * esp_http_client_handle_t

typedef struct esp_http_client_event * esp_http_client_event_handle_t

typedef struct esp_http_client_event esp_http_client_event_t

Дані подій клієнта HTTP.

typedef struct esp_http_client_on_data esp_http_client_on_data_t

Структура аргументу для події HTTP_EVENT_ON_DATA.

typedef struct esp_http_client_redirect_event_data esp_http_client_redirect_event_data_t

Структура аргументу для події HTTP_EVENT_REDIRECT.

typedef esp_err_t (* http_event_handle_cb) (esp_http_client_event_t * evt)

Перерахування

enum esp_http_client_event_id_t

Ідентифікатор подій клієнта HTTP.

Значення:

> ***enumerator* HTTP_EVENT_ERROR**

Ця подія виникає, коли під час виконання виникають будь-які помилки

> ***enumerator* HTTP_EVENT_ON_CONNECTED**

Після підключення HTTP до сервера обмін даними не здійснювався

> ***enumerator* HTTP_EVENT_HEADERS_SENT**

Після надсилання всіх заголовків на сервер

> ***enumerator* HTTP_EVENT_HEADER_SENT**

Цей заголовок було збережено для зворотної сумісності та буде застарілим у наступних версіях esp-idf

enumerator **HTTP_EVENT_ON_HEADER**

Відбувається під час отримання кожного заголовка, надісланого з сервера

enumerator **HTTP_EVENT_ON_DATA**

Виникає під час отримання даних із сервера, можливо, кількох частин пакету

enumerator **HTTP_EVENT_ON_FINISH**

Відбувається після завершення сеансу HTTP

enumerator **HTTP_EVENT_DISCONNECTED**

З'єднання було розірвано

enumerator **HTTP_EVENT_REDIRECT**

Перехоплення перенаправлень HTTP для обробки вручну

enum esp_http_client_transport_t

Транспорт клієнта HTTP.

Значення:

enumerator **HTTP_TRANSPORT_UNKNOWN**

Невідомий

enumerator **HTTP_TRANSPORT_OVER_TCP**

Транспорт через tcp

enumerator **HTTP_TRANSPORT_OVER_SSL**

Транспорт через ssl

enum esp_http_client_proto_ver_t

Значення:

enumerator **ESP_HTTP_CLIENT_TLS_VER_ANY**

enumerator **ESP_HTTP_CLIENT_TLS_VER_TLS_1_2**

enumerator **ESP_HTTP_CLIENT_TLS_VER_TLS_1_3**

enumerator **ESP_HTTP_CLIENT_TLS_VER_MAX**

enum esp_http_client_method_t

Метод HTTP.

Значення:

***enumerator* HTTP_METHOD_GET**

Метод HTTP GET

***enumerator* HTTP_METHOD_POST**

Метод HTTP POST

***enumerator* HTTP_METHOD_PUT**

Метод HTTP PUT

***enumerator* HTTP_METHOD_PATCH**

Метод HTTP PATCH

***enumerator* HTTP_METHOD_DELETE**

Метод HTTP DELETE

***enumerator* HTTP_METHOD_HEAD**

Метод HTTP HEAD

***enumerator* HTTP_METHOD_NOTIFY**

Метод HTTP NOTIFY

***enumerator* HTTP_METHOD_SUBSCRIBE**

Метод HTTP SUBSCRIBE

***enumerator* HTTP_METHOD_UNSUBSCRIBE**

Метод HTTP UNSUBSCRIBE

***enumerator* HTTP_METHOD_OPTIONS**

Метод HTTP OPTIONS

***enumerator* HTTP_METHOD_COPY**

Метод HTTP COPY

***enumerator* HTTP_METHOD_MOVE**

Метод HTTP MOVE

***enumerator* HTTP_METHOD_LOCK**

Метод HTTP LOCK

***enumerator* HTTP_METHOD_UNLOCK**

Метод HTTP UNLOCK

enumerator **HTTP_METHOD_PROPFIND**

Метод HTTP PROPFIND

enumerator **HTTP_METHOD_PROPPATCH**

Метод HTTP PROPPATCH

enumerator **HTTP_METHOD_MKCOL**

Метод HTTP MKCOL

enumerator **HTTP_METHOD_MAX**

enum esp_http_client_auth_type_t

Тип автентифікації HTTP.

Значення:

enumerator **HTTP_AUTH_TYPE_NONE**

Немає аутентифікації

enumerator **HTTP_AUTH_TYPE_BASIC**

Базова автентифікація HTTP

enumerator **HTTP_AUTH_TYPE_DIGEST**

Аутентифікація HTTP Digest

enum HttpStatus_Code

Enum для кодів стану HTTP.

Значення:

enumerator **HttpStatus_Ok**

enumerator **HttpStatus_MultipleChoices**

enumerator **HttpStatus_MovedPermanently**

enumerator **HttpStatus_Found**

enumerator **HttpStatus_SeeOther**

enumerator **HttpStatus_TemporaryRedirect**

enumerator **HttpStatus_PermanentRedirect**

enumerator **HttpStatus_BadRequest**

> ***enumerator* HttpStatus_Unauthorized**
>
> ***enumerator* HttpStatus_Forbidden**
>
> ***enumerator* HttpStatus_NotFound**
>
> ***enumerator* HttpStatus_InternalError**

ПРОТОКОЛ ICMP

Огляд

ICMP (Internet Control Message Protocol) використовується для діагностики чи контролю або створюється у відповідь на помилки в IP-операціях.

Загальна мережева утиліта `ping` реалізована на основі пакетів ICMP із значенням поля типу 0, також називається . `Echo Reply`

Під час сеансу ping вихідний хост спочатку надсилає пакет ехо-запиту ICMP і чекає на ехо-відповідь ICMP із зазначенням певного часу.

Таким чином, він також вимірює час повернення повідомлень. Отримавши дійсну ехо-відповідь ICMP, хост-джерело генеруватиме статистичні дані про рівень каналу IP (наприклад, втрату пакетів, час, що минув тощо).

Зазвичай пристрою IoT потрібно перевірити, чи працює віддалений сервер. Пристрій має показувати попередження користувачам, коли він виходить з мережі.

Це можна досягти шляхом створення сеансу ping і періодичного надсилання або аналізу ехо-пакетів ICMP.

Щоб полегшити цю внутрішню процедуру для користувачів, ESP-IDF надає деякі готові API.

Створіть новий сеанс ping

Щоб створити сеанс ping, вам потрібно спочатку заповнити `esp_ping_config_t` структуру конфігурації, вказавши цільову IP-адресу, інтервали часу тощо. За бажанням ви також можете зареєструвати деякі функції зворотного виклику в структурі `esp_ping_callbacks_t`.

Приклад методу створення нового сеансу ping і реєстрації зворотних викликів:

```
static void test_on_ping_success(esp_ping_handle_t hdl, void *args)
{
    // optionally, get callback arguments
    // const char* str = (const char*) args;
    // printf("%s\r\n", str); // "foo"
    uint8_t ttl;
    uint16_t seqno;
    uint32_t elapsed_time, recv_len;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TTL, &ttl, sizeof(ttl));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr,
sizeof(target_addr));
    esp_ping_get_profile(hdl, ESP_PING_PROF_SIZE, &recv_len,
sizeof(recv_len));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TIMEGAP, &elapsed_time,
sizeof(elapsed_time));
    printf("%d bytes from %s icmp_seq=%d ttl=%d time=%d ms\n",
           recv_len, inet_ntoa(target_addr.u_addr.ip4), seqno, ttl,
elapsed_time);
}

static void test_on_ping_timeout(esp_ping_handle_t hdl, void *args)
{
    uint16_t seqno;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr,
sizeof(target_addr));
    printf("From %s icmp_seq=%d timeout\n",
inet_ntoa(target_addr.u_addr.ip4), seqno);
}

static void test_on_ping_end(esp_ping_handle_t hdl, void *args)
{
    uint32_t transmitted;
    uint32_t received;
    uint32_t total_time_ms;

    esp_ping_get_profile(hdl, ESP_PING_PROF_REQUEST, &transmitted,
sizeof(transmitted));
    esp_ping_get_profile(hdl, ESP_PING_PROF_REPLY, &received,
sizeof(received));
    esp_ping_get_profile(hdl, ESP_PING_PROF_DURATION, &total_time_ms,
sizeof(total_time_ms));
    printf("%d packets transmitted, %d received, time %dms\n",
transmitted, received, total_time_ms);
}

void initialize_ping()
{
```

```
    /* convert URL to IP address */
    ip_addr_t target_addr;
    struct addrinfo hint;
    struct addrinfo *res = NULL;
    memset(&hint, 0, sizeof(hint));
    memset(&target_addr, 0, sizeof(target_addr));
    getaddrinfo("www.espressif.com", NULL, &hint, &res);
    struct in_addr addr4 = ((struct sockaddr_in *) (res->ai_addr))-
>sin_addr;
    inet_addr_to_ip4addr(ip_2_ip4(&target_addr), &addr4);
    freeaddrinfo(res);

    esp_ping_config_t ping_config = ESP_PING_DEFAULT_CONFIG();
    ping_config.target_addr = target_addr;          // target IP address
    ping_config.count = ESP_PING_COUNT_INFINITE;    // ping in infinite
mode, esp_ping_stop can stop it

    /* set callback functions */
    esp_ping_callbacks_t cbs;
    cbs.on_ping_success = test_on_ping_success;
    cbs.on_ping_timeout = test_on_ping_timeout;
    cbs.on_ping_end = test_on_ping_end;
    cbs.cb_args = "foo";  // arguments that feeds to all callback
functions, can be NULL
    cbs.cb_args = eth_event_group;

    esp_ping_handle_t ping;
    esp_ping_new_session(&ping_config, &cbs, &ping);
}
```

Запуск і зупинка сеансу ping

Ви можете запускати та зупиняти сеанс ping за допомогою маркера, який повертає `esp_ping_new_session`. Зауважте, що сеанс ping не запускається автоматично після створення.

Якщо сеанс ping зупинено та перезапущено знову, порядковий номер у пакетах ICMP знову буде перераховано з нуля.

Видалити сеанс ping

Якщо сеанс ping більше не використовуватиметься, ви можете видалити його за допомогою `esp_ping_delete_session`. Під час виклику цієї

функції переконайтеся, що сеанс ping перебуває в стані зупинки (тобто ви викликали esp_ping_stop раніше або сеанс ping завершив усі процедури).

Отримати статистику виконання

Як у прикладі коду вище, ви можете викликати esp_ping_get_profile функцію зворотного виклику, щоб отримати різну статистику часу виконання сеансу ping.

ДОВІДНИК API

Функції

esp_err_t esp_ping_new_session (const esp_ping_config_t * config , const esp_ping_callbacks_t * cbs , esp_ping_handle_t * hdl_out)

Створіть сеанс ping.

Параметри

- **config** -- налаштування ping
- **cbs** -- набір функцій зворотного виклику, викликаних внутрішнім завданням ping
- **hdl_out** -- дескриптор сесії ping

Повернення

- ESP_ERR_INVALID_ARG: void параметри (наприклад, конфігурація нульова тощо)
- ESP_ERR_NO_MEM: брак пам'яті
- ESP_FAIL: інша внутрішня помилка (наприклад, помилка сокета)
- ESP_OK: успішно створено сеанс ping, користувач може взяти дескриптор ping для виконання наступних завдань

esp_err_t esp_ping_delete_session (esp_ping_handle_t hdl)

Видалити сеанс ping.

Параметри

hdl -- дескриптор сесії ping

Повернення

- ESP_ERR_INVALID_ARG: void параметри (наприклад, дескриптор ping нульовий тощо)
- ESP_OK: успішно видалити сеанс ping

esp_err_t esp_ping_start (esp_ping_handle_t hdl)

Почніть сеанс ping.

Параметри

hdl -- дескриптор сесії ping

Повернення

- ESP_ERR_INVALID_ARG: void параметри (наприклад, дескриптор ping нульовий тощо)
- ESP_OK: успішно почати сеанс ping

esp_err_t esp_ping_stop (esp_ping_handle_t hdl)

Зупиніть сеанс ping.

Параметри

hdl -- дескриптор сесії ping

Повернення

- ESP_ERR_INVALID_ARG: void параметри (наприклад, дескриптор ping нульовий тощо)
- ESP_OK: завершити сеанс ping успішно

esp_err_t esp_ping_get_profile (esp_ping_handle_t hdl , профіль esp_ping _profile_t , дані void * , розмір uint32_t)

Отримати профіль виконання сеансу ping.

Параметри

- **hdl** -- дескриптор сесії ping
- **профіль** -- вид профілю
- **дані** -- дані профілю
- **size** -- розмір даних профілю

Повернення

- ESP_ERR_INVALID_ARG: void параметри (наприклад, дескриптор ping нульовий тощо)
- ESP_ERR_INVALID_SIZE: фактичний розмір даних профілю не відповідає параметру "size"
- ESP_OK: отримати профіль успішно

Структури

struct esp_ping_callbacks_t

Тип функції зворотного виклику "ping".

Public члени

> **void * cb_args**

аргументи для функцій зворотного виклику

> **void (* on_ping_success) (esp_ping_handle_t hdl , void * args)**

Викликається внутрішнім потоком ping під час отримання пакета ехо-відповіді ICMP.

> **void (* on_ping_timeout) (esp_ping_handle_t hdl , void * args)**

Викликається внутрішнім потоком ping під час очікування пакета ехо-відповіді ICMP.

> **void (* on_ping_end) (esp_ping_handle_t hdl , void * args)**

Викликається внутрішнім потоком ping після завершення сеансу ping.

struct esp_ping_config_t

Тип конфігурації "ping".

Public члени

> **count uint32_t**

Сеанс "ping" містить процедури підрахунку

> **uint32_t interval_ms**

Мілісекунди між кожною процедурою ping

> **uint32_t timeout_ms**

Значення часу очікування (у мілісекундах) кожної процедури ping

uint32_t розмір_даних

Розмір даних поруч із заголовком пакета ICMP

int tos

Тип послуги, поле, указане в IP-заголовку

int ttl

Термін життя, поле, указане в IP-заголовку

ip_addr_t target_addr

Цільова IP-адреса, IPv4 або IPv6

uint32_t task_stack_size

Розмір стека внутрішнього завдання ping

uint32_t task_prio

Пріоритет внутрішнього завдання ping

interface uint32_t

Індекс Netif, interface=0 означає NETIF_NO_INDEX

Макроси

ESP_PING_DEFAULT_CONFIG ()

Конфігурація ping за замовчуванням.

ESP_PING_COUNT_INFINITE

Встановіть для лічильника ping значення нуль

Визначення типів

typedef void * esp_ping_handle_t

Тип дескриптора сеансу "ping".

Перерахування

enum esp_ping_profile_t

Профіль сесії ping.

Значення:

> ***enumerator* ESP_PING_PROF_SEQNO**

Порядковий номер процедури ping

> ***enumerator* ESP_PING_PROF_TOS**

Тип послуги процедури ping

> ***enumerator* ESP_PING_PROF_TTL**

Час жити для процедури ping

> ***enumerator* ESP_PING_PROF_REQUEST**

Кількість надісланих пакетів запитів

> ***enumerator* ESP_PING_PROF_REPLY**

Кількість отриманих пакетів відповіді

> ***enumerator* ESP_PING_PROF_IPADDR**

IP-адреса об’єкта відповіді

> ***enumerator* ESP_PING_PROF_SIZE**

Розмір отриманого пакета

> ***enumerator* ESP_PING_PROF_TIMEGAP**

Час, що минув між запитом і пакетом відповіді

> ***enumerator* ESP_PING_PROF_DURATION**

Минув час усього сеансу ping

СЛУЖБА MDNS

Огляд

mDNS - це багатоадресна служба UDP, яка використовується для надання послуг локальної мережі та виявлення хостів.

mDNS встановлено за замовчуванням у більшості операційних систем або доступний як окремий пакет.

Властивості mDNS

- `hostname`: ім’я хосту, на яке відповідатиме пристрій. Якщо не встановлено, `hostname` буде зчитано з interface. Приклад: `my-esp32` вирішить `my-esp32.local`
- `default_instance`: зрозуміла назва пристрою.

Приклад методу запуску mDNS для interface STA та встановлення `hostname` та `default_instance`:

```
void start_mdns_service()
{
    //initialize mDNS service
    esp_err_t err = mdns_init();
    if (err) {
        printf("MDNS Init failed: %d\n", err);
        return;
    }

    //set hostname
    mdns_hostname_set("my-esp32");
    //set default instance
    mdns_instance_name_set("Jhon's ESP32 Thing");
}
```

Послуги mDNS

- mDNS може рекламувати інформацію про мережеві послуги, які пропонує ваш пристрій. Кожна послуга визначається кількома властивостями.
- `instance_name` : зрозуміла назва вашої служби,
- `proto` : (обов'язково) протокол, на якому працює служба, із підкресленням на початку. Приклад: `_tcp` або `_udp`
- `port` : (обов'язково) мережевий порт, на якому працює служба
- `txt` : масив рядків, який використовується для визначення властивостей вашої служби `{var, val}`

У бібліотеці mDNS Espressif не підтримуються самозапити імен, це був навмисний вибір для спрощення впровадження, запобігання забрудненню локальної мережі та вирішення проблеми багатоадресної поведінки WiFi)

Встановлення власного імені хоста є передумовою (обов'язковою) для рекламних послуг або делегування інших імен.

Приклад методу додавання кількох служб і різних властивостей:

```
void add_mdns_services()
{
    //add our services
    mdns_service_add(NULL, "_http", "_tcp", 80, NULL, 0);
    mdns_service_add(NULL, "_arduino", "_tcp", 3232, NULL, 0);
    mdns_service_add(NULL, "_myservice", "_udp", 1234, NULL, 0);

    //NOTE: services must be added before their properties can be set
    //use custom instance for the web server
    mdns_service_instance_name_set("_http", "_tcp", "Jhon's ESP32 Web
Server");

    mdns_txt_item_t serviceTxtData[3] = {
        {"board","{esp32}"},
        {"u","user"},
        {"p","password"}
    };
```

```
//set txt data for service (will free and replace current data)
    mdns_service_txt_set("_http", "_tcp", serviceTxtData, 3);

    //change service port
    mdns_service_port_set("_myservice", "_udp", 4321);
}
```

Запит mDNS

mDNS надає методи для перегляду служб і визначення IP/IPv6-адрес хоста. Результати для послуг повертаються як пов’язаний список `mdns_result_t` об’єктів.

Приклад методу визначення IP-адрес хоста:

```
void resolve_mdns_host(const char * host_name)
{
    printf("Query A: %s.local", host_name);

    struct ip4_addr addr;
    addr.addr = 0;

    esp_err_t err = mdns_query_a(host_name, 2000,  &addr);
    if(err){
        if(err == ESP_ERR_NOT_FOUND){
            printf("Host was not found!");
            return;
        }
        printf("Query Failed");
        return;
    }

    printf(IPSTR, IP2STR(&addr));
}
Приклад методу вирішення локальних служб:
static const char * if_str[] = {"STA", "AP", "ETH", "MAX"};
static const char * ip_protocol_str[] = {"V4", "V6", "MAX"};

void mdns_print_results(mdns_result_t * results){
    mdns_result_t * r = results;
    mdns_ip_addr_t * a = NULL;
    int i = 1, t;
    while(r){
        printf("%d: Interface: %s, Type: %s\n", i++, if_str[r->tcpip_if],
ip_protocol_str[r->ip_protocol]);
        if(r->instance_name){
            printf("  PTR : %s\n", r->instance_name);
```

```
        }
        if(r->hostname){
            printf("  SRV : %s.local:%u\n", r->hostname, r->port);
     }
        if(r->txt_count){
            printf("  TXT : [%u] ", r->txt_count);
            for(t=0; t<r->txt_count; t++){
                printf("%s=%s; ", r->txt[t].key, r->txt[t].value);
            }
            printf("\n");
        }
        a = r->addr;
        while(a){
            if(a->addr.type == IPADDR_TYPE_V6){
                printf("  AAAA: " IPV6STR "\n", IPV62STR(a-
>addr.u_addr.ip6));
            } else {
                printf("  A   : " IPSTR "\n", IP2STR(&(a-
>addr.u_addr.ip4)));
            }
            a = a->next;
        }
        r = r->next;
    }

}

void find_mdns_service(const char * service_name, const char * proto)
{
    ESP_LOGI(TAG, "Query PTR: %s.%s.local", service_name, proto);

    mdns_result_t * results = NULL;
    esp_err_t err = mdns_query_ptr(service_name, proto, 3000, 20,
&results);
    if(err){
        ESP_LOGE(TAG, "Query Failed");
        return;
    }
    if(!results){
        ESP_LOGW(TAG, "No results found!");
        return;
    }

    mdns_print_results(results);
    mdns_query_results_free(results);
}
Приклад використання методів вище:
void my_app_some_method(){
    //search for esp32-mdns.local
    resolve_mdns_host("esp32-mdns");
```

```
    //search for HTTP servers
    find_mdns_service("_http", "_tcp");
    //or file servers
    find_mdns_service("_smb", "_tcp"); //windows sharing
    find_mdns_service("_afpovertcp", "_tcp"); //apple sharing
    find_mdns_service("_nfs", "_tcp"); //NFS server
    find_mdns_service("_ftp", "_tcp"); //FTP server
    //or networked printer
    find_mdns_service("_printer", "_tcp");
    find_mdns_service("_ipp", "_tcp");
}
```

Оптимізація продуктивності

Швидкість виконання

mDNS створює завдання з низьким пріоритетом за замовчуванням 1 `CONFIG_MDNS_TASK_PRIORITY` (якщо `CONFIG_FREERTOS_UNICORE` ввімкнено, воно закріплюється на CPU0 (`CONFIG_MDNS_TASK_AFFINITY`).

Мінімізація використання оперативної пам'яті

mDNS створює завдання з розміром стека, налаштованим `CONFIG_MDNS_TASK_STACK_SIZE`.

Приклад застосування

Приклад сервера/сканера mDNS

ДОВІДНИК API

Функції

esp_err_t mdns_init (void)

Ініціалізувати mDNS на заданому interfacei.

Повернення

- ESP_OK в разі успіху
- ESP_ERR_INVALID_STATE, коли не вдалося зареєструвати обробник подій
- ESP_ERR_NO_MEM через помилку пам’яті
- ESP_FAIL, коли не вдалося запустити завдання mdns

void mdns_free (void)

Зупинити та звільнити сервер mDNS.

esp_err_t mdns_hostname_set (const char * ім’я хоста)

Встановіть ім’я хоста для сервера mDNS, якщо ви хочете рекламувати послуги.

Параметри

ім’я хосту – ім’я хосту, яке потрібно встановити

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_hostname_get (символ * ім'я хоста)

Отримайте ім’я хоста для сервера mDNS.

Параметри

ім’я хоста – вказівник на ім’я хоста, воно має бути виділене та містити принаймні MDNS_NAME_BUF_LEN символів

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_INVALID_STATE, коли mdns не ініціалізовано

esp_err_t mdns_delegate_hostname_add (const char * ім’я хоста , const mdns_ip_addr_t * address_list)

Додає ім’я хоста та адресу для делегування На запити A/AAAA буде надана відповідь для імені хоста, і до цього хосту можна буде додати служби.

Параметри

- **ім’я хосту** – ім’я хосту, яке потрібно додати
- **список_адрес** - список IP-адрес хоста

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_delegate_hostname_set_address (const char * hostname , const mdns_ip_addr_t * address_list)

Установіть адресу делегованого імені хоста.

Параметри

- **ім’я хосту** – ім’я хосту, яке потрібно встановити
- **список_адрес** - список IP-адрес хоста

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_delegate_hostname_remove (const char * ім'я хоста)

Видалення делегованого імені хоста. Усі служби, додані до цього хосту, також буде видалено.

Параметри

ім’я хосту – ім’я хосту, яке потрібно видалити

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM

bool mdns_hostname_exists (const char * ім'я хоста)

Запит, чи було додано ім’я хоста.

Параметри

ім’я хоста – ім’я хосту для запиту

Повернення

- true Ім'я хоста було додано.
- false Ім'я хоста не було додано.

esp_err_t mdns_instance_name_set (const char * instance_name)

Встановіть назву екземпляра за замовчуванням для сервера mDNS.

Параметри

instance_name – ім’я екземпляра для встановлення

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_add (const char * ім’я_примірника , const char * тип_послуги , const char * proto , uint16_t порт , mdns_txt_item_t txt [ ] , size_t num_items)

Додайте службу до сервера mDNS.

Довжина значення текстових елементів буде автоматично визначена strlen

Параметри

- **instance_name** – ім’я екземпляра для встановлення. Якщо значення NULL, використовуватиметься ім’я глобального екземпляра або ім’я хоста. Зауважте, що параметр конфігурації MDNS_MULTIPLE_ INSTANCE потрібно ввімкнути для додавання кількох екземплярів одного типу.
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **port** – службовий порт
- **txt** – рядковий масив даних TXT (наприклад, {{“var”,”val”}, {“other”,”2”}})
- **num_items** – кількість елементів у TXT-даних

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM
- ESP_FAIL не вдалося додати службу

esp_err_t mdns_service_add_for_host (const char * instance_name , const char * service_type , const char * proto , const char * hostname , uint16_t port , mdns_txt_item_t txt [ ] , size_t num_items)

Додайте службу до сервера mDNS із делегованим іменем хоста.

Довжина значення текстових елементів буде автоматично визначена strlen

Параметри

- **instance_name** – ім’я екземпляра для встановлення. Якщо значення NULL, використовуватиметься ім’я глобального екземпляра або ім’я хоста. Зауважте, що параметр конфігурації MDNS_MULTIPLE_INSTANCE потрібно ввімкнути для додавання кількох екземплярів з однаковим типом екземпляра.
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **порт** – службовий порт
- **txt** – рядковий масив даних TXT (наприклад, {{“var”,”val”}, {“other”,”2”}})
- **num_items** – кількість елементів у TXT-даних

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- Помилка пам'яті ESP_ERR_NO_MEM
- ESP_FAIL не вдалося додати службу

bool mdns_service_exists (const char * service_type , const char * proto , const char * hostname)

Перевірте, чи додано послугу.

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, перевіряється локальне ім’я хоста.

Повернення

- правда Додано відповідний сервіс.
- false Сервіс не знайдено.

bool mdns_service_exists_with_instance (const char * екземпляр , const char * тип_послуги , const char * proto , const char * ім’я хоста)

Перевірте, чи додано послугу.

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, перевіряється локальне ім’я хоста.

Повернення

- правда Додано відповідний сервіс.
- false Сервіс не знайдено.

esp_err_t mdns_service_remove (const char * service_type , const char * proto)

Видалити службу з сервера mDNS.

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_remove_for_host (const char * екземпляр , const char * тип_послуги , const char * proto , const char * ім’я хоста)

Видалити службу з сервера mDNS з іменем хоста.

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_instance_name_set (const char * service_type , const char * proto , const char * instance_name)

Встановіть назву екземпляра для служби.

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **instance_name** – ім’я екземпляра для встановлення

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_instance_name_set_for_host (const char * instance_old , const char * service_type , const char * proto , const char * hostname , const char * instance_name)

Установіть назву екземпляра для служби з іменем хоста.

Параметри

- **instance_old** – оригінальна назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **instance_name** – ім’я екземпляра для встановлення

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_port_set (const char * service_type , const char * proto , uint16_t port)

Встановити службовий порт.

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **порт** – службовий порт

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_port_set_for_host (const char * екземпляр , const char * тип_послуги , const char * proto , const char * ім’я хоста , порт uint16_t)

Установіть порт служби з іменем хоста.

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **порт** – службовий порт

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_set (const char * service_type , const char * proto , mdns_txt_item_t txt [ ] , uint8_t num_items)

Замініть усі елементи TXT для обслуговування.

Довжина значення текстових елементів буде автоматично визначена strlen

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **txt** – масив даних TXT (наприклад, {{“var”,”val”},{“other”,”2”}})
- **num_items** – кількість елементів у TXT-даних

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_set_for_host (const char * екземпляр , const char * тип_послуги , const char * proto , const char * ім’я хоста , mdns_txt_item_t txt [ ] , uint8_t num_items)

Замініть усі елементи TXT для служби на ім’я хоста.

Довжина значення текстових елементів буде автоматично визначена strlen

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **txt** – масив даних TXT (наприклад, {{“var”,”val”},{“other”,”2”}})
- **num_items** – кількість елементів у TXT-даних

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_item_set (const char * service_type , const char * proto , const char * key , const char * value)

Встановити/додати елемент TXT для запису TXT служби.

Довжина значення буде автоматично визначена strlen

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **ключ** – ключ, який потрібно додати/оновити
- **value** – нове значення ключа

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_item_set_with_explicit_value_len (const char * service_type , const char * proto , const char * key , const char * value , uint8_t va lue_len)

Встановити/додати елемент TXT для запису TXT служби.

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **ключ** – ключ, який потрібно додати/оновити
- **value** – нове значення ключа
- **value_len** – довжина значення

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_item_set_for_host (const char * екземпляр , c onst char * тип_послуги , const char * proto , const char * ім’я хоста , const char * ключ , const char * значення)

Встановити/додати елемент TXT для запису TXT служби з іменем хоста.

Довжина значення буде автоматично визначена strlen

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **ключ** – ключ, який потрібно додати/оновити
- **value** – нове значення ключа

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_item_set_for_host_with_explicit_value_len (const char * екземпляр , const char * тип_послуги , const char * proto , const char * ім’я хоста , const char * ключ , const char * значення , uint8_t value_len)

Установити/додати елемент TXT для запису TXT служби з іменем хоста та довжиною текстового значення.

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **ключ** – ключ, який потрібно додати/оновити
- **value** – нове значення ключа
- **value_len** – довжина значення

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_item_remove (const char * service_type , const char * proto , const char * key)

Видалити елемент TXT для запису TXT служби.

Параметри

- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **ключ** – ключ, який потрібно видалити

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_txt_item_remove_for_host (const char * екземпляр , const char * тип_послуги , const char * proto , const char * ім’я хоста , const char * ключ)

Видалити елемент TXT для запису TXT служби з іменем хоста.

Параметри

- **instance** – назва екземпляра
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **ключ** – ключ, який потрібно видалити

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_subtype_add_for_host (const char * instance_name , const char * service_type , const char * proto , const char * hostname , const char * subtype)

Додати підтип для служби.

Параметри

- **instance_name** – ім’я екземпляра. Якщо NULL, буде знайдено першу службу з тим самим типом служби та протоколом.
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **hostname** – ім’я хоста служби. Якщо NULL, використовуватиметься локальне ім’я хоста.
- **підтип** – підтип, який потрібно додати.

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра
- ESP_ERR_NOT_FOUND Сервіс не знайдено
- Помилка пам'яті ESP_ERR_NO_MEM

esp_err_t mdns_service_remove_all (void)

Видалити та звільнити всі служби з сервера mDNS.

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_ARG Помилка параметра

esp_err_t mdns_query_async_delete (mdns_search_once_t * search)

Видаляє готовий запит. Телефонуйте тільки після закінчення пошуку!

Параметри

search – покажчик на об’єкт пошуку

Повернення

- ESP_OK успіх
- Пошук ESP_ERR_INVALID_STATE не завершено
- Покажчик ESP_ERR_INVALID_ARG на об’єкт пошуку має значення NULL

bool mdns_query_async_get_results (mdns_search_once_t * search , uint32_t тайм-аут , mdns_result_t * * results, uint8_t * num_results)

Отримати результати за вказівником пошуку. Результати доступні як вказівник на вихідний параметр. mdns_query_async_delete

Після завершення запиту вказівник на пошуковий об’єкт потрібно видалити.

Параметри

- **search** – покажчик на об’єкт пошуку
- **тайм-аут** – час очікування відповіді в мілісекундах
- **результати** – покажчик на результати запиту
- **num_results** – покажчик на кількість фактичних елементів результату (встановіть значення NULL, щоб ігнорувати це значення, що повертається)

Повернення

True, якщо пошук завершився до або під час очікування. False, якщо час очікування завершено

mdns_search_once_t * mdns_query_async_new (const char * name , const char * service_type , const char * proto , uint16_t type , uint32_t timeout , size_t max_results , mdns_query_notify_t сповіщувач)

Запит mDNS для хоста або служби асинхронно. Пошук необхідно перевірити на прогрес і видалити вручну!

Параметри

- **name** – екземпляр служби або ім’я хоста (NULL для запитів PTR)
- **service_type** – тип служби (_http, _arduino, _ftp тощо) (NULL для запитів на хост)
- **proto** – сервісний протокол (_tcp, _udp тощо) (NULL для запитів хосту)
- **type** – тип запиту (MDNS_TYPE_*)
- **timeout** – час у мілісекундах, протягом якого активний запит mDNS
- **max_results** – максимальна кількість результатів для збору
- **сповіщувач** – функція сповіщення, яка викликається, коли результат буде готовий, може мати значення NULL

Повернення

mdns_search_once_s покажчик на новий об’єкт пошуку, якщо запит розпочато успішно. NULL інакше.

esp_err_t mdns_query_generic (const char * name , const char * service_type , const char * proto , uint16_t type , mdns_query_transmission_type_t transmission_type , uint32_t timeout , size_t max_results , mdns_result_t * * results)

Загальний запит mDNS Усі наступні методи запиту походять від цього.

Параметри

- **name** – екземпляр служби або ім’я хоста (NULL для запитів PTR)
- **service_type** – тип служби (_http, _arduino, _ftp тощо) (NULL для запитів на хост)
- **proto** – сервісний протокол (_tcp, _udp тощо) (NULL для запитів хосту)
- **type** – тип запиту (MDNS_TYPE_*)
- **Transmission_type** – одноадресний запит або багатоадресний запит

- **time-out**– час очікування відповіді в мілісекундах.
- **max_results** – максимальна кількість результатів для збору
- **result** – покажчик на результати результатів запиту необхідно звільнити за допомогою mdns_query_results_free нижче

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Тайм-аут ESP_ERR_INVALID_ARG не вказано

esp_err_t mdns_query (const char * name , const char * service_type , const char * proto , uint16_t type , uint32_t timeout , size_t max_results , mdns_result_t * * results)

Запит mDNS для хоста або служби.

Зауважте, що запит типів PTR надсилає багатоадресний запит, усі інші типи надсилають одноадресні запити

Параметри

- **name** – екземпляр служби або ім’я хоста (NULL для запитів PTR)
- **service_type** – тип служби (_http, _arduino, _ftp тощо) (NULL для запитів на хост)
- **proto** – сервісний протокол (_tcp, _udp тощо) (NULL для запитів хосту)
- **type** – тип запиту (MDNS_TYPE_*)
- **тайм-аут** – час очікування відповіді в мілісекундах.
- **max_results** – максимальна кількість результатів для збору
- **результати** – покажчик на результати результатів запиту необхідно звільнити за допомогою mdns_query_results_free нижче

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Тайм-аут ESP_ERR_INVALID_ARG не вказано

void mdns_query_results_free (mdns_result_t * results)

Безкоштовні результати запиту.

Параметри

результати – пов’язаний список результатів, які потрібно звільнити

esp_err_t mdns_query_ptr (const char * service_type , const char * proto , uint32_t timeout , size_t max_results , mdns_result_t * * results)

Зверніться до служби mDNS.

Параметри

- **service_type** – тип служби (_http, _arduino, _ftp тощо)
- **proto** – сервісний протокол (_tcp, _udp тощо)
- **timeout** – час очікування відповіді в мілісекундах.
- **max_results** – максимальна кількість результатів для збору
- **результати** – покажчик на результати запиту

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Помилка параметра ESP_ERR_INVALID_ARG

esp_err_t mdns_query_srv (const char * instance_name , const char * service_type , const char * proto , uint32_t timeout , mdns_result_t * * result)

Запит mDNS для запису SRV.

Параметри

- **instance_name** – ім’я екземпляра служби
- **service_type** – тип служби (_http, _arduino, _ftp тощо)
- **proto** – сервісний протокол (_tcp, _udp тощо)
- **timeout** – час очікування відповіді в мілісекундах.
- **результат** – покажчик на результат запиту

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Помилка параметра ESP_ERR_INVALID_ARG

esp_err_t mdns_query_txt (const char * instance_name , const char * service_type , const char * proto , uint32_t timeout , mdns_result_t * * result)

Запит mDNS для запису TXT.

Параметри

- **instance_name** – ім’я екземпляра служби
- **service_type** – тип служби (_http, _arduino, _ftp тощо)
- **proto** – сервісний протокол (_tcp, _udp тощо)
- **timeout** – час очікування відповіді в мілісекундах.
- **результат** – покажчик на результат запиту

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Помилка параметра ESP_ERR_INVALID_ARG

esp_err_t mdns_lookup_delegated_service (const char * екземпляр , const char * тип_послуги , const char * proto , size_t max_results , mdns_result_t * * результат)

Знайдіть делеговані служби.

Параметри

- **екземпляр** – ім’я екземпляра (NULL для невизначеного екземпляра)
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **max_results** – максимальна кількість результатів для збору
- **результат** – покажчик на результат пошуку

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Помилка параметра ESP_ERR_INVALID_ARG

esp_err_t mdns_lookup_selfhosted_service (const char * екземпляр , const char * тип_послуги , const char * proto , size_t max_results , mdns_result_t * * результат)

Шукайте самостійно розміщені послуги.

Параметри

- **екземпляр** – ім’я екземпляра (NULL для невизначеного екземпляра)
- **service_type** – тип служби (_http, _ftp тощо)
- **proto** – службовий протокол (_tcp, _udp)
- **max_results** – максимальна кількість результатів для збору
- **результат** – покажчик на результат пошуку

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM

- Помилка параметра ESP_ERR_INVALID_ARG

esp_err_t mdns_query_a (const char * host_name , uint32_t timeout , esp_ip4_addr_t * addr)

Запит mDNS для запису A.

Параметри

- **host_name** – ім’я хоста для пошуку
- **timeout** – час очікування відповіді в мілісекундах.
- **addr** – покажчик на результуючу адресу IP4

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- Помилка пам'яті ESP_ERR_NO_MEM
- Помилка параметра ESP_ERR_INVALID_ARG

esp_err_t mdns_register_netif (esp_netif_t * esp_netif)

Зареєструйте настроюваний esp_netif із функцією mDNS Служба mDNS працює за замовчуванням на попередньо налаштованих interfaceax (STA, AP, ETH).

Цей API дозволяє запускати службу на будь-якому налаштованому interfacei, використовуючи стандартний драйвер WiFi або Ethernet або будь-який драйвер, визначений користувачем.

Параметри

esp_netif – вказівник на interface esp-netif

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не працює або цей netif уже зареєстровано
- ESP_ERR_NO_MEM недостатньо пам’яті для цього в interfacei в списку netif (див. CONFIG_MDNS_MAX_INTERFACES)

esp_err_t mdns_unregister_netif (esp_netif_t * esp_netif)

Скасувати реєстрацію esp-netif, уже зареєстрованого в службі mDNS.

Параметри

esp_netif – вказівник на interface esp-netif

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не запущено
- ESP_ERR_NOT_FOUND цей esp-netif не був зареєстрований у службі mDNS

esp_err_t mdns_netif_action (esp_netif_t * esp_netif , mdns_event_ actions_t event_action)

Встановіть esp_netif у потрібний стан або виконайте потрібну дію, наприклад увімкніть/вимкніть цей interface або надішліть пакети оголошень до цього netif.

- Ця функція використовується для ввімкнення (тестування, вирішення конфліктів і оголошення), оголошення або вимкнення (надіслання до побачення) служб mDNS на вказаному мережевому interfacei.
- Цю функцію необхідно викликати, якщо користувачі реєструють певний interface за допомогою mdns_register_netif(), щоб увімкнути служби mDNS на цьому interfacei.
- Цю функцію можна використовувати в обробниках подій IP/підключення для автоматичного ввімкнення/оголошення служб mDNS, коли властивості мережі змінюються, і/або вимикання їх після відключення.

Параметри

- **esp_netif** – вказівник на interface esp-netif
- **event_action** – Вимкнути/Увімкнути/Оголосити на цьому interfacei через протокол IPv4/IPv6. Дії, перераховані в типі mdns_event_actions_t.

Повернення

- ESP_OK успіх
- ESP_ERR_INVALID_STATE mDNS не працює або цей netif не зареєстровано
- Помилка пам'яті ESP_ERR_NO_MEM

mdns_browse_t * mdns_browse_new (const char * service , const char * proto , mdns_browse_notify_t сповіщувач)

Перегляньте послугу в mDNS `_service._proto`.

Параметри

- **service** – вказівник на той `_service`, який буде переглядатися.
- **proto** – покажчик на те `_proto`, що буде переглядатися.
- **сповіщувач** – зворотний виклик, який буде викликано, коли служба перегляду змінена.

Повернення

mdns_browse_t вказівник на новий об’єкт перегляду, якщо ініціювався успішно. NULL інакше.

esp_err_t mdns_browse_delete (const char * service , const char * proto)

Зупиніть `_service._proto` перегляд.

Параметри

- **сервіс** – вказівник на той `_service`, який буде переглядатися.
- **proto** – покажчик на те `_proto`, що буде переглядатися.

Повернення

- ESP_OK успіх.
- ESP_ERR_FAIL mDNS не працює або перегляд `_service._proto` не запускається.
- Помилка пам'яті ESP_ERR_NO_MEM.

Структури

struct mdns_txt_item_t

Основна структура текстового елемента mDNS Використовується в mdns_service_add()

Public члени

> ***const* char * ключ**

назва ключа елемента

> ***const* char * значення**

рядок значення елемента

struct mdns_ip_addr_s

IP-елемент пов’язаного списку запитів mDNS

Public члени

> **esp_ip_addr_t адр**

IP-адреса

struct mdns_ip_addr_s * далі

наступний IP або NULL для останнього IP у списку

struct mdns_result_s

Структура результату запиту mDNS

Public члени

struct mdns_result_s * далі

наступний результат або NULL для останнього результату в списку

> **esp_netif_t * esp_netif**

ptr до відповідного esp-netif

> **uint32_t ttl**

час жити

> **mdns_ip_protocol_t ip_protocol**

ip_protocol тип interface (v4/v6)

> **char * ім'я_примірника**

ім'я екземпляра

> **char * тип_послуги**

тип послуги

> **символ * прото**

протокол обслуговування

> **char * ім'я хоста**

ім'я хоста

> **порт uint16_t**

службовий порт

> **mdns_txt_item_t * txt**

txt запис

> **uint8_t * txt_value_len**

масив текстових значень len кожного запису

> **size_t txt_count**

кількість текстових елементів

> **mdns_ip_addr_t * адр**

пов’язаний список знайдених IP-адрес

Макроси

MDNS_TYPE_A

MDNS_TYPE_PTR

MDNS_TYPE_TXT

MDNS_TYPE_AAAA

MDNS_TYPE_SRV

MDNS_TYPE_OPT

MDNS_TYPE_NSEC

MDNS_TYPE_ANY

Визначення типів

typedef struct mdns_search_once_s mdns_search_once_t

Асинхронний дескриптор запиту.

typedef struct mdns_browse_s mdns_browse_t

Демон-дескриптор запиту.

typedef struct mdns_ip_addr_s mdns_ip_addr_t

IP-елемент пов’язаного списку запитів mDNS

typedef struct mdns_result_s mdns_result_t

Структура результату запиту mDNS

typedef void (* mdns_query_notify_t) (mdns_search_once_t * search)

typedef void (* mdns_browse_notify_t) (mdns_result_t * результат)

Перерахування

enum mdns_event_actions_t

Значення:

> ***enumerator* MDNS_EVENT_ENABLE_IP4**
>
> ***enumerator* MDNS_EVENT_ENABLE_IP6**
>
> ***enumerator* MDNS_EVENT_ANNOUNCE_IP4**
>
> ***enumerator* MDNS_EVENT_ANNOUNCE_IP6**

> ***enumerator* MDNS_EVENT_DISABLE_IP4**
>
> ***enumerator* MDNS_EVENT_DISABLE_IP6**
>
> ***enumerator* MDNS_EVENT_IP4_REVERSE_LOOKUP**
>
> ***enumerator* MDNS_EVENT_IP6_REVERSE_LOOKUP**

enum mdns_ip_protocol_t

Enum mDNS для визначення типу ip_protocol

Значення:

> ***enumerator* MDNS_IP_PROTOCOL_V4**
>
> ***enumerator* MDNS_IP_PROTOCOL_V6**
>
> ***enumerator* MDNS_IP_PROTOCOL_MAX**

enum mdns_query_transmission_type_t

Тип запиту mDNS, який буде явно встановлено як одноадресний або груповий

Значення:

> ***enumerator* MDNS_QUERY_UNICAST**
>
> ***enumerator* MDNS_QUERY_MULTICAST**

ESP-TLS

Огляд

Компонент ESP-TLS надає спрощений interface API для доступу до поширених функцій TLS. Він підтримує типові сценарії, такі як перевірка сертифікації ЦС, SNI, узгодження ALPN і неблокуюче з'єднання.

Усі конфігурації можна вказати в `esp_tls_cfg_t` структурі даних. Після завершення зв'язок TLS можна здійснювати за допомогою таких API:

- **`esp_tls_init()`**: для ініціалізації дескриптора підключення TLS.
- **`esp_tls_conn_new_sync()`**: для відкриття нового блокуючого з'єднання TLS.
- **`esp_tls_conn_new_async()`**: для відкриття нового неблокуючого підключення TLS.
- **`esp_tls_conn_read()`**: для читання з підключення.
- **`esp_tls_conn_write()`**: для запису в з'єднання.
- **`esp_tls_conn_destroy()`**: для звільнення з'єднання.

Будь-який протокол прикладного рівня, як-от HTTP1, HTTP2 тощо, може бути виконано поверх цього рівня.

Деревоподібна структура для компонента ESP-TLS

```
├── esp_tls.c
├── esp_tls.h
├── esp_tls_mbedtls.c
├── esp_tls_wolfssl.c
└── private_include
    ├── esp_tls_mbedtls.h
    └── esp_tls_wolfssl.h
```

Компонент ESP-TLS має файл esp-tls/esp_tls.h, який містить публічні заголовки API для компонента.

Всередині компонент ESP-TLS працює за допомогою MbedTLS або WolfSSL, які є бібліотеками SSL/TLS. API, специфічні для MbedTLS, присутні

в esp-tls/private_include/esp_tls_mbedtls.h , а API, специфічні для WolfSSL, присутні в esp-tls/private_include/esp_tls_wolfssl.h .

Перевірка сервера TLS

ESP-TLS надає кілька варіантів перевірки сервера TLS на стороні клієнта. Клієнт ESP-TLS може перевірити сервер шляхом перевірки сертифіката сервера однорангового вузла або за допомогою попередньо наданих ключів.

Користувач повинен вибрати лише одну з наступних опцій у **esp_tls_cfg_t** структурі перевірки TLS-сервера.

Якщо жоден параметр не вибрано, клієнт за замовчуванням поверне фатальну помилку під час налаштування підключення TLS.

- **cacert_buf** і **cacert_bytes** : сертифікат ЦС можна надати в буфері структури **esp_tls_cfg_t** . ESP-TLS використовує сертифікат CA, наявний у буфері, для перевірки сервера. **esp_tls_cfg_t** Необхідно встановити наступні змінні в структурі.
  - cacert_buf - покажчик на буфер, який містить сертифікат CA.
  - cacert_bytes - розмір сертифіката ЦС в байтах.
- **use_global_ca_store** : global_ca_store можна ініціалізувати та встановити відразу. Потім його можна використовувати для перевірки сервера на наявність усіх з’єднань ESP-TLS, встановлених у відповідній структурі. Перегляньте розділ «Довідка щодо API» нижче, щоб отримати інформацію про різні API, які використовуються для ініціалізації та налаштування . use_global_ca_store = true**esp_tls_cfg_t**global_ca_store
- **crt_bundle_attach** : ESP x509 Certificate Bundle API надає простий спосіб включити набір власних кореневих сертифікатів x509 для

перевірки сервера TLS. Більш детальну інформацію можна знайти на ESP x509 Certificate Bundle .

- **psk_hint_key** : щоб використовувати попередні спільні ключі для перевірки сервера, CONFIG_ESP_TLS_PSK_VERIFICATION має бути ввімкнено в конфігурації меню ESP-TLS. Потім вказівник на підказку PSK і ключ повинні бути надані структурі **esp_tls_cfg_t**. ESP-TLS використовуватиме PSK для перевірки сервера, лише якщо не вибрано жодної іншої опції щодо перевірки сервера.
- **пропустити перевірку сервера** : це небезпечна опція, надана в ESP-TLS для тестування. Цей параметр можна налаштувати, увімкнувши CONFIG_ESP_TLS_INSECURE та CONFIG_ESP_TLS_SKIP_SERVER_CERT_VERIFY у конфігурації меню ESP-TLS. Якщо цей параметр увімкнено, ESP-TLS пропускатиме перевірку сервера за умовчанням, якщо в структурі не вибрано жодних інших параметрів для перевірки сервера **esp_tls_cfg_t**.

Увімкнення цієї опції пов’язане з потенційним ризиком встановлення з’єднання TLS із сервером, який має підроблену особу, за умови, що сертифікат сервера не надається через API чи інші механізми, такі як ca_store тощо.

Вибор сертифіката сервера ESP-TLS

Компонент ESP-TLS надає можливість налаштувати перехоплювач вибору сертифікації сервера під час використання стека MbedTLS. Це надає можливість налаштувати та використовувати зворотний виклик вибору сертифіката під час рукостискання сервера.

Зворотний виклик допомагає вибрати сертифікат для представлення клієнту на основі розширень TLS, наданих у повідомленні привітання клієнта, наприклад ALPN і SNI. Щоб увімкнути цю функцію, увімкніть CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK у конфігурації меню ESP-TLS.

Зворотний виклик вибору сертифіката можна налаштувати в **esp_tls_cfg_t** структурі наступним чином:

```
int cert_selection_callback(mbedtls_ssl_context *ssl)
{
    /* Code that the callback should execute */
    return 0;
}

esp_tls_cfg_t cfg = {
    cert_select_cb = cert_section_callback,
};
```

Основні параметри бібліотеки SSL/TLS

Компонент ESP-TLS пропонує можливість використовувати MbedTLS або WolfSSL як базову бібліотеку SSL/TLS. За замовчуванням доступний і використовується лише MbedTLS.

Оскільки параметри бібліотеки є внутрішніми для ESP-TLS, перемикання бібліотек не змінить спеціальний код ESP-TLS для проекту.

Використання WolfSSL з ESP-IDF

Існує два способи використання WolfSSL:

1. Безпосередньо додайте WolfSSL як компонент у свій проект за допомогою таких трьох команд:
2. (First, change the directory (cd) to your project directory)
3. mkdir components
4. cd components
5. git clone --recursive https://github.com/espressif/esp-wolfssl.git
6. Додайте WolfSSL як додатковий компонент у свій проект.

- Завантажте WolfSSL за допомогою:
- git clone --recursive https://github.com/espressif/esp-wolfssl.git

- Включіть ESP-WolfSSL в ESP-IDF із налаштуваннями `EXTRA_COMPONENT_DIRS` вашого `CMakeLists.txt` проекту.

Після виконання вищевказаних кроків у вас буде можливість вибрати WolfSSL як базову бібліотеку SSL/TLS у меню конфігурації вашого проекту таким чином:

```
idf.py menuconfig > ESP-TLS > SSL/TLS Library > Mbedtls/Wolfssl
```

Порівняння між MbedTLS і WolfSSL

У наведеній нижче таблиці показано типове порівняння між WolfSSL і MbedTLS . Для MbedTLS довжина IN_CONTENT і OUT_CONTENT встановлені на 16384 байт і 4096 байт відповідно.

Власність	**WolfSSL**	**MbedTLS**
Загальна кількість використаної купи	~ 19 КБ	~ 37 Кб
Використовується стек завдань	~ 2,2 Кб	~ 3,6 Кб
Розмір контейнера	~ 858 Кб	~ 736 Кб

Ці значення можуть відрізнятися залежно від параметрів конфігурації та версії відповідних бібліотек.

ATECC608A (захищений елемент) з ESP-TLS

ESP-TLS забезпечує підтримку використання мікросхеми шифрування ATECC608A із серією SoC ESP32. Використання ATECC608A підтримується лише тоді, коли ESP-TLS використовується з MbedTLS як базовим стеком SSL/TLS. ESP-TLS за замовчуванням використовує MbedTLS як базовий стек TLS/SSL, якщо його не змінено вручну.

Мікросхема ATECC608A, підключена до ESP32, має бути вже налаштована.

Набір шифрів TLS

ESP-TLS надає можливість установити список наборів шифрів у режимі клієнта. Список наборів шифрів TLS інформує сервер про підтримувані набори шифрів для певного підключення TLS незалежно від конфігурації стеку TLS. Якщо сервер підтримує будь-який набір шифрів із цього списку, з'єднання TLS буде успішним; інакше він зазнає невдачі.

Ви можете встановити `ciphersuites_list` структуру **`esp_tls_cfg_t`** під час підключення клієнта наступним чином:

```
/* ciphersuites_list must end with 0 and must be available in the memory
scope active during the entire TLS connection */
static const int ciphersuites_list[] =
{MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, 0};
esp_tls_cfg_t cfg = {
    .ciphersuites_list = ciphersuites_list,
};
```

ESP-TLS не перевірятиме правильність `ciphersuites_list` встановленого значення, слід отримати список наборів шифрів, які підтримуються стеком TLS, і перевірити його за наданим списком.

Ця функція підтримується лише в стеку MbedTLS.

Версія протоколу TLS

ESP-TLS надає можливість встановити версію протоколу TLS для відповідного підключення TLS. Після визначення версії її слід використовувати виключно для встановлення з'єднання TLS. Це забезпечує можливість

маршрутизації різних з’єднань TLS до різних версій протоколу, наприклад TLS 1.2 і TLS 1.3 під час виконання.

На даний момент ця функція підтримується лише тоді, коли ESP-TLS використовується з MbedTLS як базовим стеком SSL/TLS.

Щоб установити версію протоколу TLS за допомогою ESP-TLS, установіть **esp_tls_cfg_t::tls_version** потрібну версію протоколу з **esp_tls_proto_ver_t**.

Якщо поле версії протоколу не встановлено, політика за замовчуванням дозволяє дозволити підключення TLS на основі вимог сервера.

З’єднання ESP-TLS можна налаштувати для використання вказаної версії протоколу наступним чином:

```
#include "esp_tls.h"
esp_tls_cfg_t cfg = {
    .tls_version = ESP_TLS_VER_TLS_1_2,
};
```

ДОВІДНИК API

Функції

esp_err_t esp_tls_cfg_server_session_tickets_init (esp_tls_cfg_server_t*cfg)

Ініціалізуйте контекст квитка TLS на стороні сервера.

Ця функція ініціалізує контекст квитка сеансу tls на стороні сервера, який містить усі необхідні структури даних для ввімкнення квитків сеансу tls відповідно до RFC5077. Використовуйте esp_tls_cfg_server_session_tickets_free, щоб звільнити дані.

Параметри

cfg -- **[in]** конфігурація сервера як esp_tls_cfg_server_t

Повернення

ESP_OK, якщо налаштування пройшло успішно ESP_ERR_INVALID_ARG, якщо контекст уже ініціалізовано ESP_ERR_NO_MEM, якщо виділення пам’яті не вдалося ESP_ERR_NOT_SUPPORTED, якщо квитки сеансу недоступні через конфігурацію збірки ESP_FAIL, якщо налаштування не вдалося

void esp_tls_cfg_server_session_tickets_free (esp_tls_cfg_server_t * cfg)

Звільніть контекст квитка TLS на стороні сервера.

Параметри

cfg -- конфігурація сервера як esp_tls_cfg_server_t

esp_tls_t * esp_tls_init (void)

Створити з'єднання TLS.

Ця функція виділяє та ініціалізує дескриптор структури esp-tls.

Повернення

tls Покажчик на esp-tls як дескриптор esp-tls у разі успішної ініціалізації, NULL у разі помилки розподілу

esp_tls_t * esp_tls_conn_http_new (const char * url , const esp_tls_cfg_t * cfg)

Створіть нове блокуюче з’єднання TLS/SSL із заданою URL-адресою «HTTP».

Цей API існує з причин зворотної сумісності. Альтернативна функція з такою ж функціональністю esp_tls_conn_http_new_sync (і її асинхронна версія esp_tls_conn_http_new_async)

Параметри

- **url** -- **[in]** url хосту.
- **cfg** -- **[в]** конфігурації TLS як esp_tls_cfg_t. Якщо ви бажаєте відкрити не-TLS-з’єднання, залиште це значення NULL. Для підключення TLS передайте вказівник на 'esp_tls_cfg_t'. Як мінімум ця структура повинна бути ініціалізована нулем.

Повернення

вказівник на esp_tls_t або NULL, якщо з’єднання не вдалося відкрити.

int esp_tls_conn_new_sync (const char * ім'я хоста , int hostlen , int порт , const esp_tls_cfg_t * cfg , esp_tls_t * tls)

Створіть нове блокуюче з’єднання TLS/SSL.

Ця функція встановлює з’єднання TLS/SSL із вказаним хостом у спосіб блокування.

Параметри

- **ім'я хоста** -- **[in]** ім'я хоста хоста.
- **hostlen** -- **[in]** Довжина імені хоста.
- **port** -- **[in]** Номер порту хосту.
- **cfg** -- **[в]** конфігурації TLS як esp_tls_cfg_t. Якщо ви бажаєте відкрити не-TLS-з’єднання, залиште це значення NULL. Для з’єднання TLS передайте покажчик на esp_tls_cfg_t. Як мінімум ця структура повинна бути ініціалізована нулем.

- **tls** -- **[in]** Покажчик на esp-tls як дескриптор esp-tls.

Повернення

- -1 Якщо не вдається встановити з'єднання.
- 1 Якщо з’єднання встановлено успішно.
- 0 Якщо стан підключення триває.

int esp_tls_conn_http_new_sync (const char * url , const esp_tls_cfg_t * cfg, esp_tls_t * tls)

Створіть нове блокуюче з’єднання TLS/SSL із заданою URL-адресою «HTTP».

Поведінка така ж, як і API esp_tls_conn_new_sync(). Однак цей API приймає URL-адресу хоста.

Параметри

- **url** -- **[in]** url хосту.
- **cfg** -- **[в]** конфігурації TLS як esp_tls_cfg_t. Якщо ви бажаєте відкрити не-TLS-з’єднання, залиште це значення NULL. Для підключення TLS передайте вказівник на 'esp_tls_cfg_t'. Як мінімум ця структура повинна бути ініціалізована нулем.
- **tls** -- **[in]** Покажчик на esp-tls як дескриптор esp-tls.

Повернення

- -1 Якщо не вдається встановити з'єднання.
- 1 Якщо з’єднання встановлено успішно.
- 0 Якщо стан підключення триває.

int esp_tls_conn_new_async (const char * ім’я хоста , int hostlen , int порт , const esp_tls_cfg_t * cfg , esp_tls_t * tls)

Створіть нове неблокуюче з’єднання TLS/SSL.

Ця функція ініціює неблокуюче з’єднання TLS/SSL із вказаним хостом, але через свою неблокуючу природу вона не чекає встановлення з’єднання.

Параметри

- **ім'я хоста** -- **[in]** ім'я хоста хоста.
- **hostlen** -- **[in]** Довжина імені хоста.
- **port** -- **[in]** Номер порту хосту.
- **cfg** -- **[в]** конфігурації TLS як esp_tls_cfg_t. non_block для члена цієї структури має бути встановлено значення true.
- **tls** -- **[in]** покажчик на esp-tls як дескриптор esp-tls.

Повернення

- -1 Якщо не вдається встановити з'єднання.
- 0 Якщо встановлення з’єднання триває.
- 1 Якщо з’єднання встановлено успішно.

int esp_tls_conn_http_new_async (const char * url , const esp_tls_cfg_t * cfg , esp_tls_t * tls)

Створіть нове неблокуюче з’єднання TLS/SSL із заданою URL-адресою «HTTP».

Поведінка така ж, як і API esp_tls_conn_new_async(). Однак цей API приймає URL-адресу хоста.

Параметри

- **url** -- **[in]** url хосту.
- **cfg** -- **[в]** конфігурації TLS як esp_tls_cfg_t.
- **tls** -- **[in]** покажчик на esp-tls як дескриптор esp-tls.

Повернення

- -1 Якщо не вдається встановити з'єднання.
- 0 Якщо встановлення з’єднання триває.
- 1 Якщо з’єднання встановлено успішно.

ssize_t esp_tls_conn_write (esp_tls_t * tls , const void * data , size_t datalen)

Записувати з буфера 'data' у вказане підключення tls.

Параметри

- **tls** -- **[in]** покажчик на esp-tls як дескриптор esp-tls.
- **data** -- **[in]** Буфер, з якого будуть записані дані.
- **datalen** -- **[in]** Довжина буфера даних.

Повернення

- >=0, якщо операція запису була успішною, повертається значення кількості байтів, фактично записаних у з'єднання TLS/SSL.
- <0, якщо операція запису була невдалою, оскільки або сталася помилка, або процес, що викликає, повинен виконати дію.
- ESP_TLS_ERR_SSL_WANT_READ/ ESP_TLS_ERR_SSL_WANT_WRITE. якщо рукостискання є неповним і очікує, поки дані стануть доступними для читання. У цьому випадку цю функцію потрібно викликати знову, коли основний транспорт буде готовий до роботи.

ssize_t esp_tls_conn_read (esp_tls_t * tls , void * data , size_t datalen)

Читання з указаного з'єднання tls у буфер даних.

Параметри

- **tls** -- **[in]** покажчик на esp-tls як дескриптор esp-tls.
- **дані** -- **[в]** Буфер для зберігання прочитаних даних.
- **datalen** -- **[in]** Довжина буфера даних.

Повернення

- >0, якщо операція читання була успішною, повертається значення кількості байтів, фактично прочитаних із з'єднання TLS/SSL.
- 0, якщо операція читання не була успішною. Базове підключення було закрито.
- <0, якщо операція читання не була успішною, тому що або сталася помилка, або процес, що викликає, повинен виконати дію.

int esp_tls_conn_destroy (esp_tls_t * tls)

Закрийте з’єднання TLS/SSL і звільніть усі виділені ресурси.

Цю функцію слід викликати, щоб закрити кожне з’єднання tls, відкрите за допомогою API esp_tls_conn_new_sync() (або esp_tls_conn_http_new_sync()) і esp_tls_conn_new_async() (або esp_tls_conn_http_new_async()).

Параметри

tls -- **[in]** покажчик на esp-tls як дескриптор esp-tls.

Повернення

- 0 за успішність
- -1, якщо помилка сокета або void аргумент

ssize_t esp_tls_get_bytes_avail (esp_tls_t * tls)

Повертає кількість байтів даних програми, які залишилися для читання з поточного запису.

Цей API є оболонкою над API mbedtls_ssl_get_bytes_avail() mbedtls.

Параметри

tls -- **[in]** покажчик на esp-tls як дескриптор esp-tls.

Повернення

- -1 у разі недійсного аргументу
- байтів, доступних у буфері читання запису даних програми

esp_err_t esp_tls_get_conn_sockfd (esp_tls_t * tls , int * sockfd)

Повертає дескриптор файлу сокета підключення із сеансу esp_tls.

Параметри

- **tls** -- **[in]** дескриптор контексту esp_tls
- **sockfd** -- **[out]** int покажчик на значення sockfd.

Повернення

- ESP_OK у разі успіху та значення sockfd буде оновлено дескриптором файлу сокета для підключення
- ESP_ERR_INVALID_ARG if (tls == NULL || sockfd == NULL)

esp_err_t esp_tls_set_conn_sockfd (esp_tls_t * tls , int sockfd)

Встановлює дескриптор файлу сокета підключення для сеансу esp_tls.

Параметри

- **tls** -- **[in]** дескриптор контексту esp_tls
- **sockfd** -- **[in]** значення sockfd для встановлення.

Повернення

- ESP_OK у разі успіху та значення sockfd для з'єднання tls має оновлюватися за допомогою наданого значення
- ESP_ERR_INVALID_ARG if (tls == NULL || sockfd < 0)

esp_err_t esp_tls_get_conn_state (esp_tls_t * tls , esp_tls_conn_state_t * conn_state)

Отримує стан з'єднання для сеансу esp_tls.

Параметри

- **tls** -- **[in]** дескриптор контексту esp_tls
- **conn_state** -- **[out]** покажчик на значення стану підключення.

Повернення

- ESP_OK у разі успіху та значення sockfd для з'єднання tls має оновлюватися за допомогою наданого значення
- ESP_ERR_INVALID_ARG (void аргументи)

esp_err_t esp_tls_set_conn_state (esp_tls_t * tls , esp_tls_conn_state_t conn_state)

Встановлює стан з'єднання для сеансу esp_tls.

Параметри

- **tls** -- **[in]** дескриптор контексту esp_tls
- **conn_state** -- **[in]** значення стану з'єднання, яке потрібно встановити.

Повернення

- ESP_OK у разі успіху та значення sockfd для з'єднання tls має оновлюватися за допомогою наданого значення

- ESP_ERR_INVALID_ARG (void аргументи)

void * esp_tls_get_ssl_context (esp_tls_t * tls)

Повертає контекст ssl.

Параметри

tls -- [in] дескриптор контексту esp_tls

Повернення

- покажчик ssl_ctx на контекст ssl базового рівня TLS у разі успіху

- NULL у разі помилки

esp_err_t esp_tls_init_global_ca_store (void)

Створіть глобальне сховище ЦС, спочатку порожнє.

Цю функцію слід викликати, якщо програма хоче використовувати те саме сховище ЦС для кількох з'єднань. Ця функція ініціалізує глобальне сховище ЦС, яке потім можна встановити, викликавши esp_tls_set_global_ca_store(). Щоб бути ефективною, цю функцію потрібно викликати перед будь-яким викликом esp_tls_set_global_ca_store().

Повернення

- ESP_OK, якщо створення глобального сховища CA було успішним.
- ESP_ERR_NO_MEM, якщо сталася помилка під час розподілу ресурсів mbedTLS.

esp_err_t esp_tls_set_global_ca_store (const unsigned char * cacert_pem_buf , const unsigned int cacert_pem_bytes)

Налаштуйте глобальне сховище ЦС із буфером, наданим у форматі pem.

Цю функцію слід викликати, якщо програма хоче встановити глобальне сховище ЦС для кількох з'єднань, тобто додати сертифікати у наданому буфері до ланцюжка сертифікатів.

Ця функція неявно викликає esp_tls_init_global_ca_store(), якщо вона ще не була викликана. Програма повинна викликати цю функцію перед викликом esp_tls_conn_new().

Параметри

- **cacert_pem_buf -- [in]** Буфер, який містить сертифікати у форматі pem. Цей буфер використовується для створення глобального сховища CA, яке може використовуватися іншими підключеннями tls.
- **cacert_pem_bytes -- [in]** Довжина буфера.

Повернення

- ESP_OK, якщо додавання сертифікатів пройшло успішно.
- Інше, якщо сталася помилка або процес виклику повинен виконати дію.

void esp_tls_free_global_ca_store (void)

Звільніть глобальний магазин CA, який зараз використовується.

Пам’ять, яка використовується глобальним сховищем ЦС для зберігання всіх проаналізованих сертифікатів, звільняється.

Програма може викликати цей API, якщо їй більше не потрібне глобальне сховище ЦС.

esp_err_t esp_tls_get_and_clear_last_error (esp_tls_error_handle_t h , int * esp_tls_code , int * esp_tls_flags)

Повертає останню помилку в esp_tls із детальними кодами помилок, пов’язаних з mbedtls. Інформація про помилку очищається внутрішньо після повернення.

Параметри

- **h -- [in]** дескриптор помилки esp-tls.
- **esp_tls_code -- [out]** останній код помилки, повернутий mbedtls api (встановлюється на нуль, якщо немає) Цей вказівник може мати значення NULL, якщо абонента не хвилює esp_tls_code
- **esp_tls_flags -- [out]** останні прапорці перевірки сертифікації (встановлюється на нуль, якщо немає) Цей покажчик може мати значення NULL, якщо абонента не цікавить esp_tls_code

Повернення

- ESP_ERR_INVALID_STATE, якщо параметри void
- ESP_OK (0), якщо помилки не сталося
- конкретний код помилки (на основі ESP_ERR_ESP_TLS_BASE) інакше

esp_err_t esp_tls_get_and_clear_error_type (esp_tls_error_handle_t h , esp_tls_error_type_t err_type , int * error_code)

Повертає останню помилку, зафіксовану в esp_tls певного типу. Інформація про помилку очищається внутрішньо після повернення.

Параметри

- **h** -- **[in]** дескриптор помилки esp-tls.
- **err_type** -- **[in]** певний тип помилки
- **error_code** -- **[out]** останній код помилки, повернутий mbedtls api (встановлюється на нуль, якщо немає) Цей вказівник може мати значення NULL, якщо абонента не хвилює esp_tls_code

Повернення

- ESP_ERR_INVALID_STATE, якщо параметри void
- ESP_OK, якщо дійсна помилка повернулася та була очищена

esp_err_t esp_tls_get_error_handle (esp_tls_t * tls , esp_tls_error_handle_t * error_handle)

Повертає ESP-TLS error_handle.

Параметри

- **tls** -- **[in]** дескриптор контексту esp_tls
- **error_handle** -- **[out]** покажчик на дескриптор помилки.

Повернення

- ESP_OK у разі успіху та error_handle буде оновлено за допомогою опису помилки ESP-TLS.
- ESP_ERR_INVALID_ARG if (tls == NULL || error_handle == NULL)

mbedtls_x509_crt * esp_tls_get_global_ca_store (void)

Отримайте вказівник на глобальне сховище ЦС, яке зараз використовується.

Програма повинна спочатку викликати esp_tls_set_global_ca_store(). Тоді те саме сховище ЦС може використовуватися програмою для API, відмінних від esp_tls.

Зміна вказівника може призвести до збою під час перевірки сертифікатів.

Повернення

- Покажчик на глобальне сховище ЦС, яке зараз використовується, якщо успішно.
- NULL, якщо немає глобального набору сховищ CA.

const int * esp_tls_get_ciphersuites_list (void)

Отримайте список підтримуваних наборів шифрів TLS.

Повернення

Покажчик на масив ідентифікаторів IANA наборів шифрів TLS із закінченням нулем.

int esp_tls_server_session_create (esp_tls_cfg_server_t * cfg , int sockfd , esp_tls_t * tls)

Створіть сеанс сервера TLS/SSL.

Ця функція створює контекст сервера TLS/SSL для вже прийнятого підключення клієнта та виконує TLS/SSL рукостискання з клієнтом

Параметри

- **cfg -- [in]** Покажчик на esp_tls_cfg_server_t
- **sockfd -- [in]** FD прийнятого з'єднання
- **tls -- [out]** Покажчик на виділений esp_tls_t

Повернення

- 0 у разі успіху
- <0 у разі помилки

void esp_tls_server_session_delete (esp_tls_t * tls)

Закрийте TLS/SSL-з’єднання на стороні сервера та звільніть усі виділені ресурси.

Цю функцію слід викликати, щоб закрити кожне tls-з’єднання, відкрите за допомогою esp_tls_server_session_create()

Параметри

tls -- **[in]** покажчик на esp_tls_t

esp_err_t esp_tls_plain_tcp_connect (const char * host , int hostlen , int port , const esp_tls_cfg_t * cfg , esp_tls_error_handle_t error_handle, int * sockfd)

Створює звичайне з’єднання TCP, повертаючи дійсний fd сокета в разі успіху або дескриптора помилки.

Параметри

- **хост** -- **[in]** ім'я хосту.
- **hostlen** -- **[in]** Довжина імені хоста.
- **port** -- **[in]** Номер порту хосту.
- **cfg** -- **[in]** Конфігурація ESP-TLS як esp_tls_cfg_t.
- **error_handle** -- **[out]** дескриптор помилки ESP-TLS, що містить потенційні помилки під час підключення
- **sockfd** -- **[out]** Дескриптор сокета, якщо успішно підключено на рівні TCP

Повернення

ESP_OK у разі успіху ESP_ERR_INVALID_ARG, якщо void вихідні параметри Коди помилок на основі ESP-TLS у разі невдачі

Структури

struct psk_key_hint

Спільний ключ і структура підказок ESP-TLS.

Public члени

> ***const* uint8_t * ключ**

ключ у режимі автентифікації PSK у двійковому форматі

> ***const* size_t розмір_ключа**

довжина ключа

> ***const* char * підказка**

підказка в режимі автентифікації PSK у форматі рядка

struct tls_keep_alive_cfg

esp-tls квиток сеансу клієнта ctx

Підтримуйте структуру параметрів

Public члени

> **bool keep_alive_enable**

Увімкнути тайм-аут підтримки активності

> **int keep_alive_idle**

Час простою Keep-alive (секунди)

> **int keep_alive_interval**

Час інтервалу підтримки активності (секунди)

> **int keep_alive_count**

Кількість повторних спроб надсилання пакетів Keep-alive

struct esp_tls_cfg

Параметри конфігурації ESP-TLS.

- щодо формату сертифікатів:
- Ця структура включає сертифікати центру сертифікації, клієнта або сервера, а також закриті ключі, які можуть мати формат PEM або DER. У випадку формату PEM буфер має завершуватися NULL (з символом NULL, що входить до розміру сертифіката).
- Сертифікат центру сертифікації може бути ланцюжком сертифікатів у випадку формату PEM, але може бути лише одним сертифікатом у випадку формату DER
- Назви змінних сертифікатів і буферів приватних ключів, а також розміри визначаються як union, що забезпечує зворотну сумісність для застарілих імен *_pem_buf і *_pem_bytes, які передбачають підтримку лише формату PEM. Рекомендується використовувати загальні імена, такі як cacert_buf і cacert_bytes.

Public члени

> ***const* char * * alpn_protos**

Протоколи програми, необхідні для HTTP2. Якщо потрібна підтримка HTTP2/ALPN, список протоколів, які слід узгодити. Формат - це довжина, за якою слідує назва протоколу. Для найпоширеніших випадків допустимо наступне: const char **alpn_protos = { "h2", NULL };

де 'h2' - назва протоколу

> ***const* unsigned char * cacert_buf**

Сертифікат центру сертифікації в буфері. Формат може бути PEM або DER, залежно від підтримки mbedtls Цей буфер має завершуватися NULL у випадку PEM

> ***const* unsigned char * cacert_pem_buf**

Застаріла назва буфера сертифіката ЦС

unsigned int cacert_bytes

Розмір сертифіката центру сертифікації, на який вказує cacert_buf (включаючи NULL-термінатор у випадку формату PEM)

unsigned int cacert_pem_bytes

Розмір назви застарілого сертифіката центру сертифікації

***const* unsigned char * clientcert_buf**

Сертифікат клієнта у форматі буфера може бути PEM або DER, залежно від підтримки mbedtls. Цей буфер має завершуватися NULL у випадку PEM

***const* unsigned char * clientcert_pem_buf**

Застаріла назва сертифіката клієнта

unsigned int clientcert_bytes

Розмір сертифіката клієнта, на який вказує clientcert_pem_buf (включаючи NULL-термінатор у випадку формату PEM)

unsigned int clientcert_pem_bytes

Розмір застарілої назви сертифіката клієнта

***const* unsigned char * clientkey_buf**

Клієнтський ключ у буфері. Формат може бути PEM або DER, залежно від підтримки mbedtls. У випадку PEM цей буфер має завершуватися NULL.

***const* unsigned char * clientkey_pem_buf**

Застаріла назва ключа клієнта

unsigned int clientkey_bytes

Розмір клієнтського ключа, на який вказує clientkey_pem_buf (включаючи NULL-термінатор у випадку формату PEM)

unsigned int clientkey_pem_bytes

Розмір застарілої назви ключа клієнта

***const* unsigned char * clientkey_password**

Рядок пароля розшифровки ключа клієнта

unsigned int clientkey_password_len

Довжина рядка пароля, на який вказує clientkey_password

bool use_ecdsa_peripheral

Використовуйте периферійний пристрій ECDSA для операцій із закритим ключем

uint8_t ecdsa_key_efuse_blk

Блок efuse, де зберігається ключ ECDSA

bool non_block

Налаштувати неблокуючий режим. Якщо встановлено значення true, нижній сокет буде налаштовано в режимі без блокування після встановлення сеансу tls

bool use_secure_element

Увімкніть цей параметр, щоб використовувати безпечний елемент або чіп atecc608a

int timeout_ms

Час очікування мережі в мілісекундах. . Якщо це значення не встановлено, за замовчуванням час очікування становить 10 секунд. Якщо ви бажаєте, щоб сеанс чекав нескінченно, використовуйте більше значення, наприклад, INT32_MAX

bool use_global_ca_store

Використовуйте глобальне ca_store для всіх з’єднань, у яких встановлено цей логічний параметр.

***const* char * загальне_ім'я**

Якщо не NULL, сертифікат сервера CN має відповідати цьому імені. Якщо NULL, сертифікат сервера CN має відповідати імені хоста.

bool skip_common_name

Пропустіть будь-яке поле CN сертифіката сервера

tls_keep_alive_cfg_t * keep_alive_cfg

Увімкніть тайм-аут підтримки TCP для підключення SSL

***const* psk_hint_key_t * psk_hint_key**

Покажчик на підказку та ключ PSK. якщо не NULL (і сертифікати мають NULL), тоді автентифікація PSK увімкнена з налаштованим

налаштуванням. Важливе зауваження: покажчик має бути дійсним для підключення

esp_err_t (* crt_bundle_attach) (void * conf)

Покажчик функції на esp_crt_bundle_attach. Вмикає використання пакета сертифікації для перевірки сервера, має бути ввімкнено в меню конфігурації

void * ds_data

Покажчик на периферійний контекст цифрового підпису

bool is_plain_tcp

Використовувати з’єднання без TLS: якщо встановлено значення true, esp-tls використовує звичайний транспорт TCP, а не з’єднання TLS/SSL. Зауважте, що можна підключитися за допомогою простого протоколу tcp безпосередньо за допомогою API esp_tls_plain_tcp_connect()

***struct* ifreq * if_name**

Назва interface для проходження даних. Використовуйте стандартний interface без налаштування

esp_tls_addr_family_t addr_family

Сімейство адрес для використання під час підключення до хосту.

***const* int * ciphersuites_list**

Покажчик на масив ідентифікаторів IANA наборів шифрів TLS із закінченням нулем. Перевірте дійсність списку за допомогою API esp_tls_get_ciphersuites_list().

esp_tls_proto_ver_t tls_version

Версія протоколу TLS підключення, наприклад, TLS 1.2, TLS 1.3 (за замовчуванням – без переваг)

struct esp_tls_cfg_server

Параметри конфігурації сервера ESP-TLS.

Public члени

> ***const* char * * alpn_protos**

Протоколи програми, необхідні для HTTP2. Якщо потрібна підтримка HTTP2/ALPN, список протоколів, які слід узгодити. Формат - це довжина, за якою слідує назва протоколу. Для найпоширеніших випадків допустимо наступне: const char **alpn_protos = { "h2", NULL };

- де 'h2' - назва протоколу

> ***const* unsigned char * cacert_buf**

Сертифікат ЦС клієнта в буфері. Цей буфер повинен завершуватися значенням NULL

> ***const* unsigned char * cacert_pem_buf**

Застаріла назва сертифіката ЦС клієнта

> **unsigned int cacert_bytes**

Розмір клієнтського сертифіката ЦС, на який вказує cacert_pem_buf

> **unsigned int cacert_pem_bytes**

Розмір старого імені сертифіката ЦС клієнта

> ***const* unsigned char * servercert_buf**

Сертифікат сервера в буфері Цей буфер має завершуватися значенням NULL

> ***const* unsigned char * servercert_pem_buf**

Застаріла назва сертифіката сервера

> **unsigned int servercert_bytes**

Розмір сертифіката сервера, на який вказує servercert_pem_buf

> **unsigned int servercert_pem_bytes**

Розмір застарілої назви сертифіката сервера

> ***const* unsigned char * serverkey_buf**

Ключ сервера в буфері Цей буфер має завершуватися значенням NULL

> ***const* unsigned char * serverkey_pem_buf**

Застаріла назва ключа сервера

unsigned int serverkey_bytes

Розмір ключа сервера, на який вказує serverkey_pem_buf

unsigned int serverkey_pem_bytes

Розмір застарілої назви ключа сервера

***const* unsigned char * serverkey_password**

Рядок пароля розшифровки ключа сервера

unsigned int serverkey_password_len

Довжина рядка пароля, на який вказує serverkey_password

bool use_ecdsa_peripheral

Для використання закритого ключа використовуйте периферійний пристрій ECDSA

uint8_t ecdsa_key_efuse_blk

Блок efuse, де зберігається ключ ECDSA

bool use_secure_element

Увімкніть цей параметр, щоб використовувати безпечний елемент або чіп atecc608a

void * дані користувача

Дані користувача, які потрібно додати до контексту ssl. Можна отримати за допомогою зворотних викликів

Визначення типів

typedef enum esp_tls_conn_state esp_tls_conn_state_t

Стан підключення ESP-TLS.

typedef enum esp_tls_role esp_tls_role_t

typedef struct psk_key_hint psk_hint_key_t

Спільний ключ і структура підказок ESP-TLS.

typedef struct tls_keep_alive_cfg tls_keep_alive_cfg_t

esp-tls квиток сеансу клієнта ctx

Підтримуйте структуру параметрів

typedef enum esp_tls_addr_family esp_tls_addr_family_t

typedef struct esp_tls_cfg esp_tls_cfg_t

Параметри конфігурації ESP-TLS.

Щодо формату сертифікатів:

Ця структура включає сертифікати центру сертифікації, клієнта або сервера, а також закриті ключі, які можуть мати формат PEM або DER. У випадку формату PEM буфер має завершуватися NULL (з символом NULL, що входить до розміру сертифіката).

Сертифікат центру сертифікації може бути ланцюжком сертифікатів у випадку формату PEM, але може бути лише одним сертифікатом у випадку формату DER

Назви змінних сертифікатів і буферів приватних ключів, а також розміри визначаються як union, що забезпечує зворотну сумісність для застарілих імен *_pem_buf і *_pem_bytes, які передбачають підтримку лише формату PEM. Рекомендується використовувати загальні імена, такі як cacert_buf і cacert_bytes.

typedef void * esp_tls_handshake_callback

typedef struct esp_tls_cfg_server esp_tls_cfg_server_t

Параметри конфігурації сервера ESP-TLS.

typedef struct esp_tls esp_tls_t

Enums

enum esp_tls_conn_state

Стан підключення ESP-TLS.

Значення:

> ***enumerator* ESP_TLS_INIT**
>
> ***enumerator* ESP_TLS_CONNECTING**
>
> ***enumerator* ESP_TLS_HANDSHAKE**
>
> ***enumerator* ESP_TLS_FAIL**
>
> ***enumerator* ESP_TLS_DONE**

enum esp_tls_role

Значення:

> ***enumerator* ESP_TLS_CLIENT**
>
> ***enumerator* ESP_TLS_SERVER**

enum esp_tls_addr_family

Значення:

> ***enumerator* ESP_TLS_AF_UNSPEC**

Невизначена родина адрес.

> ***enumerator* ESP_TLS_AF_INET**

Сімейство адрес IPv4.

> ***enumerator* ESP_TLS_AF_INET6**

Сімейство адрес IPv6.

enum esp_tls_proto_ver_t

Значення:

> ***enumerator* ESP_TLS_VER_ANY**
>
> ***enumerator* ESP_TLS_VER_TLS_1_2**
>
> ***enumerator* ESP_TLS_VER_TLS_1_3**
>
> ***enumerator* ESP_TLS_VER_TLS_MAX**

Файл заголовка

#include *"esp_tls_errors.h"*

- Цей файл заголовка є частиною API, що надається компонентом `esp-tls`.

  Щоб оголосити, що ваш компонент залежить від `esp-tls`, додайте наступне до свого CMakeLists.txt:

  REQUIRES esp-tls

  або

  PRIV_REQUIRES esp-tls

Структури

struct esp_tls_last_error

Структура помилок, що містить відповідні помилки у випадку виникнення помилки tls.

Public члени

> **esp_err_t остання_помилка**

код помилки (на основі ESP_ERR_ESP_TLS_BASE) останньої помилки, що сталася

> **int esp_tls_error_code**

код помилки esp_tls з останнього невдалого API esp_tls

> **int esp_tls_flags**

позначки перевірки останньої сертифікації

Макроси

ESP_ERR_ESP_TLS_BASE

Початкова кількість кодів помилок ESP-TLS

ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME

Помилка, якщо ім’я хосту не вдалося розпізнати під час з’єднання tls

ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET

Не вдалося створити сокет

ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY

Сімейство протоколів не підтримується

ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST

Не вдалося підключитися до хосту

ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED

не вдалося встановити/отримати параметр сокета

ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT

нове підключення в esp_tls_low_level_conn тайм-аут підключення

ESP_ERR_ESP_TLS_SE_FAILED

ESP_ERR_ESP_TLS_TCP_CLOSED_FIN

ESP_ERR_MBEDTLS_CERT_PARTLY_OK

Розбір сертифікатів mbedtls був частково успішним

ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_SETUP_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_WRITE_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED

mbedtls API повернув помилку

ESP_ERR_MBEDTLS_SSL_TICKET_SETUP_FAILED

mbedtls API повернув помилку

ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_CERT_VERIFY_SETUP_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_KEY_VERIFY_SETUP_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_CTX_SETUP_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_SSL_SETUP_FAILED

API wolfSSL повернув помилку

ESP_ERR_WOLFSSL_SSL_WRITE_FAILED

API wolfSSL повернув помилку

ESP_TLS_ERR_SSL_WANT_READ

Визначення помилок, про які повідомляє API вводу-виводу (потенційно неблокуючих) у разі помилки:

- esp_tls_conn_read()
- esp_tls_conn_write()

ESP_TLS_ERR_SSL_WANT_WRITE

ESP_TLS_ERR_SSL_TIMEOUT

Визначення типів

typedef struct esp_tls_last_error * esp_tls_error_handle_t

typedef struct esp_tls_last_error esp_tls_last_error_t

Структура помилок, що містить відповідні помилки у випадку виникнення помилки tls.

Перерахування

enum esp_tls_error_type_t

Визначення різних типів/джерел кодів помилок, які повідомляються з різних компонентів

Значення:

> ***enumerator*** **ESP_TLS_ERR_TYPE_UNKNOWN**
>
> ***enumerator*** **ESP_TLS_ERR_TYPE_SYSTEM**

Системна помилка – errno

> ***enumerator*** **ESP_TLS_ERR_TYPE_MBEDTLS**

Код помилки з бібліотеки mbedTLS

> ***enumerator*** **ESP_TLS_ERR_TYPE_MBEDTLS_CERT_FLAGS**

Прапори сертифікатів, визначені в mbedTLS

> ***enumerator*** **ESP_TLS_ERR_TYPE_ESP**

Тип помилки ESP-IDF – esp_err_t

> ***enumerator*** **ESP_TLS_ERR_TYPE_WOLFSSL**

Код помилки з бібліотеки wolfSSL

> ***enumerator*** **ESP_TLS_ERR_TYPE_WOLFSSL_CERT_FLAGS**

Прапори сертифікатів, визначені в wolfSSL

> ***enumerator*** **ESP_TLS_ERR_TYPE_MAX**

Тип останньої помилки – неправильний запис

ESP-MODBUS

Огляд

Бібліотека Espressif ESP-Modbus (esp-modbus) - це бібліотека для підтримки зв’язку Modbus у мережах на основі interfaceів RS485 або Ethernet. Modbus - це протокол передачі даних, спочатку опублікований Modicon (нині Schneider Electric) у 1979 році для використання з її програмованими логічними контролерами (PLC).

Протокол послідовного зв'язку Modbus є де-факто стандартним протоколом, який широко використовується для підключення промислових електронних пристроїв.

Modbus дозволяє спілкуватися між багатьма пристроями, підключеними до однієї мережі, наприклад, системою, яка вимірює температуру та вологість і передає результати на комп’ютер.

Протокол Modbus використовує кілька типів даних: регістри зберігання, вхідні регістри, котушки (однобітовий вихід), дискретні входи.

Існують версії протоколу Modbus для послідовного порту, Ethernet та інших протоколів, які підтримують набір протоколів Інтернету.

Існує багато варіантів протоколів Modbus, деякі з них:

- Modbus RTU - Це використовується в послідовному зв’язку та використовує компактне двійкове представлення даних для зв’язку протоколу.

  Формат RTU слідує за командами/даними з контрольною сумою циклічної надлишковості як механізм перевірки помилок для забезпечення надійності даних. Modbus RTU є найпоширенішою реалізацією, доступною для Modbus.

  Повідомлення Modbus RTU має передаватися безперервно без коливань між символами.

Повідомлення Modbus обрамляються (розділені) періодами неактивності (мовчання). Для цього типу зазвичай використовується interface RS-485.

- Modbus ASCII - Це використовується в послідовному зв’язку та використовує символи ASCII для зв’язку протоколу.

  Формат ASCII використовує контрольну суму поздовжньої надлишковості.

  Повідомлення Modbus ASCII обрамляються першою двокрапкою («:») і кінцевим символом нового рядка (CR/LF).
- Modbus TCP/IP or Modbus TCP- Це варіант Modbus, який використовується для зв’язку через мережі TCP/IP, підключаючись через порт 502. Він не потребує обчислення контрольної суми, оскільки нижчі рівні вже забезпечують захист контрольної суми.

Підтримувані параметри зв'язку Modbus

Бібліотека Modbus підтримує стандартні параметри зв’язку відповідно до специфікації Modbus, зазначеної нижче.

Стандартні варіанти зв'язку Modbus

Опція Modbus	Опис опції
Зв'язок RTU	• 1 стартовий біт • 8 біт даних, молодший біт надсилається першим • 1 біт для парності/непарності - немає біта для відсутності парності • 1 стоповий біт, якщо парність використовується, 2 стопові біти, якщо парність відсутня • Перевірка циклічної надлишковості (CRC)
Зв'язок ASCII	• 1 стартовий біт • 7-8 біт даних, молодший біт надсилається першим • 1 біт для парності/непарності - немає біта для відсутності парності

Стандартні варіанти зв'язку Modbus

Опція Modbus	Опис опції
	• 1 стоповий біт, якщо парність використовується, 2 стопові біти, якщо парність відсутня • Поздовжня перевірка надмірності (LRC)
Зв'язок TCP	• Зв'язок між клієнтом (master) - сервером (slave) через мережі TCP/IP • Для підключення використовується стандартний порт 502 • Кадри не вимагають обчислення контрольної суми (надається нижчими рівнями)

Деякі постачальники можуть використовувати підмножину параметрів зв’язку. У цьому випадку детальна інформація уточнюється в інструкції до пристрою, і можна перевизначити стандартні параметри зв’язку для підтримки таких пристроїв.

Модель обміну повідомленнями та відображення даних

Modbus - це прикладний протокол, який визначає правила для структури обміну повідомленнями та організації даних, які не залежать від середовища передачі даних.

Традиційний послідовний Modbus - це протокол на основі реєстру, який визначає транзакції повідомлень, що відбуваються між провідним(-ими) і підлеглими пристроями (при використанні Modbus TCP/IP допускається використання кількох провідних).

Підлеглі пристрої слухають повідомлення від головного і просто відповідають за вказівками.

Ведучий(і) завжди контролює зв’язок і може спілкуватися безпосередньо з одним підлеглим або всіма підключеними підлеглими, але підлеглі не можуть спілкуватися безпосередньо один з одним.

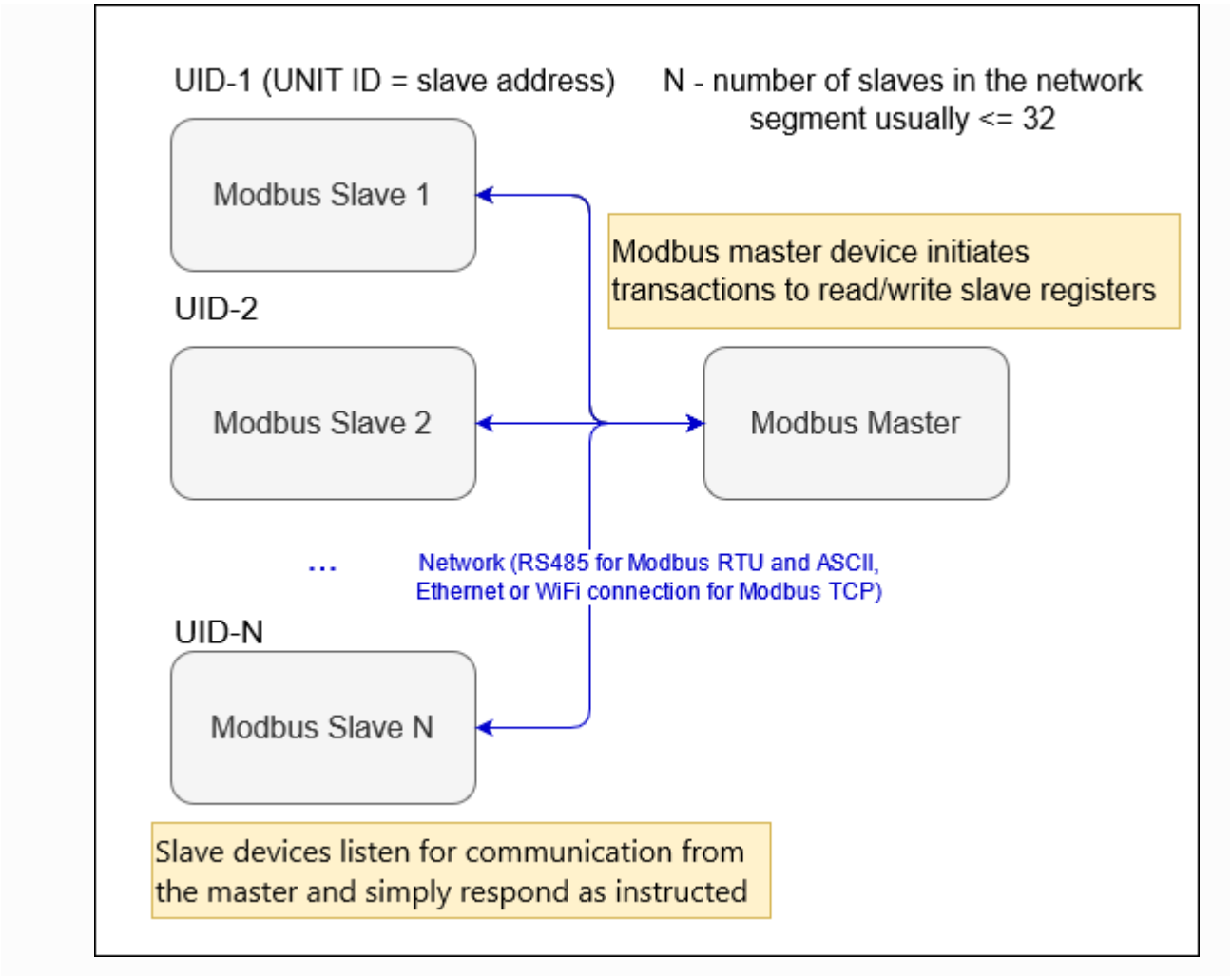


Рисунок 2 - Діаграма сегментів Modbus

Передбачається, що кількість підлеглих пристроїв і їхні карти регістрів відомі головному Modbus до початку стека.

Карта реєстру кожного підлеглого пристрою зазвичай є частиною посібника з пристрою. Підлеглий пристрій зазвичай дозволяє конфігурувати свою коротку підлеглу адресу та параметри зв'язку, які використовуються в сегменті мережі пристрою.

Протокол Modbus дозволяє пристроям відображати дані в чотирьох типах регістрів (Holding, Input, Discrete, Coil). На малюнку нижче показано приклад зіставлення даних пристрою з чотирма типами регістрів.

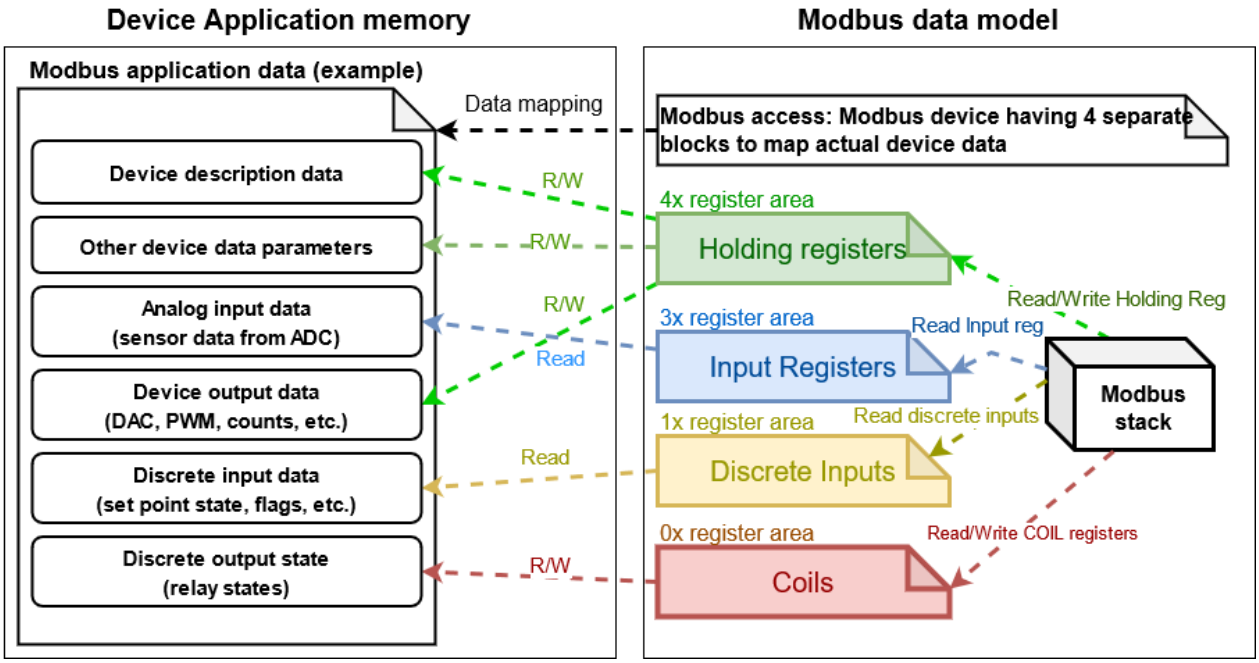


Рисунок 3 - Відображення даних Modbus

Відображення складних типів даних

Відповідно до розділу 4.2 специфікації Modbus, «MODBUS використовує big-Endian представлення для адрес і елементів даних.

Це означає, що коли передається числова величина, більша за один байт, першим надсилається старший байт».

Найбільшою офіційною структурою, визначеною специфікацією Modbus, є 16-бітний регістр слів, який складається з 2 байтів.

Однак постачальники іноді групують два або навіть чотири 16-розрядні регістри разом, щоб інтерпретувати їх як 32-розрядні або 64-розрядні значення відповідно.

Це також можливо, коли постачальники Modbus групують разом багато регістрів для серійних номерів, текстових рядків, часу/дати тощо.

Незалежно від того, як постачальник має намір інтерпретувати дані, сам протокол Modbus просто передає 16-бітні регістри слів.

Ці значення, згруповані з регістрів, можуть використовувати порядок регістрів як від маленького, так і від старшого.

Кожен окремий 16-бітний регістр кодується в порядку старшого байту (за умови, що пристрій Modbus відповідає специфікації Modbus). Однак 32-розрядні та 64-розрядні угоди про найменування типів, такі як ABCD або ABCDEFGH, не враховують порядок байтів кадру в мережевому форматі.

Наприклад: префікс ABCD для 32-розрядних значень означає загальний формат відображення Modbus і відповідає мережевому формату CDAB (порядок у кадрі).

Загальні типи даних, які підтримуються постачальниками Modbus

Тип	**Діапазон**	**Опис формату**
U8, I8 - беззнаковий/знаковий 8-розрядний тип	(0 .. 255)/(-128 .. 127)	Загальний беззнаковий 8-бітовий тип, який зазвичай зберігається в одному регістрі Modbus. Значення може бути збережено в байті HI або LO регістра або упаковано з наступним байтом в один 16-розрядний регістр.
U16 - Беззнакове ціле число 16-розрядного типу	0 - 65535	Зберігається в одному 16-розрядному регістрі. Значення можна зберігати з порядком байтів AB або BA.
I16 - Ціле число зі знаком 16-розрядного типу	Допускається від -32768 до 32767.	Зберігається в одному 16-розрядному регістрі. Значення можна зберігати за допомогою AB або BA forendiannessmat.

Тип	Діапазон	Опис формату
I32 – 32-розрядний тип довгого цілого числа зі знаком	Допускається від -2147483648 до 2147483647.	Зберігається в двох послідовних 16-розрядних регістрах. Значення можна зберігати з порядком байтів ABCD - DCBA (див. нижче).
U32 - Довге ціле число без знаку 32-розрядний тип	Допускається від 0 до 4294967295.	Зберігається в двох послідовних 16-розрядних регістрах. Значення можна зберігати з порядком байтів ABCD - DCBA.
U64 Беззнакові довгі цілі числа (беззнакове ціле число 64)	Допускається від 0 до 18446744073709551615.	Зберігається в чотирьох послідовних 16-розрядних регістрах. Значення можна зберігати з порядком байтів ABCDEFGH - BADCFEHG.
I64 Довгі цілі числа зі знаком (ціле число зі знаком 64)	-9223372036854775808 до 9223372036854775807 дозволено.	Зберігається в чотирьох послідовних 16-розрядних регістрах. Значення можна зберігати з порядком байтів ABCDEFGH - BADCFEHG.
Число з плаваючою комою одинарної точності 32 біта	Допускається від 1,17549435E-38 до 3,40282347E+38.	Зберігається в двох послідовних 16-бітних регістрах відповідно до IEEE754. Значення можна зберігати з порядком байтів ABCD - DCBA.
Число з плаваючою точкою подвійної точності 64 біт	Допускається від +/-5,0E-324 до +/-1,7E+308.	Зберігається в чотирьох послідовних 16-бітних регістрах відповідно до

Тип	Діапазон	Опис формату
		IEEE754. Значення можна зберігати з порядком байтів ABCDEFGH - BADCFEHG.

Як показано в таблиці вище, типи float і double не підходять для 16-бітного регістра і вимагають використання кількох послідовних регістрів для зберігання значення.

Однак різні виробники зберігають послідовні байти в різному порядку (не стандартизовано).

Наприклад: префікс DCBA означає інверсний формат Modbus (порядок BADC у мережевому форматі).

Порядок байтів Modbus для розширених типів

Постфікс	Опис формату
А Б В Г	Великий порядок байтів, старший байт першим
CDAB	Великий порядок байтів, зворотний порядок реєстрів (Маленький порядок байтів із заміною байтів)
BADC	Little endian, зворотний порядок реєстрів (Big endian із заміною байтів)
DCBA	Little endian (молодший байт спочатку)

Розширені типи даних використовуються для визначення всіх можливих комбінацій згрупованих значень, представлених нижче та відповідають param_type полям словника даних, як описано нижче:

Тип	Опис типу формату (загальний формат)	Тип формату (мережевий формат)
PARAM_TYPE_U8	тип сумісності відповідає **PARAM_TYPE_U8_A**	Беззнакове ціле число 8 біт
PARAM_TYPE_U16	Беззнакове ціле число 16 біт, відповідає **PARAM_TYPE_U16_AB**	Заміна байтів у порядку байтів
PARAM_TYPE_U32	Ціле число без знаку за замовчуванням 32-розрядний тип, відповідає **PARAM_TYPE_U32_ABCD**	Заміна байтів у порядку байтів
PARAM_TYPE_FLOAT	Ціле число без знаку за замовчуванням 32-розрядний тип, відповідає **PARAM_TYPE_FLOAT_ABCD**	Заміна байтів у порядку байтів
PARAM_TYPE_ASCII	Типовий формат рядка ASCII	Упаковані рядкові дані ASCII
PARAM_TYPE_BIN	Двійковий тип даних	Тип за замовчуванням для бінарних упакованих даних
PARAM_TYPE_I8_A	I8 ціле число зі знаком у молодшому байті регістра,	I8 ціле число зі знаком LO

Тип	Опис типу формату (загальний формат)	Тип формату (мережевий формат)
	старший байт дорівнює нулю	
PARAM_TYPE_I8_B	I8 ціле число зі знаком у старшому байті регістра, молодший байт дорівнює нулю	I8 ціле число зі знаком HI
PARAM_TYPE_U8_A	U8 беззнакове ціле число, записане в молодший байт регістра, старший байт дорівнює нулю	U8 беззнакове ціле число LO
PARAM_TYPE_U8_B	U8 беззнакове ціле число, записане в старший байт регістра, молодший байт дорівнює нулю	U8 беззнакове ціле число HI
PARAM_TYPE_I16_AB	I16 ціле число зі знаком, порядок байтів	Великий порядок байтів
PARAM_TYPE_I16_BA	I16 ціле число зі знаком, порядок байтів	Little endian
PARAM_TYPE_U16_AB	U16 беззнакове ціле число, великий порядок байтів	Великий порядок байтів
PARAM_TYPE_U16_BA	U16 ціле число без знаку, порядок байтів	Little endian
PARAM_TYPE_I32_ABCD	I32 ABCD ціле число зі знаком, великий порядок байтів	Заміна байтів у порядку байтів

Тип	Опис типу формату (загальний формат)	Тип формату (мережевий формат)
PARAM_TYPE_I32_CDAB	I32 Ціле число зі знаком CDAB, старший порядок байтів, зворотний порядок регістрів	Великий порядок байтів
PARAM_TYPE_I32_BADC	I32 Ціле число зі знаком BADC, порядок порядків байтів у зворотному порядку	Little endian
PARAM_TYPE_I32_DCBA	I32 DCBA ціле число зі знаком, порядок байтів	Перестановка байтів у порядку байтів
PARAM_TYPE_U32_ABCD	U32 ABCD беззнакове ціле число, великий порядок байтів	Заміна байтів у порядку байтів
PARAM_TYPE_U32_CDAB	U32 CDAB беззнакове ціле число, старший порядок байтів, зворотний порядок регістрів	Великий порядок байтів
PARAM_TYPE_U32_BADC	U32 BADC беззнакове ціле число, порядковий порядок байтів, зворотний порядок регістрів	Little endian
PARAM_TYPE_U32_DCBA	U32 DCBA ціле число без знаку, порядок байтів	Перестановка байтів у порядку байтів

Тип	Опис типу формату (загальний формат)	Тип формату (мережевий формат)
PARAM_TYPE_FLOAT_ABCD	Float ABCD з плаваючою комою, старший порядок байтів	Заміна байтів у порядку байтів
PARAM_TYPE_FLOAT_CDAB	Float CDAB з плаваючою точкою, старший порядок байтів, зворотний порядок регістрів	Великий порядок байтів
PARAM_TYPE_FLOAT_BADC	Число з плаваючою точкою BADC, порядок байтів у порядку байтів, зворотний порядок регістрів	Little endian
PARAM_TYPE_FLOAT_DCBA	Float DCBA з плаваючою точкою, маленький порядок байтів	Перестановка байтів у порядку байтів
PARAM_TYPE_I64_ABCDEFGH	I64, ABCDEFGH ціле число зі знаком, старший порядок байтів	Заміна байтів у порядку байтів
PARAM_TYPE_I64_HGFEDCBA	I64, HGFEDCBA ціле число зі знаком, порядок байтів	Перестановка байтів у порядку байтів
PARAM_TYPE_I64_GHEFCDAB	I64, ціле число зі знаком GHEFCDAB, старший порядок байтів, зворотний порядок регістрів	Великий порядок байтів
PARAM_TYPE_I64_BADCFEHG	I64, ціле число зі знаком BADCFEHG, порядок байтів	Little endian

Тип	Опис типу формату (загальний формат)	Тип формату (мережевий формат)
	від маленького байтів, зворотний порядок регістрів	
PARAM_TYPE_U64_ABCDEFGH	U64, ABCDEFGH беззнакове ціле число, великий порядок байтів	Заміна байтів у порядку байтів
PARAM_TYPE_U64_HGFEDCBA	U64, HGFEDCBA Ціле число без знака, порядок байтів	Перестановка байтів у порядку байтів
PARAM_TYPE_U64_GHEFCDAB	U64, GHEFCDAB беззнакове ціле число, старший порядок байтів, зворотний порядок регістрів	Великий порядок байтів
PARAM_TYPE_U64_BADCFEHG	U64, BADCFEHG беззнакове ціле число, порядковий порядок байтів, зворотний порядок регістрів	Little endian
PARAM_TYPE_DOUBLE_ABCDEFGH	Double ABCDEFGH з плаваючою комою, старший порядок байтів	Заміна байтів у порядку байтів
PARAM_TYPE_DOUBLE_HGFEDCBA	Double HGFEDCBA з плаваючою комою, маленький порядок байтів	Перестановка байтів у порядку байтів
PARAM_TYPE_DOUBLE_GHEFCDAB	Double GHEFCDAB з плаваючою комою, старший порядок байтів, зворотний порядок регістрів	Великий порядок байтів

Тип	Опис типу формату (загальний формат)	Тип формату (мережевий формат)
PARAM_TYPE_DOUBLE_BADCFEHG	Double BADCFEHG з плаваючою комою, маленький порядок байтів, зворотний порядок регістрів	Little endian

Підтримку розширених типів даних слід увімкнути за допомогою пункту CONFIG_FMB_MASTER_TIMEOUT_MS_RESPONDменю kconfig.

На діаграмах нижче показано, як розширені типи даних відображаються на мережевому рівні.

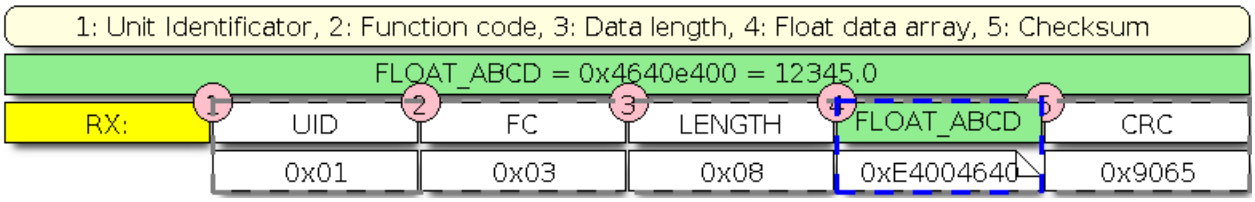


Рисунок 4 - Головна відповідь Modbus з кадром ABCD

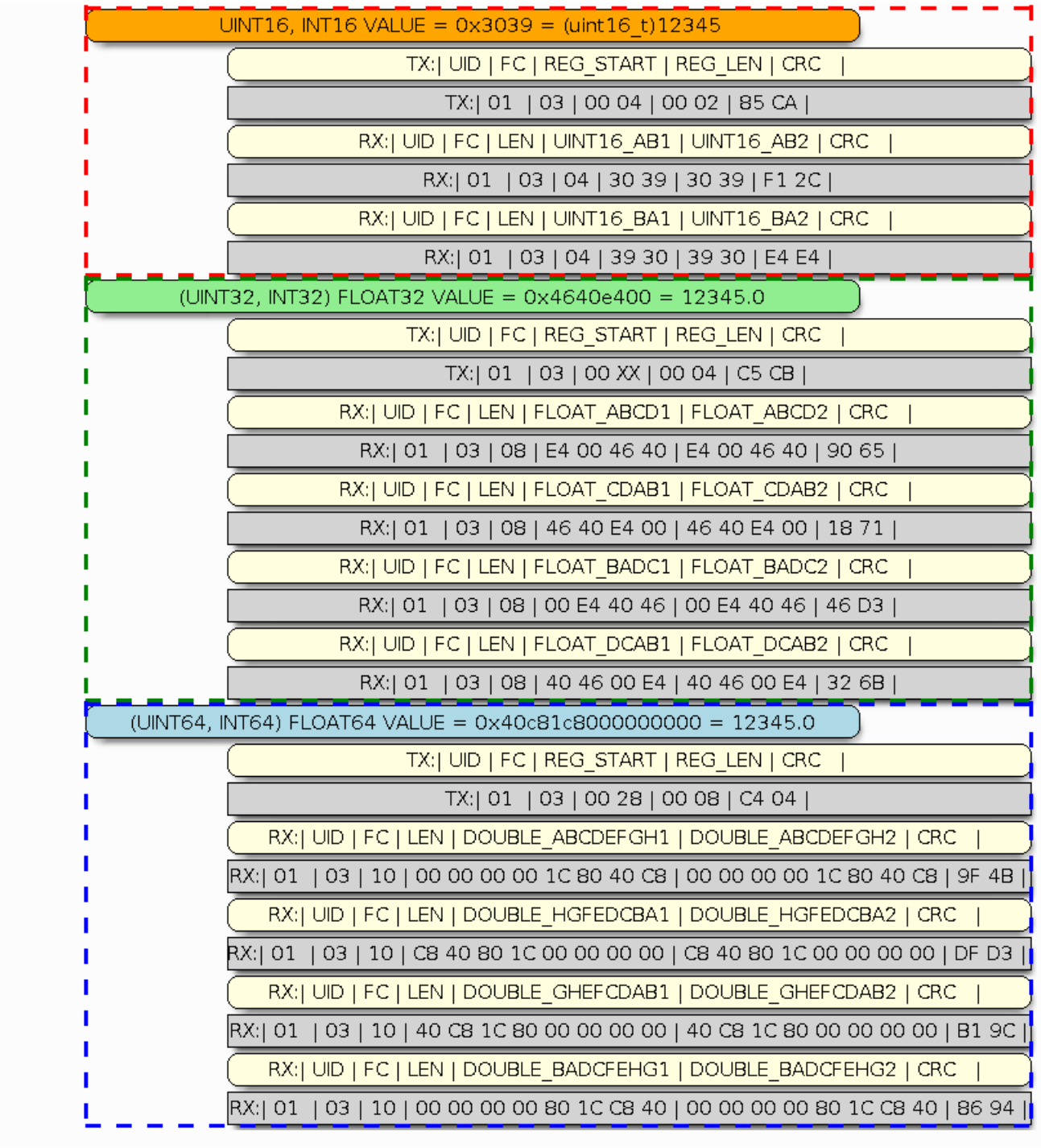


Рисунок 5 - Приклади упаковки кадрів Modbus (16-бітні, 32-бітні, 64-бітні дані)

Показаний вище підхід можна використовувати для упаковки даних у кадри MBAP, які використовуються Modbus TCP, а також для інших типів подібного розміру.

Ініціалізація порту Modbus

ESP_Modbus підтримує порти Modbus SERIAL і TCP, і порт потрібно ініціалізувати перед викликом будь-якого іншого Modbus API.

Наведені нижче функції використовуються для створення та ініціалізації interface контролера Modbus (головного або підлеглого) через певне середовище передачі (послідовне або TCP/IP):

- **mbc_slave_init()**
- **mbc_master_init()**
- **mbc_slave_init_tcp()**
- **mbc_master_init_tcp()**

Виклик API використовує перший параметр для розпізнавання типу ініціалізованого порту.

Підтримуване перерахування для різних портів: MB_PORT_SERIAL_MASTER, MB_PORT_SERIAL_SLAVE відповідно. Параметри MB_PORT_TCP_MASTER зарезервовано MB_PORT_TCP_SLAVE для внутрішнього використання.

```
void* master_handler = NULL; // Pointer to allocate interface structure
// Initialization of Modbus master for serial port
esp_err_t err = mbc_master_init(MB_PORT_SERIAL_MASTER, &master_handler);
if (master_handler == NULL || err != ESP_OK) {
    ESP_LOGE(TAG, "mb controller initialization fail.");
}
```

Цей приклад коду для ініціалізації підлеглого порту:

```
void* slave_handler = NULL; // Pointer to allocate interface structure
// Initialization of Modbus slave for TCP
esp_err_t err = mbc_slave_init_tcp(&slave_handler);
if (slave_handler == NULL || err != ESP_OK) {
    // Error handling is performed here
    ESP_LOGE(TAG, "mb controller initialization fail.");
}
```

Огляд Modbus Master API

Налаштування доступу до основних даних

Архітектурний підхід ESP_Modbus включає один рівень вище стандартного драйвера Modbus IO.

Додатковий рівень називається контролером Modbus і його метою є додавання абстракції, такої як CID - ідентифікатор характеристики. CID пов’язаний із відповідними регістрами

Modbus через таблицю під назвою «Словник даних» і представляє фізичні параметри пристрою (такі як температура, вологість тощо) у певному веденому пристрої Modbus.

Цей підхід дозволяє ізолювати верхній рівень (наприклад, MESH або MQTT) від особливостей

Modbus, таким чином спрощуючи інтеграцію Modbus з іншими протоколами/мережами.

Словник даних - це список у головному пристрої Modbus, який має бути визначений користувачем для зв’язування кожного CID із відповідним представленням регістрів Modbus за допомогою таблиці відображення реєстрів підлеглого пристрою Modbus, який використовується.

Кожен елемент у цьому словнику даних має тип mb_parameter_descriptor_t і представляє опис однієї фізичної характеристики:

Поле	**опис**	**Детальна інформація**
cid	Ідентифікатор характеристики	Ідентифікатор характеристики (має бути унікальним).
param_key	Характерне ім'я	Рядковий опис характеристики.
param_units	Одиниці характеристики	Фізичні одиниці характеристики.

Поле	опис	Детальна інформація
mb_slave_addr	Підлегла адреса Modbus	Коротка адреса пристрою з відповідним параметром UID.
mb_param_type	Тип реєстру Modbus	Тип області реєстрації Modbus. **MB_PARAM_INPUT** , **MB_PARAM_HOLDING** , **MB_PARAM_COIL** , **MB_PARAM_DISCRETE** - представляє область вхідних регістрів Input , Holding, Coil і Discrete відповідно;
mb_reg_start	Початок реєстрації Modbus	Відносна адреса реєстру характеристики в області реєстру.
mb_size	Розмір регістра Modbus	Довжина характеристики в регістрах (два байти).
param_offset	Зсув екземпляра	Зсув до екземпляра характеристики в байтах. Він використовується для обчислення абсолютної адреси характеристики в структурі зберігання. Це необов’язкове поле, і його можна встановити на нуль, якщо параметр не використовується в програмі.
param_type	Тип даних	Визначає тип характеристики. Можливі типи описані в розділі Відображення складних типів даних .
param_size	Розмір даних	Розмір зберігання характеристики (у байтах) описує розмір даних, які потрібно зберігати в екземплярі даних

Поле	опис	Детальна інформація
		під час відображення. Для зіставлення складних типів даних це дозволяє визначити контейнер даних відповідного типу.
param_opts	Параметри	Межі, параметри характеристики, що використовуються під час обробки сигналу тривоги в програмі користувача (необов'язково)
access	Тип доступу до параметрів	Може використовуватися в програмі користувача для визначення поведінки характеристики під час обробки даних у програмі користувача; **PAR_PERMS_READ_WRITE_TRIGGER**, **PAR_PERMS_READ**, **PAR_PERMS_READ_WRITE_TRIGGER**;

Приклади картографування

Налаштування доступу до застарілих типів параметрів.

CID	зареєструватися	Довжина	Діапазон	Тип	одиниці	опис
0	30000	4	MAX_UINT	U32	Не визначений	Серійний номер пристрою (4 байти) лише для читання
1	30002	2	MAX_UINT	U16	Не визначений	Версія програмного забезпечення (4 байти) лише для читання
2	40000	4	-20..40	ПЛАВАТИ	DegC	Кімнатна температура в градусах С. Запис значення температури в цей регістр для калібрування по одній точці.
3	40002	16	1..100 байт	ASCII або двійковий масив	Не визначений	Назва пристрою (16 байт) Рядок ASCII. Тип *PARAM_TYPE_ASCII* дозволяє читати/записувати складний параметр (рядок або двійкові дані), який відповідає одному CID.

```
// Enumeration of modbus slave addresses accessed by master device
enum {
    MB_DEVICE_ADDR1 = 1,
    MB_DEVICE_ADDR2,
    MB_SLAVE_COUNT
};

// Enumeration of all supported CIDs for device
enum {
    CID_SER_NUM1 = 0,
    CID_SW_VER1,
    CID_DEV_NAME1,
    CID_TEMP_DATA_1,
    CID_SER_NUM2,
    CID_SW_VER2,
    CID_DEV_NAME2,
    CID_TEMP_DATA_2
};

// Example Data Dictionary for Modbus parameters in 2 slaves in the
segment
mb_parameter_descriptor_t device_parameters[] = {
    // CID, Name, Units, Modbus addr, register type, Modbus Reg Start
Addr, Modbus Reg read length,
    // Instance offset (NA), Instance type, Instance length (bytes),
Options (NA), Permissions
    { CID_SER_NUM1, STR("Serial_number_1"), STR("--"), MB_DEVICE_ADDR1,
MB_PARAM_INPUT, 0, 2,
                    0, PARAM_TYPE_U32, 4, OPTS( 0,0,0 ),
PAR_PERMS_READ_WRITE_TRIGGER },
    { CID_SW_VER1, STR("Software_version_1"), STR("--"), MB_DEVICE_ADDR1,
MB_PARAM_INPUT, 2, 1,
                    0, PARAM_TYPE_U16, 2, OPTS( 0,0,0 ),
PAR_PERMS_READ_WRITE_TRIGGER },
    { CID_DEV_NAME1, STR("Device name"), STR("__"), MB_DEVICE_ADDR1,
MB_PARAM_HOLDING, 2, 8,
                    0, PARAM_TYPE_ASCII, 16, OPTS( 0, 0, 0 ),
PAR_PERMS_READ_WRITE_TRIGGER },
    { CID_TEMP_DATA_1, STR("Temperature_1"), STR("C"), MB_DEVICE_ADDR1,
MB_PARAM_HOLDING, 0, 2,
                    0, PARAM_TYPE_FLOAT, 4, OPTS( 16, 30, 1 ),
PAR_PERMS_READ_WRITE_TRIGGER },
    { CID_SER_NUM2, STR("Serial_number_2"), STR("--"), MB_DEVICE_ADDR2,
MB_PARAM_INPUT, 0, 2,
                    0, PARAM_TYPE_U32, 4, OPTS( 0,0,0 ),
PAR_PERMS_READ_WRITE_TRIGGER },
    { CID_SW_VER2, STR("Software_version_2"), STR("--"), MB_DEVICE_ADDR2,
MB_PARAM_INPUT, 2, 1,
                    0, PARAM_TYPE_U16, 2, OPTS( 0,0,0 ),
PAR_PERMS_READ_WRITE_TRIGGER },
```

```
    { CID_DEV_NAME2, STR("Device name"), STR("__"), MB_DEVICE_ADDR1,
MB_PARAM_HOLDING, 2, 8,
                    0, PARAM_TYPE_ASCII, 16, OPTS( 0, 0, 0 ),
PAR_PERMS_READ_WRITE_TRIGGER },
    { CID_TEMP_DATA_2, STR("Temperature_2"), STR("C"), MB_DEVICE_ADDR2,
MB_PARAM_HOLDING, 0, 2,
                    0, PARAM_TYPE_FLOAT, 4, OPTS( 20, 30, 1 ),
PAR_PERMS_READ_WRITE_TRIGGER },
};
// Calculate number of parameters in the table
uint16_t num_device_parameters = (sizeof(device_parameters) /
sizeof(device_parameters[0]));
```

Приклад 2: Налаштування доступу за допомогою розширених типів параметрів для пристроїв сторонніх виробників.

CID	зареєструватися	Довжина	Діапазон	одиниці	опис
0	40000	4	0 … 255	Немає одиниць	**PARAM_TYPE_U8_A** - беззнакове ціле число 8 біт
1	40002	4	0 … 65535	Немає одиниць	**PARAM_TYPE_U16_AB** unsigned integer 16-bit
3	40004	8	0 … Беззнакове ціле число 32-розрядний діапазон	Немає одиниць	**PARAM_TYPE_U32_ABCD** - беззнакове ціле число 32 біт у форматі ABCD
4	40008	8	0 … Беззнакове ціле число 32-розрядний діапазон	Немає одиниць	**PARAM_TYPE_FLOAT_CDAB** - 32-розрядне значення FLOAT у форматі CDAB

CID	зареєструватися	Довжина	Діапазон	одиниці	опис
5	400012	16	0 … Беззнакове ціле число 64-розрядний діапазон	Немає одиниць	**PARAM_TYPE_U64_ABCDEFGH** - 64-розрядне ціле число без знаку у форматі ABCDEFGH
6	400020	16	0 … Беззнакове ціле число 64-розрядний діапазон	Немає одиниць	**PARAM_TYPE_DOUBLE_HGFEDCBA** - 64-розрядне значення подвійної точності у форматі HGFEDCBA

```
  #include "limits.h"
  #include "mbcontroller.h"

  #define HOLD_OFFSET(field) ((uint16_t)(offsetof(holding_reg_params_t,
field) + 1))
  #define HOLD_REG_START(field) (HOLD_OFFSET(field) >> 1)
  #define HOLD_REG_SIZE(field) (sizeof(((holding_reg_params_t *)0)->field)
>> 1)

  #pragma pack(push, 1)
  // Example structure that contains parameter arrays of different types
  // with different options of endianness.
  typedef struct
  {
      uint16_t holding_u8_a[2];
      uint16_t holding_u16_ab[2];
      uint32_t holding_uint32_abcd[2];
      float holding_float_cdab[2];
      double holding_uint64_abcdefgh[2];
      double holding_double_hgfedcba[2];
  } holding_reg_params_t;
  #pragma pack(pop)

  // Enumeration of modbus slave addresses accessed by master device
  enum {
      MB_DEVICE_ADDR1 = 1, // Short address of Modbus slave device
      MB_SLAVE_COUNT
```

```
  };

// Enumeration of all supported CIDs for device (used in parameter
definition table)
  enum {
      CID_HOLD_U8_A = 0,
      CID_HOLD_U16_AB,
      CID_HOLD_UINT32_ABCD,
      CID_HOLD_FLOAT_CDAB,
      CID_HOLD_UINT64_ABCDEFGH,
      CID_HOLD_DOUBLE_HGFEDCBA,
      CID_COUNT
  };

  // Example Data Dictionary for to address parameters from slaves with
different options of endianness
  mb_parameter_descriptor_t device_parameters[] = {
      // CID, Name, Units, Modbus addr, register type, Modbus Reg Start
Addr, Modbus Reg read length,
      // Instance offset (NA), Instance type, Instance length (bytes),
Options (NA), Permissions
      { CID_HOLD_U8_A, STR("U8_A"), STR("--"), MB_DEVICE_ADDR1,
MB_PARAM_HOLDING,
              HOLD_REG_START(holding_u8_a), HOLD_REG_SIZE(holding_u8_a),
              HOLD_OFFSET(holding_u8_a), PARAM_TYPE_U8_A,
(HOLD_REG_SIZE(holding_u8_a) << 1),
              OPTS( 0, UCHAR_MAX, 0 ), PAR_PERMS_READ_WRITE_TRIGGER },
      { CID_HOLD_U16_AB, STR("U16_AB"), STR("--"), MB_DEVICE_ADDR1,
MB_PARAM_HOLDING,
              HOLD_REG_START(holding_u16_ab),
HOLD_REG_SIZE(holding_u16_ab),
              HOLD_OFFSET(holding_u16_ab), PARAM_TYPE_U16_AB,
(HOLD_REG_SIZE(holding_u16_ab) << 1),
              OPTS( 0, USHRT_MAX, 0 ), PAR_PERMS_READ_WRITE_TRIGGER },
      { CID_HOLD_UINT32_ABCD, STR("UINT32_ABCD"), STR("--"),
MB_DEVICE_ADDR1, MB_PARAM_HOLDING,
              HOLD_REG_START(holding_uint32_abcd),
HOLD_REG_SIZE(holding_uint32_abcd),
              HOLD_OFFSET(holding_uint32_abcd), PARAM_TYPE_U32_ABCD,
(HOLD_REG_SIZE(holding_uint32_abcd) << 1),
              OPTS( 0, ULONG_MAX, 0 ), PAR_PERMS_READ_WRITE_TRIGGER },
      { CID_HOLD_FLOAT_CDAB, STR("FLOAT_CDAB"), STR("--"),
MB_DEVICE_ADDR1, MB_PARAM_HOLDING,
              HOLD_REG_START(holding_float_cdab),
HOLD_REG_SIZE(holding_float_cdab),
              HOLD_OFFSET(holding_float_cdab), PARAM_TYPE_FLOAT_CDAB,
(HOLD_REG_SIZE(holding_float_cdab) << 1),
              OPTS( 0, ULONG_MAX, 0 ), PAR_PERMS_READ_WRITE_TRIGGER },
      { CID_HOLD_UINT64_ABCDEFGH, STR("UINT64_ABCDEFGH"), STR("--"),
MB_DEVICE_ADDR1, MB_PARAM_HOLDING,
```

```
                HOLD_REG_START(holding_uint64_abcdefgh),
HOLD_REG_SIZE(holding_uint64_abcdefgh),
                HOLD_OFFSET(holding_uint64_abcdefgh),
PARAM_TYPE_UINT64_ABCDEFGH, (HOLD_REG_SIZE(holding_uint64_abcdefgh) << 1),
                OPTS( 0, ULLONG_MAX, 0 ), PAR_PERMS_READ_WRITE_TRIGGER },
      { CID_HOLD_DOUBLE_HGFEDCBA, STR("DOUBLE_HGFEDCBA"), STR("--"),
MB_DEVICE_ADDR1, MB_PARAM_HOLDING,
                HOLD_REG_START(holding_double_hgfedcba),
HOLD_REG_SIZE(holding_double_hgfedcba),
                HOLD_OFFSET(holding_double_hgfedcba),
PARAM_TYPE_DOUBLE_HGFEDCBA, (HOLD_REG_SIZE(holding_double_hgfedcba) << 1),
                OPTS( 0, ULLONG_MAX, 0 ), PAR_PERMS_READ_WRITE_TRIGGER }
  };
  uint16_t num_device_parameters = (sizeof(device_parameters) /
sizeof(device_parameters[0]));
```

Стек Modbus також містить Modbus Endianness Conversion API Reference - функції API перетворення endianness, які дозволяють перетворювати значення з/до кожного розширеного типу в представлення компілятора.

Під час ініціалізації стека Modbus вказівник на словник даних (так званий дескриптор) повинен бути наданий як параметр функції нижче.

mbc_master_set_descriptor(): Ініціалізація головного дескриптора.

Дескриптор представляє масив типу mb_parameter_descriptor_t та описує всі характеристики, до яких звертається майстер.

ESP_ERROR_CHECK(mbc_master_set_descriptor(&device_parameters[0], num_device_parameters));

Словник даних можна ініціалізувати з SD-карти, MQTT або іншого джерела перед запуском стека.

Після завершення ініціалізації та налаштування контролер Modbus дозволяє зчитувати складні параметри з будь-якого підлеглого пристрою, включеного в таблицю дескрипторів, використовуючи його CID.

Параметри спілкування

Виклик функції налаштування дозволяє визначити конкретні параметри зв’язку для порту.

mbc_master_setup()

Структура зв’язку, надана як параметр, відрізняється для режиму послідовного та TCP зв’язку.

Приклад налаштування для послідовного порту:

```
mb_communication_info_t comm_info = {
    .port = MB_PORT_NUM,         // Serial port number
    .mode = MB_MODE_RTU,         // Modbus mode of communication
(MB_MODE_RTU or MB_MODE_ASCII)
    .baudrate = 9600,            // Modbus communication baud rate
    .parity = MB_PARITY_NONE     // parity option for serial port
};
ESP_ERROR_CHECK(mbc_master_setup((void*)&comm_info));
```

Однак можна перевизначити параметри послідовного зв’язку, викликавши функцію **uart_param_config()** одразу після **mbc_slave_setup()**.

Головний TCP-порт Modbus вимагає додаткового визначення таблиці IP-адрес, де кількість адрес має дорівнювати кількості унікальних підлеглих адрес у головному словнику даних Modbus:

Порядок рядка IP-адреси відповідає короткій підлеглій адресі в словнику даних.

```
#define MB_SLAVE_COUNT 2 // Number of slaves in the segment being accessed
(as defined in Data Dictionary)

char* slave_ip_address_table[MB_SLAVE_COUNT] = {
    "192.168.1.2",     // Address corresponds to UID1 and set to
predefined value by user
    "192.168.1.3",     // corresponds to UID2 in the segment
    NULL               // end of table
};

mb_communication_info_t comm_info = {
    .ip_port = MB_TCP_PORT,                    // Modbus TCP port number
(default = 502)
    .ip_addr_type = MB_IPV4,                   // version of IP protocol
    .ip_mode = MB_MODE_TCP,                    // Port communication mode
    .ip_addr = (void*)slave_ip_address_table,  // assign table of IP
addresses
    .ip_netif_ptr = esp_netif_ptr              // esp_netif_ptr pointer to
the corresponding network interface
};

ESP_ERROR_CHECK(mbc_master_setup((void*)&comm_info));
```

Майстер спілкування

Запуск контролера Modbus є останнім кроком у активації зв’язку. Це виконується за допомогою функції нижче:

mbc_master_start()

```
esp_err_t err = mbc_master_start();
if (err != ESP_OK) {
    ESP_LOGE(TAG, "mb controller start fail, err=%x.", err);
}
```

Наведений нижче список функцій використовується головним стеком Modbus із програми користувача:

mbc_master_send_request():

Ця функція виконує блокуючий запит Modbus. Ведучий надсилає запит на дані (як визначено в структурі запиту параметрів **mb_param_request_t**), а потім блокує до відповіді від відповідного підлеглого та повертає статус виконання команди. Ця функція забезпечує стандартний спосіб доступу для читання/запису до пристроїв Modbus у мережі.

mbc_master_get_cid_info():

функція отримує інформацію про кожну характеристику, яка підтримується в словнику даних, і повертає опис характеристики у вигляді структури **mb_parameter_descriptor_t**. Доступ до кожної характеристики здійснюється за допомогою її CID.

mbc_master_get_parameter():

функція читає дані характеристики, визначені в параметрах веденого пристрою Modbus. Додаткові дані для запиту беруться з таблиці опису параметрів.

приклад:

```
const mb_parameter_descriptor_t* param_descriptor = NULL;
uint8_t temp_data[4] = {0}; // temporary buffer to hold maximum CID size
uint8_t type = 0;
....

// Get the information for characteristic cid from data dictionary
esp_err_t err = mbc_master_get_cid_info(cid, &param_descriptor);
if ((err != ESP_ERR_NOT_FOUND) && (param_descriptor != NULL)) {
    err          =          mbc_master_get_parameter(param_descriptor->cid,
(char*)param_descriptor->param_key, (uint8_t*)temp_data, &type);
    if (err == ESP_OK) {
        ESP_LOGI(TAG, "Characteristic #%d %s (%s) value = (0x%" PRIx32 ")
read successful.",
                         param_descriptor->cid,
                         (char*)param_descriptor->param_key,
                         (char*)param_descriptor->param_units,
                         *(uint32_t*)temp_data);
    } else {
        ESP_LOGE(TAG, "Characteristic #%d (%s) read fail, err = 0x%x (%s).",
                        param_descriptor->cid,
                        (char*)param_descriptor->param_key,
                        (int)err,
                        (char*)esp_err_to_name(err));
    }
} else {
    ESP_LOGE(TAG, "Could not get information for characteristic %d.", cid);
}
```

mbc_master_set_parameter()

Функція записує значення характеристики, визначене як ім’я та параметр cid, у відповідний підлеглий пристрій. Додаткові дані для запиту параметрів беруться з головної таблиці опису параметрів.

```
uint8_t type = 0; // Type of parameter
uint8_t temp_data[4] = {0}; // temporary buffer

esp_err_t err = mbc_master_set_parameter(CID_TEMP_DATA_2, "Temperature_2",
(uint8_t*)temp_data, &type);
if (err == ESP_OK) {
    ESP_LOGI(TAG, "Set parameter data successfully.");
} else {
    ESP_LOGE(TAG,   "Set   data   fail,   err   =   0x%x   (%s).",   (int)err,
(char*)esp_err_to_name(err));
}
```

Modbus Master Teardown()

Ця функція зупиняє комунікаційний стек Modbus і руйнує interface контролера та звільняє всі використані активні об’єкти.

mbc_master_destroy()

```
ESP_ERROR_CHECK(mbc_master_destroy());
```

MODBUS SLAVE API

Типовий робочий процес програмування для підлеглого API, який слід викликати в такому порядку:

1. Ініціалізація порту Modbus - ініціалізація interface контролера Modbus для вибраного порту.
2. Налаштування доступу до підлеглих даних – налаштуйте дескриптори даних для доступу до підлеглих параметрів.
3. Параметри підлеглого зв’язку – дозволяє налаштувати параметри зв’язку для вибраного порту.
4. Підлеглий зв'язок - початок стеку та надсилання/отримання даних. Фільтрувати події, коли майстер отримує доступ до областей реєстру.
5. Modbus Slave Teardown - Знищення контролера Modbus та його ресурсів.

Налаштування підлеглого доступу до даних

Наступні функції необхідно викликати, коли підлеглий порт контролера Modbus уже ініціалізовано.

Підлеглий стек вимагає від користувача визначення структур (областей зберігання пам’яті), які зберігають параметри Modbus, до яких стек звертається.

Ці структури повинні бути підготовлені користувачем і призначені interface контролера Modbus за допомогою **mbc_slave_set_descriptor()** виклику API перед початком зв’язку.

Підлегле завдання може викликати **mbc_slave_check_event()** функцію, яка блокуватиметься, доки головний Modbus не отримає доступу до підлеглого. Потім підлегле завдання може отримати інформацію про дані, до яких здійснюється доступ.

Один підлеглий пристрій може визначити кілька дескрипторів області для кожного типу області реєстру Modbus з різним початковим зсувом.

Область реєстру визначається за допомогою структури **mb_register_area_descriptor_t**.

Поле	**опис**
start_offset	Відносне зміщення регістра на основі нуля для визначеної області регістра. Приклад: адреса реєстру = 40002 (4x область реєстру - функція 3 - реєстр зберігання), start_offset = 2
type	Тип області реєстру Modbus. **mb_param_type_t** Для отримання додаткової інформації зверніться до .
address	Покажчик на область пам'яті, яка використовується для зберігання даних реєстру для цього дескриптора області.
size	Розмір області пам'яті в байтах, яка використовується для зберігання даних регістру.

mbc_slave_set_descriptor()

Функція ініціалізує дескриптори зв’язку Modbus для кожного типу області регістра Modbus (регістри зберігання, регістри введення, котушки (однобітовий вихід), дискретні входи).

Після ініціалізації областей і **mbc_slave_start()** виклику API стек Modbus може отримати доступ до даних у структурах даних користувача за запитом від головного.

```
#define MB_REG_INPUT_START_AREA0    (0)
#define MB_REG_HOLDING_START_AREA0  (0)
#define MB_REG_HOLD_CNT             (100)
#define MB_REG_INPUT_CNT            (100)

mb_register_area_descriptor_t reg_area; // Modbus register area descriptor
structure
unit16_t holding_reg_area[MB_REG_HOLD_CNT] = {0}; // storage area for
holding registers
unit16_t input_reg_area[MB_REG_INPUT_CNT] = {0}; // storage area for input
registers

reg_area.type = MB_PARAM_HOLDING;                                  // Set
type of register area
reg_area.start_offset = MB_REG_HOLDING_START_AREA0;                // Offset
of register area in Modbus protocol
reg_area.address = (void*)&holding_reg_area[0];                    // Set
pointer to storage instance
reg_area.size = sizeof(holding_reg_area) << 1;                     // Set the
size of register storage area in bytes
ESP_ERROR_CHECK(mbc_slave_set_descriptor(reg_area));

reg_area.type = MB_PARAM_INPUT;
reg_area.start_offset = MB_REG_INPUT_START_AREA0;
reg_area.address = (void*)&input_reg_area[0];
reg_area.size = sizeof(input_reg_area) << 1;
ESP_ERROR_CHECK(mbc_slave_set_descriptor(reg_area));
```

Принаймні один дескриптор області для кожного типу реєстру Modbus має бути встановлений, щоб забезпечити доступ до реєстру до його області.

Якщо майстер намагається отримати доступ до невизначеної області, стек створить виняток Modbus.

Прямий доступ до області реєстрації з програми користувача має бути захищений критичним розділом:

```
portENTER_CRITICAL(&param_lock);
holding_reg_area[2] += 10;
portEXIT_CRITICAL(&param_lock);
```

Стек підтримує розширені типи даних, якщо його ввімкнути за допомогою пункту `CONFIG_FMB_MASTER_TIMEOUT_MS_RESPOND` меню kconfig. У цьому випадку зіставлені значення даних можна ініціалізувати до певного формату за допомогою довідника Modbus Endianness Conversion API Reference.

Приклад ініціалізації зіставлених значень:

```
#include "mbcontroller.h"       // for mbcontroller defines and api
val_32_arr holding_float_abcd[2] = {0};
val_64_arr holding_double_ghefcdab[2] = {0};
...
// set the Modbus parameter to specific format
portENTER_CRITICAL(&param_lock); // critical section is required if the
stack is active
mb_set_float_abcd(&holding_float_abcd[0], (float)12345.0);
mb_set_float_abcd(&holding_float_abcd[1], (float)12345.0);
mb_set_double_ghefcdab(&holding_double_ghefcdab[0], (double)12345.0);
portEXIT_CRITICAL(&param_lock);
...
// The actual abcd formatted value can be converted to actual float
represenatation as below
ESP_LOGI("TEST", "Test value abcd: %f",
mb_get_float_abcd(&holding_float_abcd[0]));
ESP_LOGI("TEST", "Test value abcd: %f",
mb_get_float_abcd(&holding_float_abcd[1]));
ESP_LOGI("TEST", "Test value ghefcdab: %lf",
mb_get_double_ghefcdab(&holding_double_ghefcdab[0]));
...
```

Параметри підлеглого зв'язку

Функція ініціалізує interface контролера Modbus і його активний контекст (завдання, об’єкти RTOS та інші ресурси).

mbc_slave_setup()

Ця функція використовується для налаштування параметрів зв’язку стеку Modbus.

Приклад ініціалізації зв’язку Modbus TCP:

```
esp_netif_init();
...

mb_communication_info_t comm_info = {
    .ip_port = MB_TCP_PORT,                    // Modbus TCP port number
(default = 502)
    .ip_addr_type = MB_IPV4,                   // version of IP protocol
    .ip_mode = MB_MODE_TCP,                    // Port communication mode
    .ip_addr = NULL,                           // This field keeps the
client IP address to bind, NULL - bind to any client
    .ip_netif_ptr = esp_netif_ptr              // esp_netif_ptr - pointer
to the corresponding network interface
```

```
};

// Setup communication parameters and start stack
ESP_ERROR_CHECK(mbc_slave_setup((void*)&comm_info));
```

Приклад ініціалізації послідовного зв’язку Modbus:

```
#define MB_SLAVE_DEV_SPEED 9600
#define MB_SLAVE_ADDR 1
#define MB_SLAVE_PORT_NUM 2
...

// Setup communication parameters and start stack
mb_communication_info_t comm_info = {
    .mode = MB_MODE_RTU,                    // Communication type
    .slave_addr = MB_SLAVE_ADDR,            // Short address of the slave
    .port = MB_SLAVE_PORT_NUM,              // UART physical port number
    .baudrate = MB_SLAVE_DEV_SPEED,         // Baud rate for communication
    .parity = MB_PARITY_NONE                // Parity option
};

ESP_ERROR_CHECK(mbc_slave_setup((void*)&comm_info));
```

Параметри зв’язку, які підтримує ця бібліотека, описані в розділі «Підтримувані параметри зв’язку Modbus» .

Однак можна перевизначити параметри послідовного зв’язку, викликавши функцію **uart_param_config()** одразу після **mbc_slave_setup()**.

Slave спілкування

Наведена нижче функція використовується для запуску interface контролера Modbus і забезпечує зв’язок.

mbc_slave_start()

```
ESP_ERROR_CHECK(mbc_slave_start());
```

mbc_slave_check_event()

Блокуючий виклик функції очікує на вказану подію (представлену як параметр маски події).

Після того, як майстер отримає доступ до параметра, а маска події збігається з типом параметра, завдання програми буде розблоковано, а функція поверне відповідну подію, **mb_event_group_t** яка описує тип доступу до реєстру.

mbc_slave_get_param_info()

Функція отримує інформацію про доступні параметри з черги подій контролера Modbus.

Ключ KConfig CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_SIZE можна використовувати для налаштування розміру черги сповіщень.

Параметр timeout дозволяє вказати час очікування під час очікування сповіщення.

Структура **mb_param_info_t** містить інформацію про доступний параметр.

Поле	**опис**
time_stamp	мітка часу події, коли здійснюється доступ до визначеного параметра
mb_offset	запустити реєстр Modbus, до якого має доступ майстер
type	тип області реєстру Modbus, до якої здійснюється доступ (див. **mb_event_group_t** для отримання додаткової інформації)
address	адреса пам'яті, яка відповідає регістру, до якого здійснюється доступ, у визначеному дескрипторі області
size	кількість регістрів, до яких має доступ майстер

Приклад отримання події під час утримання або введення регістрів, доступ до яких здійснюється в підпорядкованому пристрої:

```
#define MB_READ_MASK            (MB_EVENT_INPUT_REG_RD |
MB_EVENT_HOLDING_REG_RD)
#define MB_WRITE_MASK           (MB_EVENT_HOLDING_REG_WR)
#define MB_READ_WRITE_MASK      (MB_READ_MASK | MB_WRITE_MASK)
#define MB_PAR_INFO_GET_TOUT    (10 / portTICK_RATE_MS)
....

// The function blocks while waiting for register access
(void)mbc_slave_check_event(MB_READ_WRITE_MASK);

// Get information about data accessed from master
ESP_ERROR_CHECK(mbc_slave_get_param_info(&reg_info,
MB_PAR_INFO_GET_TOUT));
const char* rw_str = (reg_info.type & MB_READ_MASK) ? "READ" : "WRITE";

// Filter events and process them accordingly
if (reg_info.type & (MB_EVENT_HOLDING_REG_WR | MB_EVENT_HOLDING_REG_RD)) {
    ESP_LOGI(TAG, "HOLDING %s (%u us), ADDR:%u, TYPE:%u, INST_ADDR:0x%.4x,
SIZE:%u",
                rw_str,
                (uint32_t)reg_info.time_stamp,
                (uint32_t)reg_info.mb_offset,
                (uint32_t)reg_info.type,
                (uint32_t)reg_info.address,
                (uint32_t)reg_info.size);
} else if (reg_info.type & (MB_EVENT_INPUT_REG_RD)) {
    ESP_LOGI(TAG, "INPUT %s (%u us), ADDR:%u, TYPE:%u, INST_ADDR:0x%.4x,
SIZE:%u",
                rw_str,
                (uint32_t)reg_info.time_stamp,
                (uint32_t)reg_info.mb_offset,
                (uint32_t)reg_info.type,
                (uint32_t)reg_info.address,
                (uint32_t)reg_info.size);
}
```

Відключення Modbus Slave

Ця функція зупиняє комунікаційний стек Modbus, руйнує interface контролера та звільняє всі використані активні об’єкти, виділені для підлеглого.

mbc_slave_destroy()

```
ESP_ERROR_CHECK(mbc_slave_destroy());
```

Довідник API

Файл заголовка

freemodbus/common/include/esp_modbus_common.h

Союзи

union mb_communication_info_t

#include <esp_modbus_common.h>

Структура зв'язку пристрою для налаштування контролера Modbus.

Public члени

режим mb_mode_type_t

Режим зв'язку Modbus

uint8_t slave_addr

Поле адреси підлеглого пристрою Modbus (фіктивне для головного)

порт uart_port_t

Номер комунікаційного порту Modbus (UART).

швидкість передачі uint32_t

Швидкість передачі даних Modbus

uart_parity_t парність

Параметри паритету Modbus UART

uint16_t фіктивний_порт

Фіктивне поле, не використовується

***struct* mb_communication_info_t :: [анонім] [анонім]**

mb_mode_type_t ip_mode

Режим зв'язку Modbus

uint8_t slave_uid

Поле адреси підлеглого пристрою Modbus для UID

uint16_t ip_port

Порт Modbus

mb_tcp_addr_type_t ip_addr_type

Тип адреси Modbus

void * ip_addr

Таблиця адрес Modbus для підключення

void * ip_netif_ptr

Мережевий interface Modbus

***struct* mb_communication_info_t :: [анонім] [анонім]**

Макроси

MB_RETURN_ON_FALSE (a , код_помилки , тег , формат , ...)

MB_CONTROLLER_STACK_SIZE

MB_CONTROLLER_PRIORITY

MB_DEVICE_ADDRESS

MB_DEVICE_SPEED

MB_UART_PORT

MB_PAR_INFO_TOUT

MB_PARITY_NONE

_XFER_2_RD (dst , src)

_XFER_2_WR (dst , src)

Визначення типів

typedef esp_err_t (* iface_init) (void * *)

загальні типи методів interface.

typedef esp_err_t (* iface_destroy) (void)

Метод знищення interface.

typedef esp_err_t (* iface_setup) (void *)

Налаштування методу interface.

typedef esp_err_t (* iface_start) (void)

Запуск методу interface

Перерахування

enum mb_port_type_t

Типи фактичної реалізації Modbus.

Значення:

***enumerator* MB_PORT_SERIAL_MASTER**

Головний серійний порт типу Modbus.

***enumerator* MB_PORT_SERIAL_SLAVE**

Послідовний підлеглий порт типу Modbus.

***enumerator* MB_PORT_TCP_MASTER**

Тип порту Modbus TCP master.

***enumerator* MB_PORT_TCP_SLAVE**

Тип порту Modbus TCP slave.

***enumerator* MB_PORT_COUNT**

Кількість портів Modbus.

***enumerator* MB_PORT_INACTIVE**

enum mb_event_group_t

Група подій для повідомлення параметрів.

Значення:

***enumerator* MB_EVENT_NO_EVENTS**

***enumerator* MB_EVENT_HOLDING_REG_WR**

Регістри зберігання запису подій Modbus.

***enumerator* MB_EVENT_HOLDING_REG_RD**

Регістри утримання читання подій Modbus.

***enumerator* MB_EVENT_INPUT_REG_RD**

Вхідні регістри читання подій Modbus.

***enumerator* MB_EVENT_COILS_WR**

Котушки запису подій Modbus.

***enumerator* MB_EVENT_COILS_RD**

Котушки зчитування подій Modbus.

***enumerator* MB_EVENT_DISCRETE_RD**

Подія Modbus Читання дискретних бітів.

***enumerator* MB_EVENT_STACK_STARTED**

Запущено стек подій Modbus

enum mb_param_type_t

Тип параметра Modbus.

Значення:

***enumerator* MB_PARAM_HOLDING**

Реєстр Modbus Holding.

***enumerator* MB_PARAM_INPUT**

Регістр входу Modbus.

***enumerator* MB_PARAM_COIL**

Котушки Modbus.

***enumerator* MB_PARAM_DISCRETE**

Дискретні біти Modbus.

***enumerator* MB_PARAM_COUNT**

***enumerator* MB_PARAM_UNKNOWN**

enum mb_mode_type_t

Режими послідовної передачі Modbus (RTU/ASCII).

Значення:

***enumerator* MB_MODE_RTU**

Режим передачі RTU.

***enumerator* MB_MODE_ASCII**

Режим передачі ASCII.

***enumerator* MB_MODE_TCP**

Режим зв'язку TCP.

***enumerator* MB_MODE_UDP**

Режим зв'язку UDP.

enum mb_tcp_addr_type_t

Тип адреси Modbus TCP.

Значення:

***enumerator* MB_IPV4**

TCP IPV4 адресація

***enumerator* MB_IPV6**

TCP IPV6 адресація

Файл заголовка

freemodbus/common/include/esp_modbus_master.h

Функції

esp_err_t mbc_master_init_tcp (обробник void * *)

Ініціалізація контролера Modbus і стека для порту TCP.

Параметри

handler – **[out]** обробник (вказівник) на головну структуру даних

Повернення

- ESP_OK Успіх
- ESP_ERR_NO_MEM Помилка параметра
- ESP_ERR_NOT_SUPPORTED Тип порту не підтримується
- ESP_ERR_INVALID_STATE Помилка ініціалізації

esp_err_t mbc_master_init (mb_port_type_t тип_порту , void * * обробник)

Ініціалізація головного контролера Modbus і стека для послідовного порту.

Параметри

- **handler** – **[out]** обробник (вказівник) на головну структуру даних
- **port_type** – **[in]** тип стека

Повернення

- ESP_OK Успіх
- ESP_ERR_NO_MEM Помилка параметра
- ESP_ERR_NOT_SUPPORTED Тип порту не підтримується
- ESP_ERR_INVALID_STATE Помилка ініціалізації

void mbc_master_init_iface (void * обробник)

Ініціалізація interface контролера Modbus Master.

Параметри

handler – **[in]** - покажчик на головну структуру даних

esp_err_t mbc_master_destroy (void)

Знищити контролер і стек Modbus.

Повернення

- ESP_OK Успіх
- ESP_ERR_INVALID_STATE Помилка параметра

esp_err_t mbc_master_start (void)

Запустіть комунікаційний стек Modbus.

Повернення

- ESP_OK Успіх
- ESP_ERR_INVALID_ARG Помилка запуску стеку Modbus

esp_err_t mbc_master_setup (void * comm_info)

Встановіть параметри зв’язку Modbus для контролера.

Параметри

comm_info – структура параметрів зв'язку.

Повернення

- ESP_OK Успіх
- ESP_ERR_INVALID_ARG Неправильні дані параметра

esp_err_t mbc_master_set_descriptor (const mb_parameter_descriptor_t * дескриптор , const uint16_t num_elements)

Призначте таблицю опису параметрів для interface контролера Modbus.

Параметри

- **дескриптор** – **[in]** покажчик на таблицю опису параметрів
- **num_elements** – кількість елементів у таблиці

Повернення

- esp_err_t ESP_OK - успішно встановлено дескриптор
- esp_err_t ESP_ERR_INVALID_ARG - void аргумент у виклику функції

esp_err_t mbc_master_send_request (mb_param_request_t * запит , void * data_ptr)

Надіслати запит даних, як визначено в запиті параметрів, очікувати відповіді від підпорядкованого пристрою та повернути статус виконання команди.

Ця функція забезпечує стандартний спосіб доступу для читання/запису до пристроїв Modbus у мережі.

Параметри

- **запит** – покажчик **[in]** на структуру запиту типу mb_param_request_t
- **data_ptr** – **[in]** покажчик на буфер даних для надсилання або отримання даних (залежно від командного поля в запиті)

Повернення

- esp_err_t ESP_OK - запит виконано успішно
- esp_err_t ESP_ERR_INVALID_ARG - void аргумент функції
- esp_err_t ESP_ERR_INVALID_RESPONSE - недійсна відповідь від підлеглого
- esp_err_t ESP_ERR_TIMEOUT - тайм-аут операції або відсутність відповіді від slave
- esp_err_t ESP_ERR_NOT_SUPPORTED - команда запиту не підтримується підлеглим
- esp_err_t ESP_FAIL - підпорядкований повернув виняток або іншу помилку

esp_err_t mbc_master_get_cid_info (uint16_t cid , const mb_parameter_descriptor_t * * param_info)

Отримує інформацію про підтримувану характеристику, визначену як cid. Для отримання цієї інформації використовується таблиця опису параметрів.

Функція перевірить, чи підтримується характеристика, визначена як параметр cid, і поверне її опис у param_info.

Повертає ESP_ERR_NOT_FOUND, якщо характеристика не підтримується.

Параметри

- **cid** – **[in]** ідентифікатор характеристики
- **param_info** – покажчик на покажчик характерних даних.

Повернення

- esp_err_t ESP_OK - запит був успішним і буфер містить підтримувану назву характеристики
- esp_err_t ESP_ERR_INVALID_ARG - void аргумент функції
- esp_err_t ESP_ERR_NOT_FOUND - характеристика (cid) не знайдена
- esp_err_t ESP_FAIL - невідома помилка під час обробки таблиці пошуку

esp_err_t mbc_master_get_parameter (uint16_t cid , char * ім'я , uint8_t * значення , uint8_t * тип)

Читання параметра з підлеглого пристрою Modbus, ім'я якого визначається ім'ям і має cid. Додаткові дані для запиту беруться з таблиці опису (пошуку) параметрів.

Параметри

- **cid** – **[in]** id характеристики для параметра
- **name** – **[in]** вказівник на назву рядка (ключ) параметра (закінчується нулем)
- **значення** – покажчик **[out]** на буфер даних параметра
- **type** – **[out]** тип параметра, пов'язаний з іменем, що повертається з таблиці опису параметра.

Повернення

- esp_err_t ESP_OK - запит був успішним і буфер значень містить представлення фактичних даних параметрів від підлеглого
- esp_err_t ESP_ERR_INVALID_ARG - void аргумент функції або дескриптора параметра
- esp_err_t ESP_ERR_INVALID_RESPONSE - недійсна відповідь від підлеглого
- esp_err_t ESP_ERR_INVALID_STATE - void стан під час обробки даних або збій розподілу
- esp_err_t ESP_ERR_TIMEOUT - час очікування операції минув і немає відповіді від підлеглого
- esp_err_t ESP_ERR_NOT_SUPPORTED - команда запиту не підтримується підлеглим
- esp_err_t ESP_ERR_NOT_FOUND - параметр не знайдено в таблиці опису параметрів
- esp_err_t ESP_FAIL - підпорядкований повернув виняток або іншу помилку

esp_err_t mbc_master_set_parameter (uint16_t cid , char * ім’я , uint8_t * значення , uint8_t * тип)

Встановіть значення характеристики, визначене як ім'я та параметр cid. Додаткові дані для запиту параметра cid беруться з таблиці пошуку основних параметрів.

Параметри

- **cid** – **[in]** id характеристики для параметра
- **name** – **[in]** вказівник на назву рядка (ключ) параметра (закінчується нулем)
- **значення** – **[out]** покажчик на буфер даних параметра (фактичне представлення поля значення json у двійковій формі)
- **type** – **[out]** вказівник на тип параметра, пов’язаний з іменем, повернутим із таблиці пошуку параметрів.

Повернення

- esp_err_t ESP_OK - запит був успішним і значення було збережено в регістрах підлеглих пристроїв
- esp_err_t ESP_ERR_INVALID_ARG - void аргумент функції або дескриптора параметра
- esp_err_t ESP_ERR_INVALID_RESPONSE - недійсна відповідь від slave під час обробки параметра
- esp_err_t ESP_ERR_INVALID_STATE - void стан під час обробки даних або збій розподілу
- esp_err_t ESP_ERR_TIMEOUT - час очікування операції минув і немає відповіді від підлеглого
- esp_err_t ESP_ERR_NOT_SUPPORTED - команда запиту не підтримується підлеглим
- esp_err_t ESP_FAIL - підпорядкований повернув виняток або іншу помилку

esp_err_t mbc_master_set_param_data (void * dest , void * src , mb_descr _type_t param_type , size_t param_size)

Допоміжна функція для встановлення даних параметрів відповідно до їх типу.

Параметри

- **dest** – **[in]** адреса призначення параметра
- **src** – **[in]** адреса джерела параметра
- **param_type** – **[out]** тип параметра зі словника даних
- **param_size** – **[out]** розмір зберігання характеристики (в байтах). Описує розмір даних, які потрібно зберігати в екземплярі даних під час зіставлення.

Повернення

- esp_err_t ESP_OK - запит був успішним і значення було збережено в регістрах підлеглих пристроїв
- esp_err_t ESP_ERR_INVALID_ARG - void аргумент функції або дескриптора параметра
- esp_err_t ESP_ERR_NOT_SUPPORTED - команда запиту не підтримується підлеглим

Союзи

union mb_parameter_opt_t

#include <esp_modbus_master.h>

Варіанти параметрів Modbus для таблиці опису.

Public члени

int opt1

Параметр option1

int opt2

Параметр option2

int opt3

Варіант параметра3

***struct* mb_parameter_opt_t :: [анонім] [анонім]**

int хв

Мінімальне значення параметра

int макс

Максимальне значення параметра

внутр . крок

Крок відстеження зміни параметрів

***struct* mb_parameter_opt_t :: [анонім] [анонім]**

Структури

struct mb_parameter_descriptor_t

Тип дескриптора характеристик використовується для опису характеристики та зв’язування її з параметрами Modbus, які відображають її дані.

Public члени

uint16_t cid

Характеристика cid

***const* char * param_key**

Ключ (ім'я) параметра

***const* char * param_units**

Фізичні одиниці параметра

uint8_t mb_slave_addr

Адреса веденого пристрою в сегменті Modbus

mb_param_type_t mb_param_type

Тип параметра modbus

uint16_t mb_reg_start

Це адреса реєстру Modbus. Це значення на основі 0.

uint16_t mb_size

Розмір параметра mb в регістрах

uint16_t param_offset

Назва параметра (ЗМІЩЕННЯ в структурі параметра)

mb_descr_type_t тип_параметра

Float, U8, U16, U32, ASCII тощо.

mb_descr_size_t param_size

Кількість байтів у параметрі.

mb_parameter_opt_t param_opts

Параметри, що використовуються для перевірки лімітів тощо.

доступ mb_param_perms_t

Права доступу залежно від режиму

struct mb_param_request_t

Структура типу запиту на реєстр Modbus.

Public члени

uint8_t slave_addr

Підпорядкована адреса Modbus

команда uint8_t

Команда Modbus для надсилання

uint16_t reg_start

Реєстр запуску Modbus

uint16_t reg_size

Кількість регістрів Modbus

Макрос для доступу до масивів елементів для перетворення типів.

Перерахування

enum mb_descr_type_t

Тип параметра таблиці дескрипторів Modbus визначає.

Значення:

***enumerator* PARAM_TYPE_U8**

Без підпису 8

***enumerator* PARAM_TYPE_U16**

Без підпису 16

***enumerator* PARAM_TYPE_U32**

Без підпису 32

***enumerator* PARAM_TYPE_FLOAT**

Поплавковий тип

***enumerator* PARAM_TYPE_ASCII**

Тип ASCII

***enumerator* PARAM_TYPE_BIN**

Тип BIN

***enumerator* PARAM_TYPE_I8_A**

I8 ціле число зі знаком у старшому байті регістра

***enumerator* PARAM_TYPE_I8_B**

I8 ціле число зі знаком у молодшому байті регістра

***enumerator* PARAM_TYPE_U8_A**

U8 беззнакове ціле число, записане в hi-байт регістра

***enumerator* PARAM_TYPE_U8_B**

U8 беззнакове ціле число, записане в молодший байт регістра

***enumerator* PARAM_TYPE_I16_AB**

I16 ціле число зі знаком, порядок байтів

***enumerator* PARAM_TYPE_I16_BA**

I16 ціле число зі знаком, порядок байтів

***enumerator* PARAM_TYPE_U16_AB**

U16 беззнакове ціле число, великий порядок байтів

***enumerator* PARAM_TYPE_U16_BA**

U16 ціле число без знаку, порядок байтів

***enumerator* PARAM_TYPE_I32_ABCD**

I32 ABCD ціле число зі знаком, великий порядок байтів

***enumerator* PARAM_TYPE_I32_CDAB**

I32 Ціле число зі знаком CDAB, старший порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_I32_BADC**

I32 Ціле число зі знаком BADC, порядок порядків байтів у зворотному порядку

***enumerator* PARAM_TYPE_I32_DCBA**

I32 DCBA ціле число зі знаком, порядок байтів

***enumerator* PARAM_TYPE_U32_ABCD**

U32 ABCD беззнакове ціле число, великий порядок байтів

***enumerator* PARAM_TYPE_U32_CDAB**

U32 CDAB беззнакове ціле число, старший порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_U32_BADC**

U32 BADC беззнакове ціле число, порядковий порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_U32_DCBA**

U32 DCBA ціле число без знаку, порядок байтів

***enumerator* PARAM_TYPE_FLOAT_ABCD**

Float ABCD з плаваючою комою, старший порядок байтів

***enumerator* PARAM_TYPE_FLOAT_CDAB**

Float CDAB з плаваючою точкою старший порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_FLOAT_BADC**

Число з плаваючою точкою BADC, порядок байтів у порядку байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_FLOAT_DCBA**

Float DCBA з плаваючою точкою, маленький порядок байтів

***enumerator* PARAM_TYPE_I64_ABCDEFGH**

I64, ABCDEFGH ціле число зі знаком, старший порядок байтів

***enumerator* PARAM_TYPE_I64_HGFEDCBA**

I64, HGFEDCBA ціле число зі знаком, порядок байтів

***enumerator* PARAM_TYPE_I64_GHEFCDAB**

I64, ціле число зі знаком GHEFCDAB, старший порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_I64_BADCFEHG**

I64, ціле число зі знаком BADCFEHG, порядок байтів від маленького байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_U64_ABCDEFGH**

U64, ABCDEFGH беззнакове ціле число, великий порядок байтів

***enumerator* PARAM_TYPE_U64_HGFEDCBA**

U64, HGFEDCBA Ціле число без знака, порядок байтів

***enumerator* PARAM_TYPE_U64_GHEFCDAB**

U64, GHEFCDAB беззнакове ціле число, старший порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_U64_BADCFEHG**

U64, BADCFEHG беззнакове ціле число, порядковий порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_DOUBLE_ABCDEFGH**

Double ABCDEFGH з плаваючою комою, старший порядок байтів

***enumerator* PARAM_TYPE_DOUBLE_HGFEDCBA**

Double HGFEDCBA з плаваючою комою, маленький порядок байтів

***enumerator* PARAM_TYPE_DOUBLE_GHEFCDAB**

Double GHEFCDAB з плаваючою комою, старший порядок байтів, зворотний порядок регістрів

***enumerator* PARAM_TYPE_DOUBLE_BADCFEHG**

Double BADCFEHG з плаваючою комою, маленький порядок байтів, зворотний порядок регістрів

enum mb_descr_size_t

Розмір параметра таблиці дескриптора Modbus у байтах.

Значення:

***enumerator* PARAM_SIZE_U8**

Без підпису 8

***enumerator* PARAM_SIZE_U8_REG**

Беззнаковий 8, регістрове значення

***enumerator* PARAM_SIZE_I8_REG**

Підпис 8, реєстрове значення

***enumerator* PARAM_SIZE_I16**

Без підпису 16

***enumerator* PARAM_SIZE_U16**

Без підпису 16

***enumerator* PARAM_SIZE_I32**

Підписано 32

***enumerator* PARAM_SIZE_U32**

Без підпису 32

***enumerator* PARAM_SIZE_FLOAT**

Поплавок 32 розміру

***enumerator* PARAM_SIZE_ASCII**

Розмір ASCII за замовчуванням

***enumerator* PARAM_SIZE_ASCII24**

Розмір ASCII24

***enumerator* PARAM_SIZE_I64**

Ціле число зі знаком 64 розміру

***enumerator* PARAM_SIZE_U64**

Беззнакове ціле число 64 розміру

***enumerator* PARAM_SIZE_DOUBLE**

Double 64 розмір

***enumerator* PARAM_MAX_SIZE**

enum mb_param_perms_t

Дозволи на характеристики.

Значення:

***enumerator* PAR_PERMS_READ**

характеристики пристрою читаються

***enumerator* PAR_PERMS_WRITE**

характеристика пристрою є записуваною

***enumerator* PAR_PERMS_TRIGGER**

характеристика пристрою є тригерною

***enumerator* PAR_PERMS_READ_WRITE**

характеристика пристрою читання та запису

***enumerator* PAR_PERMS_READ_TRIGGER**

характеристики пристрою читаються та запускаються

***enumerator* PAR_PERMS_WRITE_TRIGGER**

характеристикою пристрою є можливість запису та запуску

***enumerator* PAR_PERMS_READ_WRITE_TRIGGER**

характеристикою пристрою є можливість читання, запису та запуску

Файл заголовка

freemodbus/common/include/esp_modbus_slave.h

Функції

esp_err_t mbc_slave_init_tcp (обробник void * *)

Ініціалізація веденого контролера Modbus і стека для порту TCP.

Параметри

handler – **[out]** обробник (вказівник) на головну структуру даних

Повернення

- ESP_OK Успіх
- ESP_ERR_NO_MEM Помилка параметра
- ESP_ERR_NOT_SUPPORTED Тип порту не підтримується
- ESP_ERR_INVALID_STATE Помилка ініціалізації

esp_err_t mbc_slave_init (mb_port_type_t port_type , void * * обробник)

Ініціалізація веденого контролера Modbus і стека для послідовного порту.

Параметри

- **handler** – **[out]** обробник (вказівник) на головну структуру даних
- **port_type** – **[in]** тип порту

Повернення

- ESP_OK Успіх
- ESP_ERR_NO_MEM Помилка параметра
- ESP_ERR_NOT_SUPPORTED Тип порту не підтримується
- ESP_ERR_INVALID_STATE Помилка ініціалізації

void mbc_slave_init_iface (void * обробник)

Ініціалізація interface підлеглого контролера Modbus.

Параметри

handler – **[in]** - покажчик на структуру даних підлеглого interface

esp_err_t mbc_slave_destroy (void)

Знищити контролер і стек Modbus.

Повернення

- ESP_OK Успіх
- ESP_ERR_INVALID_STATE Помилка параметра

esp_err_t mbc_slave_start (void)

Запустіть комунікаційний стек Modbus.

Повернення

- ESP_OK Успіх
- ESP_ERR_INVALID_ARG Помилка запуску стеку Modbus

esp_err_t mbc_slave_setup (void * comm_info)

Встановіть параметри зв’язку Modbus для контролера.

Параметри

comm_info – структура параметрів зв'язку.

Повернення

- ESP_OK Успіх
- ESP_ERR_INVALID_ARG Неправильні дані параметра

mb_event_group_t mbc_slave_check_event (група mb_event_group_t)

Очікуйте певної події при зміні параметра.

Параметри

група – бітова маска події групи, щоб очікувати зміни

Повернення

- mb_event_group_t активовані біти події

esp_err_t mbc_slave_get_param_info (mb_param_info_t * reg_info , час очікування uint32_t)

Отримати інформацію про параметри.

Параметри

- **reg_info** – інформаційна структура параметра **[out].**
- **тайм-аут** – час очікування в мілісекундах для читання інформації з черги параметрів

Повернення

- ESP_OK Успіх
- ESP_ERR_TIMEOUT Неможливо отримати дані з черги параметрів або переповнення черги

esp_err_t mbc_slave_set_descriptor (mb_register_area_descriptor_t descr_data)

Встановити дескриптор області Modbus.

Параметри

descr_data – структура дескриптора області реєстрів Modbus

Повернення

- ESP_OK: встановлено відповідний дескриптор
- ESP_ERR_INVALID_ARG: аргумент невірний

Структури

struct mb_param_info_t

Тип інформації про подію доступу до параметра.

Public члени

uint32_t time_stamp

Мітка часу події Modbus (США)

uint16_t mb_offset

Зміщення регістра Modbus

тип mb_event_group_t

Тип події Modbus

uint8_t * адреса

Адреса зберігання даних Modbus

size_t розмір

Розмір регістра подій Modbus (кількість регістрів)

struct mb_register_area_descriptor_t

Дескриптор області зберігання параметрів.

Public члени

uint16_t початковий_зсув

Початкова адреса Modbus для дескриптора області

тип mb_param_type_t

Тип дескриптора області зберігання

недійсна * адреса

Адреса екземпляра для дескриптора області зберігання

size_t розмір

Розмір екземпляра для дескриптора області (байти)

Довідка по API перетворення Modbus Endianness

Файл заголовка

freemodbus/common/include/mb_endianness_utils.h

Функції

int8_t mb_get_int8_a (val_16_arr * pi16)

Отримати представлення значення int8_t (молодший байт) із реєстру.

Повернення

- int8_t значення, перетворене зі значення реєстру

uint16_t mb_set_int8_a (val_16_arr * pi16 , int8_t i8)

Встановіть значення i8 у значення регістра, на яке вказує pi16.

Повернення

- uint16_t значення, яке представляє фактичне шістнадцяткове значення регістру

int8_t mb_get_int8_b (val_16_arr * pi16)

Отримайте значення int8_t (старший байт) із значення регістра, на яке вказує pi16.

Повернення

- uint16_t значення, яке представляє фактичне шістнадцяткове значення регістру

uint16_t mb_set_int8_b (val_16_arr * pi16 , int8_t i8)

Установіть значення i8 (старший байт) зі значення регістра, на яке вказує pi16.

Повернення

- uint16_t значення, яке представляє фактичне шістнадцяткове значення регістру

uint8_t mb_get_uint8_a (val_16_arr * pu16)

Отримайте представлення значення uint8_t (молодший байт) із регістру, на який вказує pu16.

Повернення

- uint8_t значення, перетворене зі значення реєстру

uint16_t mb_set_uint8_a (val_16_arr * pu16 , uint8_t u8)

Встановіть значення u8 (молодший байт) у значення регістра, на яке вказує pu16.

Повернення

- uint16_t значення, яке представляє фактичне шістнадцяткове значення регістра

uint8_t mb_get_uint8_b (val_16_arr * pu16)

Отримайте значення uint8_t (старший байт) із значення регістра, на яке вказує pu16.

Повернення

- uint16_t значення, яке представляє фактичне шістнадцяткове значення регістра

uint16_t mb_set_uint8_b (val_16_arr * pu16 , uint8_t u8)

Встановіть значення u8 (старший байт) у значення регістра, на яке вказує pu16.

Повернення

- uint16_t значення, яке представляє фактичне шістнадцяткове значення регістра

int16_t mb_get_int16_ab (val_16_arr * pi16)

Отримайте значення int16_t зі значення регістру, на яке вказує pu16, з порядком байтів ab.

Повернення

- int16_t значення, яке представляє перетворене значення з реєстру

uint16_t mb_set_int16_ab (val_16_arr * pi16 , int16_t i16)

Встановіть значення i16 у регістр, на який вказує pi16, із порядком байтів ab.

Повернення

- int16_t значення, яке представляє перетворене значення з реєстру

uint16_t mb_get_uint16_ab (val_16_arr * pu16)

Отримайте значення uint16_t зі значення регістру, на яке вказує pu16, із порядком байтів ab.

Повернення

- uint16_t значення, яке представляє перетворене значення регістра

uint16_t mb_set_uint16_ab (val_16_arr * pu16 , uint16_t u16)

Встановіть значення u16 у регістр, на який вказує pu16, з порядком байтів ab.

Повернення

- uint16_t значення, яке представляє перетворене значення з реєстру

int16_t mb_get_int16_ba (val_16_arr * pi16)

Отримайте значення int16_t зі значення регістру, на яке вказує pu16, з порядком байтів ba.

Повернення

- Значення int16_t, яке представляє перетворене значення регістра

uint16_t mb_set_int16_ba (val_16_arr * pi16 , int16_t i16)

Встановіть значення i16 у регістр, на який вказує pi16, з порядком байтів ba.

Повернення

- uint16_t значення, яке представляє перетворене значення з реєстру

uint16_t mb_get_uint16_ba (val_16_arr * pu16)

Отримайте значення uint16_t зі значення регістру, на яке вказує pu16, з порядком байтів ba.

Повернення

- uint16_t значення, яке представляє перетворене значення регістра

uint16_t mb_set_uint16_ba (val_16_arr * pu16 , uint16_t u16)

Установіть значення u16 у регістр, на який вказує pu16, з порядком байтів ba.

Повернення

- uint16_t значення, яке представляє перетворене значення з реєстру

int32_t mb_get_int32_abcd (val_32_arr * pi32)

Отримайте значення int32_t зі значення регістру, на яке вказує pi32, із порядком байтів abcd.

Повернення

- значення int32_t, яке представляє перетворене значення регістра

uint32_t mb_set_int32_abcd (val_32_arr * pi32 , int32_t i32)

Установіть значення i32 у регістр, на який вказує pi32, із порядком байтів abcd.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

uint32_t mb_get_uint32_abcd (val_32_arr * pu32)

Отримайте значення uint32_t зі значення реєстру, на яке вказує pu32, із порядком байтів abcd.

Повернення

- uint32_t значення, яке представляє перетворене значення регістра

uint32_t mb_set_uint32_abcd (val_32_arr * pu32 , uint32_t u32)

Встановіть значення u32 у регістр, на який вказує pu32, з порядком байтів abcd.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

int32_t mb_get_int32_badc (val_32_arr * pi32)

Отримайте значення int32_t зі значення регістра, на яке вказує pi32, з порядком байтів badc.

Повернення

- значення int32_t, яке представляє перетворене значення регістра

uint32_t mb_set_int32_badc (val_32_arr * pi32 , int32_t i32)

Встановіть значення i32 у регістр, на який вказує pi32, з порядком байтів badc.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

uint32_t mb_get_uint32_badc (val_32_arr * pu32)

Отримайте значення uint32_t зі значення реєстру, на яке вказує pu32, з порядком байтів badc.

Повернення

- unt32_t значення, яке представляє перетворене значення регістра

uint32_t mb_set_uint32_badc (val_32_arr * pu32 , uint32_t u32)

Встановіть значення u32 у регістр, на який вказує pu32, із порядком байтів badc.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

int32_t mb_get_int32_cdab (val_32_arr * pi32)

Отримайте значення int32_t зі значення регістру, на яке вказує pi32, із порядком байтів cdab.

Повернення

- значення int32_t, яке представляє перетворене значення регістра

uint32_t mb_set_int32_cdab (val_32_arr * pi32 , int32_t i32)

Встановіть значення i32 у регістр, на який вказує pi32, із порядком байтів cdab.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

uint32_t mb_get_uint32_cdab (val_32_arr * pu32)

Отримайте значення uint32_t зі значення реєстру, на яке вказує pu32, із порядком байтів cdab.

Повернення

- значення int32_t, яке представляє перетворене значення регістра

uint32_t mb_set_uint32_cdab (val_32_arr * pu32 , uint32_t u32)

Встановіть значення u32 у регістр, на який вказує pu32, із порядком байтів cdab.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

int32_t mb_get_int32_dcba (val_32_arr * pi32)

Отримайте значення int32_t зі значення регістру, на яке вказує pi32, із порядком байтів dcba.

Повернення

- значення int32_t, яке представляє перетворене значення регістра

uint32_t mb_set_int32_dcba (val_32_arr * pi32 , int32_t i32)

Встановіть значення i32 у регістр, на який вказує pi32, з dcba endianness.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

uint32_t mb_get_uint32_dcba (val_32_arr * pu32)

Отримайте значення uint32_t зі значення реєстру, на яке вказує pu32, з dcba endianness.

Повернення

- uint32_t значення, яке представляє перетворене значення регістра

uint32_t mb_set_uint32_dcba (val_32_arr * pu32 , uint32_t u32)

Встановіть значення u32 у регістр, на який вказує pu32, з dcba endianness.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

float mb_get_float_abcd (val_32_arr * pf)

Отримати значення з плаваючою точкою з регістра, на який вказує pf, із порядком байтів abcd.

Повернення

- значення float, яке представляє перетворене значення регістра

uint32_t mb_set_float_abcd (val_32_arr * pf , float f)

Встановити значення f для регістра, на який вказує pf, із порядком байтів abcd.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

float mb_get_float_badc (val_32_arr * pf)

Отримайте значення з плаваючою точкою з регістру, на який вказує pf, із порядком байтів badc.

Повернення

- значення float, яке представляє перетворене значення регістра

uint32_t mb_set_float_badc (val_32_arr * pf , float f)

Встановіть значення f для регістру, на який вказує pf, із порядком байтів badc.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

float mb_get_float_cdab (val_32_arr * pf)

Отримайте значення з плаваючою речовиною з регістру, на який вказує pf, із порядком байтів cdab.

Повернення

- значення float, яке представляє перетворене значення регістра

uint32_t mb_set_float_cdab (val_32_arr * pf , float f)

Установіть значення f для регістру, на який вказує pf, із порядком байтів cdab.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

float mb_get_float_dcba (val_32_arr * pf)

Отримати значення з плаваючою точкою з регістру, на який вказує pf, із порядком байтів dcba.

Повернення

- значення float, яке представляє перетворене значення регістра

uint32_t mb_set_float_dcba (val_32_arr * pf , float f)

Встановіть значення f для регістру, на який вказує pf, із порядком байтів dcba.

Повернення

- Значення uint32_t, яке представляє перетворене значення з реєстру

double mb_get_double_abcdefgh (val_64_arr * pd)

Отримайте подвійне значення з реєстру, на який вказує pd, із порядком байтів abcdefgh.

Повернення

- подвійне значення, яке представляє перетворене значення регістра

uint64_t mb_set_double_abcdefgh (val_64_arr * pd , double d)

Встановіть значення d для регістра, на який вказує pd, із порядком байтів abcdefgh.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

double mb_get_double_hgfedcba (val_64_arr * pd)

Отримайте подвійне значення з реєстру, на який вказує pd, за допомогою байтів hgfedcba.

Повернення

- подвійне значення, яке представляє перетворене значення регістра

uint64_t mb_set_double_hgfedcba (val_64_arr * pd , double d)

Установіть значення d для регістру, на який вказує pd, із порядком байтів hgfedcba.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

double mb_get_double_ghefcdab (val_64_arr * pd)

Отримайте подвійне значення з реєстру, на який вказує pd, за допомогою байтів ghefcdab.

Повернення

- подвійне значення, яке представляє перетворене значення регістра

uint64_t mb_set_double_ghefcdab (val_64_arr * pd , double d)

Установіть значення d для регістру, на який вказує pd, із порядком байтів ghefcdab.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

double mb_get_double_badcfehg (val_64_arr * pd)

Отримайте подвійне значення з реєстру, на який вказує pd, з порядком байтів badcfehg.

Повернення

- подвійне значення, яке представляє перетворене значення регістра

uint64_t mb_set_double_badcfehg (val_64_arr * pd , double d)

Установіть значення d для регістру, на який вказує pd, із порядком байтів badcfehg.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

int64_t mb_get_int64_abcdefgh (val_64_arr * pi64)

Отримайте значення int64_t із реєстру, на який вказує pi64, із порядком байтів abcdefgh.

Повернення

- Значення int64_t, яке представляє перетворене значення регістра

uint64_t mb_set_int64_abcdefgh (val_64_arr * pi , int64_t i)

Встановіть значення i для регістру, на який вказує pi, із порядком байтів abcdefgh.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

int64_t mb_get_int64_ghefcdab (val_64_arr * pi64)

Отримайте значення int64_t із реєстру, на який вказує pi64, із порядком байтів ghefcdab.

Повернення

- Значення int64_t, яке представляє перетворене значення регістра

uint64_t mb_set_int64_ghefcdab (val_64_arr * pi , int64_t i)

Встановіть значення i для регістру, на який вказує pi, із порядком байтів ghefcdab.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

int64_t mb_get_int64_hgfedcba (val_64_arr * pi64)

Отримайте значення int64_t із реєстру, на який вказує pi64, із порядком байтів hgfedcba.

Повернення

- Значення int64_t, яке представляє перетворене значення регістра

uint64_t mb_set_int64_hgfedcba (val_64_arr * pi , int64_t i)

Встановіть значення i для регістру, на який вказує pi, із порядком байтів hgfedcba.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

int64_t mb_get_int64_badcfehg (val_64_arr * pi64)

Отримайте значення int64_t з регістру, на який вказує pi64, з порядком байтів badcfehg.

Повернення

- Значення int64_t, яке представляє перетворене значення регістра

uint64_t mb_set_int64_badcfehg (val_64_arr * pi , int64_t i)

Встановіть значення i для регістру, на який вказує pi, з порядком байтів badcfehg.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

uint64_t mb_get_uint64_abcdefgh (val_64_arr * pui)

Отримайте значення uint64_t із реєстру, на який вказує pui, із порядком байтів abcdefgh.

Повернення

- uint64_t значення, яке представляє перетворене значення регістра

uint64_t mb_set_uint64_abcdefgh (val_64_arr * pui , uint64_t ui)

Установіть значення ui для регістру, на який вказує pi, із порядком байтів abcdefgh.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

uint64_t mb_get_uint64_hgfedcba (val_64_arr * pui)

Отримайте значення uint64_t із реєстру, на який вказує pui, із порядком байтів hgfedcba.

Повернення

- uint64_t значення, яке представляє перетворене значення регістра

uint64_t mb_set_uint64_hgfedcba (val_64_arr * pui , uint64_t ui)

Установіть значення ui для регістру, на який вказує pui, з порядком байтів hgfedcba.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

uint64_t mb_get_uint64_ghefcdab (val_64_arr * pui)

Отримайте значення uint64_t із реєстру, на який вказує pui, із порядком байтів ghefcdab.

Повернення

- uint64_t значення, яке представляє перетворене значення регістра

uint64_t mb_set_uint64_ghefcdab (val_64_arr * pui , uint64_t ui)

Встановіть значення ui для регістру, на який вказує pui, з порядком байтів ghefcdab.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

uint64_t mb_get_uint64_badcfehg (val_64_arr * pui)

Отримайте значення uint64_t з реєстру, на який вказує pui, з порядком байтів badcfehg.

Повернення

- uint64_t значення, яке представляє перетворене значення регістра

uint64_t mb_set_uint64_badcfehg (val_64_arr * pui , uint64_t ui)

Установіть значення ui для регістру, на який вказує pui, з порядком байтів badcfehg.

Повернення

- Значення uint64_t, яке представляє перетворене значення з реєстру

Макроси

MB_BO16_0

Визначає постійні значення на основі власного порядку байтів компілятора.

MB_BO16_1

MB_BO32_0

MB_BO32_1

MB_BO32_2

MB_BO32_3

MB_BO64_0

MB_BO64_1

MB_BO64_2

MB_BO64_3

MB_BO64_4

MB_BO64_5

MB_BO64_6

MB_BO64_7

Визначення типів

typedef uint8_t val_16_arr [2]

Типи масивів розміру, які використовуються для відображення розширених значень.

typedef uint8_t val_32_arr [4]

typedef uint8_t val_64_arr [8]

ЛОКАЛЬНЕ КЕРУВАННЯ ESP

Огляд

Компонент локального керування ESP (**esp_local_ctrl**) у ESP-IDF забезпечує можливість керування пристроєм ESP через HTTPS або Bluetooth® Low Energy.

Він надає доступ до визначених програмою **властивостей** , які доступні для читання/запису через набір настроюваних обробників.

Ініціалізація служби **esp_local_ctrl** через Bluetooth Low Energy Transport виконується таким чином:

```
esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_BLE,
    .transport_config = {
        .ble = & (protocomm_ble_config_t) {
            .device_name  = SERVICE_NAME,
            .service_uuid = {
                /* LSB <---------------------------------------
                * ---------------------------------------> MSB */
                0x21, 0xd5, 0x3b, 0x8d, 0xbd, 0x75, 0x68, 0x8a,
                0xb4, 0x42, 0xeb, 0x31, 0x4a, 0x1e, 0x98, 0x3d
            }
        }
    },
    .proto_sec = {
        .version = PROTOCOM_SEC0,
        .custom_handle = NULL,
        .sec_params = NULL,
    },
    .handlers = {
        /* User defined handler functions */
        .get_prop_values = get_property_values,
        .set_prop_values = set_property_values,
        .usr_ctx         = NULL,
        .usr_ctx_free_fn = NULL
    },
    /* Maximum number of properties that may be set */
    .max_properties = 10
};

/* Start esp_local_ctrl service */
ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
```

Ініціалізація служби **esp_local_ctrl** через транспорт HTTPS виконується таким чином:

```
/* Set the configuration */
httpd_ssl_config_t https_conf = HTTPD_SSL_CONFIG_DEFAULT();

/* Load server certificate */
extern const unsigned char servercert_start[]
asm("_binary_servercert_pem_start");
extern const unsigned char servercert_end[]
asm("_binary_servercert_pem_end");
https_conf.servercert = servercert_start;
https_conf.servercert_len = servercert_end - servercert_start;

/* Load server private key */
extern const unsigned char prvtkey_pem_start[]
asm("_binary_prvtkey_pem_start");
extern const unsigned char prvtkey_pem_end[]
asm("_binary_prvtkey_pem_end");
https_conf.prvtkey_pem = prvtkey_pem_start;
https_conf.prvtkey_len = prvtkey_pem_end - prvtkey_pem_start;

esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_HTTPD,
    .transport_config = {
        .httpd = &https_conf
    },
    .proto_sec = {
        .version = PROTOCOM_SEC0,
        .custom_handle = NULL,
        .sec_params = NULL,
    },
    .handlers = {
        /* User defined handler functions */
        .get_prop_values = get_property_values,
        .set_prop_values = set_property_values,
        .usr_ctx         = NULL,
        .usr_ctx_free_fn = NULL
    },
    /* Maximum number of properties that may be set */
    .max_properties = 10
};

/* Start esp_local_ctrl service */
ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
```

Можна встановити безпеку для транспортування в локальному контролі ESP за допомогою таких параметрів:

- `PROTOCOM_SEC2`: вказує, що використовується обмін ключами на основі SRP6a та наскрізне шифрування на основі AES-GCM. Це найбільш бажаний варіант, оскільки він додає надійну безпеку за допомогою протоколу Augmented PAKE, тобто SRP6a.
- `PROTOCOM_SEC1`: вказує, що використовується обмін ключами на основі Curve25519 і наскрізне шифрування на основі AES-CTR.
- `PROTOCOM_SEC0`: вказує, що дані обмінюватимуться як простий текст (без безпеки).
- `PROTOCOM_SEC_CUSTOM`: ви можете визначити власні вимоги безпеки. Зверніть увагу, що вам також потрібно буде вказати `custom_handle` тип у цьому контексті. `protocomm_security_t *`

Створення властивості

Кожна властивість повинна мати унікальний `name` (рядок), а `type` (наприклад, enum), `flags` (бітові поля) і `size`.

Потрібно `size` залишити 0, якщо ми хочемо, щоб значення нашої властивості було змінної довжини (наприклад, якщо це рядок або потік байтів).

Для типів даних зі значенням властивості фіксованої довжини, як-от int, float тощо, встановлення `size` правильного значення поля допомагає **esp_local_ctrl** виконувати внутрішні перевірки аргументів, отриманих із запитами на запис.

Інтерпретація полів `type` і `flags` повністю залежить від програми, тому їх можна використовувати як перерахування, бітові поля або навіть прості цілі числа.

Ось приклад властивості, яка має функціонувати як позначка часу. Передбачається, що програма визначає `TYPE_TIMESTAMP` і `READONLY`, які використовуються для налаштування полів `type` і `flags`.

```
/* Create a timestamp property */
esp_local_ctrl_prop_t timestamp = {
    .name        = "timestamp",
    .type        = TYPE_TIMESTAMP,
    .size        = sizeof(int32_t),
    .flags       = READONLY,
    .ctx         = func_get_time,
    .ctx_free_fn = NULL
};

/* Now register the property */
esp_local_ctrl_add_property(&timestamp);
```

Також зауважте, що є поле ctx, яке вказує на певний власний *func_get_time()* . Це можна використовувати в обробниках get/set для отримання позначки часу.

Приклад `get_prop_values()` обробника, який використовується для отримання мітки часу.

```
static esp_err_t get_property_values(size_t props_count,
                                     const esp_local_ctrl_prop_t *props,
                                     esp_local_ctrl_prop_val_t
*prop_values,
                                     void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        ESP_LOGI(TAG, "Reading %s", props[i].name);
        if (props[i].type == TYPE_TIMESTAMP) {
            /* Obtain the timer function from ctx */
            int32_t (*func_get_time)(void) = props[i].ctx;

            /* Use static variable for saving the value. This is essential
because the value has to be valid even after this function returns.
Alternative is to use dynamic allocation and set the free_fn field */
            static int 32_t ts = func_get_time();
            prop_values[i].data = &ts;
        }
    }
    return ESP_OK;
}
```

Приклад `set_prop_values()` обробника.

```
static esp_err_t set_property_values(size_t props_count,
                                     const esp_local_ctrl_prop_t *props,
                                     const       esp_local_ctrl_prop_val_t
*prop_values,
                                     void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        if (props[i].flags & READONLY) {
            ESP_LOGE(TAG,  "Cannot  write  to  read-only  property  %s",
props[i].name);
            return ESP_ERR_INVALID_ARG;
        } else {
            ESP_LOGI(TAG, "Setting %s", props[i].name);

            /* For keeping it simple, lets only log the incoming data */
            ESP_LOG_BUFFER_HEX_LEVEL(TAG, prop_values[i].data,
                                     prop_values[i].size, ESP_LOG_INFO);
        }
    }
    return ESP_OK;
}
```

Впровадження на стороні клієнта

Реалізація на стороні клієнта спочатку встановлює сеанс протоколу зв'язку з пристроєм через підтримуваний режим транспортування, а потім надсилає та отримує повідомлення protobuf, які розуміє служба **esp_local_ctrl** .

Служба перетворює ці повідомлення в запити, а потім викликає відповідні обробники (set/get).

Потім згенерована відповідь для кожного обробника знову упаковується в повідомлення protobuf і передається назад клієнту.

Дивіться нижче різні повідомлення protobuf, які розуміє служба **esp_local_ctrl** :

- `get_prop_count`: це має просто повернути загальну кількість властивостей, які підтримує служба.

- `get_prop_values`: приймає масив індексів і має повертати інформацію (ім’я, тип, прапорці) і значення властивостей, що відповідають цим індексам.
- `set_prop_values`: приймає масив індексів і масив нових значень, які використовуються для встановлення значень властивостей, що відповідають індексам.

Індекси можуть або не можуть бути однаковими для властивості в кількох сеансах.

Таким чином, клієнт повинен використовувати лише назви властивостей для їх унікальної ідентифікації, кожного разу, коли встановлюється новий сеанс, клієнт повинен спочатку викликати `get_prop_count`, а потім `get_prop_values`, отже, формувати відображення індексу на ім’я для всіх властивостей.

Тепер під час виклику `set_prop_values` набору властивостей він повинен спочатку перетворити імена на індекси, використовуючи створене відображення.

Клієнт повинен оновлювати зіставлення індексу з іменами кожного разу, коли встановлюється новий сеанс з тим самим пристроєм.

Ім’я кінцевої точки (Bluetooth Low Energy + сервер GATT)	**URI (сервер HTTPS + mDNS)**	**опис**
esp_local_ctrl/версія	https:/ /<mdns-hostname>.local/esp_local_ctrl/version	Кінцева точка, яка використовується для отримання рядка версії
esp_local_ctrl/control	https:/ /<mdns-hostname>.local/esp_local_ctrl/control	Кінцева точка, яка використовується для

Ім’я кінцевої точки (Bluetooth Low Energy + сервер GATT)	URI (сервер HTTPS + mDNS)	опис
		надсилання або отримання керуючих повідомлень

Довідник API

Файл заголовка

components/esp_local_ctrl/include/esp_local_ctrl.h

Цей файл заголовка можна включити до:

#include *"esp_local_ctrl.h"*

- Цей файл заголовка є частиною API, що надається компонентом `esp_local_ctrl`. Щоб оголосити, що ваш компонент залежить від `esp_local_ctrl`, додайте наступне до свого CMakeLists.txt:

  REQUIRES esp_local_ctrl

  або

  PRIV_REQUIRES esp_local_ctrl

Функції

const esp_local_ctrl_transport_t * esp_local_ctrl_get_transport_ble (void)

Функція для отримання транспортного режиму BLE.

const esp_local_ctrl_transport_t * esp_local_ctrl_get_transport_httpd (void)

Функція для отримання транспортного режиму HTTPD.

esp_err_t esp_local_ctrl_start (const esp_local_ctrl_config_t * config)

Запустіть службу локального керування.

Параметри

config -- **[in]** Покажчик на структуру конфігурації

Повернення

- ESP_OK : успіх
- ESP_FAIL : Помилка

esp_err_t esp_local_ctrl_stop (void)

Зупинити службу місцевого контролю.

esp_err_t esp_local_ctrl_add_property (const esp_local_ctrl_prop_t * prop)

Додайте нову властивість.

Це додає нову властивість і виділяє для неї внутрішні ресурси. Загальна кількість властивостей, які можна додати, обмежена опцією конфігурації `max_properties`

Параметри

prop -- **[in]** Структура опису властивості

Повернення

- ESP_OK : успіх
- ESP_FAIL : Помилка

esp_err_t esp_local_ctrl_remove_property (const char * name)

Видалити властивість.

Це знаходить властивість за назвою та звільняє внутрішні ресурси, пов’язані з нею.

Параметри

name -- **[in]** Назва властивості, яку потрібно видалити

Повернення

- ESP_OK : успіх
- ESP_ERR_NOT_FOUND : Помилка

const esp_local_ctrl_prop_t * esp_local_ctrl_get_property(const char * ім'я)

Отримати структуру опису властивості за назвою.

Цей API можна використовувати для отримання структури контексту властивості, `esp_local_ctrl_prop_t` якщо відома її назва

Параметри

name -- [in] Назва властивості, яку потрібно знайти

Повернення

- Покажчик на властивість
- NULL, якщо не знайдено

esp_err_t esp_local_ctrl_set_handler (const char * ep_name , обробник protocomm_req_handler_t , void * user_ctx)

Зареєструйте обробник protocomm для спеціальної кінцевої точки.

Цей API може бути викликаний програмою для реєстрації обробника протоколу для кінцевої точки після запуску служби локального керування.

У випадку транспортування BLE імена та uuid усіх користувацьких кінцевих точок потрібно надати заздалегідь як частину `protocomm_ble_config_t` набору структури `esp_local_ctrl_config_t` та передати в `esp_local_ctrl_start()`.

Параметри

- **ep_name -- [in]** Назва кінцевої точки
- **обробник -- [in]** функція обробки кінцевої точки
- **user_ctx -- [in]** Дані користувача

Повернення

- ESP_OK : успіх
- ESP_FAIL : Помилка

Союзи

union esp_local_ctrl_transport_config_t

#include <esp_local_ctrl.h>

Конфігурація транспортного режиму (BLE / HTTPD).

Public члени

> **esp_local_ctrl_transport_config_ble_t * ble**

Це те саме, що `protocomm_ble_config_t`. Перегляньте `protocomm_ble.h` доступні параметри конфігурації.

> **esp_local_ctrl_transport_config_httpd_t * httpd**

Це те саме, що `httpd_ssl_config_t`. Перегляньте `esp_https_server.h` доступні параметри конфігурації.

Структури

struct esp_local_ctrl_prop

Структура даних опису властивості, яка має бути заповнена та передана функції `esp_local_ctrl_add_property()`.

Після додавання властивості її структура стає доступною для доступу лише для читання всередині `get_prop_values()` та `set_prop_values()` для обробників.

Public члени

> **char * ім'я**

Унікальна назва властивості

> **тип uint32_t**

Тип власності. Це може бути встановлено для визначених програмою enumів

size_t розмір

Розмір вартості майна, який:

- якщо нуль, властивість може мати значення змінного розміру
- якщо відмінне від нуля, властивість може мати лише значення фіксованого розміру, тому перевірки виконуються внутрішньо esp_local_ctrl під час встановлення значення такої властивості

прапори uint32_t

Для цієї властивості встановлено прапорці. Це може бути трохи поля. Прапорець може вказувати на поведінку властивості, наприклад, лише для читання / константа

void * ctx

Покажчик на деякі контекстні дані, що стосуються цієї властивості. Це буде доступно для використання всередині обробників `get_prop_values` і `set_prop_values` як частину цієї структури властивостей. Якщо встановлено, це діє протягом усього часу існування властивості, доки властивість не буде видалено або служба esp_local_ctrl не буде зупинено.

void (* ctx_free_fn) (void * ctx)

Функція, яка використовується esp_local_ctrl для внутрішнього звільнення контексту властивості під час виклику `esp_local_ctrl_remove_property()` або . `esp_local_ctrl_stop()`

struct esp_local_ctrl_prop_val

Структура даних значення властивості. Це передається обробникам `get_prop_values()` і `set_prop_values()` з метою отримання або встановлення поточного значення властивості.

Public члени

> **void * дані**

Покажчик на значення властивості зберігання пам’яті

> **size_t розмір**

Розмір

> **void (* free_fn) (void * data)**

Це може бути встановлено програмою в `get_prop_values()` обробнику, щоб вказати `esp_local_ctrl` викликати цю функцію за покажчиком даних вище, щоб звільнити її ресурси після надсилання відповіді `get_prop_values`.

struct esp_local_ctrl_handlers

Обробники для отримання та відповіді на локальні команди керування для отримання та встановлення властивостей.

Public члени

esp_err_t (* get_prop_values) (size_t props_count , const esp_local_ctrl_prop_t props [ ] , esp_local_ctrl_prop_val_t prop_values [ ] , void * usr_ctx)

Функція обробки, яка буде реалізована для отримання поточних значень властивостей.

Якщо будь-яка з властивостей має фіксовані розміри, `prop_values` потрібно встановити поле розміру відповідного елемента

Параметр props_count

[in] Загальна кількість елементів у масиві props

Реквізит параметрів

[in] Масив властивостей, поточні значення яких запитував клієнт

Параметр prop_values

[out] Масив порожніх значень властивостей, елементи якого потрібно заповнити поточними значеннями цих властивостей, указаних аргументом props

Параметр usr_ctx

[in] Це забезпечує значення `usr_ctx` поля `esp_local_ctrl_handlers_t` структури

Повернення

Повернення різних кодів помилок передасть клієнту відповідні помилки рівня протоколу:

- ESP_OK : успіх
- ESP_ERR_INVALID_ARG : Void аргумент
- ESP_ERR_INVALID_STATE : InvalidProto
- Усі інші коди помилок: InternalError

esp_err_t (* set_prop_values) (size_t props_count , const esp_local_ctrl_prop_t props [ ] , const esp_local_ctrl_prop_val_t prop_values [ ] , void * usr_ctx)

Функція обробки, яка буде реалізована для зміни значень властивостей.

Якщо будь-яка з властивостей має змінні розміри, поле розміру відповідного елемента `prop_values` має бути перевірено явно перед тим, як робити будь-які припущення щодо розміру.

Параметр props_count

[in] Загальна кількість елементів у масиві props

Реквізит параметрів

[in] Масив властивостей, значення яких запитує змінити клієнт

Параметр prop_values

[in] Масив значень властивостей, елементи якого потрібно використовувати для оновлення цих властивостей, указаних аргументом props

Параметр usr_ctx

[in] Це забезпечує значення `usr_ctx` поля `esp_local_ctrl_handlers_t` структури

Повернення

Повернення різних кодів помилок передасть клієнту відповідні помилки рівня протоколу:

- ESP_OK : успіх
- ESP_ERR_INVALID_ARG : Void аргумент
- ESP_ERR_INVALID_STATE : InvalidProto
- Усі інші коди помилок: InternalError

> **void * usr_ctx**

Покажчик контексту, який буде передано вищезгаданим функціям обробки після виклику. Це відрізняється від контексту рівня властивостей, оскільки він дійсний протягом усього життя служби `esp_local_ctrl` та звільняється, лише коли службу зупинено.

> **void (* usr_ctx_free_fn) (void * usr_ctx)**

Покажчик на функцію, яка буде внутрішньо викликана `usr_ctx` для звільнення ресурсів контексту під час `esp_local_ctrl_stop()` виклику.

struct esp_local_ctrl_proto_sec_cfg

Конфігурації безпеки Protocom

Public члени

версія esp_local_ctrl_proto_sec_t

Це встановлює версію безпеки протоколу, sec0/sec1 або користувальницьку. Якщо користувацька, користувач має вказати дескриптор за допомогою `proto_sec_custom_handle` нижче

void * custom_handle

Настроюваний дескриптор безпеки, якщо безпеку встановлено власноруч за допомогою `proto_sec` вище. Цей дескриптор має відповідати `protocomm_security_t` підпису

***const* порожнеча * поп**

Підтвердження володіння, яке використовується для місцевого контролю. Може бути NULL.

***const* void * sec_params**

Покажчик на параметри безпеки (NULL, якщо не потрібен). Це не потрібно для безпеки протоколу 0. Цей покажчик має містити структуру типу esp_local_ctrl_security1_params_t для захисту протоколу 1 та esp_local_ctrl_security2_params_t для захисту протоколу 2 відповідно. Може бути NULL.

struct esp_local_ctrl_config

Структура конфігурації для переходу `esp_local_ctrl_start()`

Public члени

***const* esp_local_ctrl_transport_t * транспорт**

Транспортний рівень, на якому буде надаватися послуга

esp_local_ctrl_transport_config_t transport_config

Транспортний рівень, на якому буде надаватися послуга

esp_local_ctrl_proto_sec_cfg_t proto_sec

Безпечна версія та POP

обробники esp_local_ctrl_handlers_t

Зареєструйте обробники для відповіді на запити отримання/встановлення властивостей

size_t max_properties

Це обмежує кількість властивостей, доступних одночасно

Макроси

ESP_LOCAL_CTRL_TRANSPORT_BLE

ESP_LOCAL_CTRL_TRANSPORT_HTTPD

Визначення типів

typedef struct esp_local_ctrl_prop esp_local_ctrl_prop_t

Структура даних опису властивості, яка має бути заповнена та передана функції `esp_local_ctrl_add_property()`.

Після додавання властивості її структура стає доступною для доступу лише для читання всередині `get_prop_values()` та `set_prop_values()` для обробників.

typedef struct esp_local_ctrl_prop_val esp_local_ctrl_prop_val_t

Структура даних значення властивості. Це передається обробникам `get_prop_values()` і `set_prop_values()` з метою отримання або встановлення поточного значення властивості.

typedef struct esp_local_ctrl_handlers esp_local_ctrl_handlers_t

Обробники для отримання та відповіді на локальні команди керування для отримання та встановлення властивостей.

typedef struct esp_local_ctrl_transport esp_local_ctrl_transport_t

Транспортний режим (BLE / HTTPD), у якому надаватиметься послуга.

Це пряма декларація приватної структури, реалізована внутрішньо `esp_local_ctrl`.

typedef struct protocomm_ble_config esp_local_ctrl_transport_config_ble_t

Конфігурація для транспортного режиму BLE.

Це форвардна декларація для `protocomm_ble_config_t`. Щоб використовувати це, програма повинна встановити CONFIG_BT_ENABLED і включити `protocomm_ble.h`.

typedef struct httpd_config esp_local_ctrl_transport_config_httpd_t

Конфігурація для транспортного режиму HTTPD.

Це пряма декларація для `httpd_ssl_config_t` (для HTTPS) або `httpd_config_t` (для HTTP)

typedef enum esp_local_ctrl_proto_sec esp_local_ctrl_proto_sec_t

Типи безпеки для esp_local_control.

typedef protocomm_security1_params_t esp_local_ctrl_security1_params_t

typedef protocomm_security2_params_t esp_local_ctrl_security2_params_t

typedef struct esp_local_ctrl_proto_sec_cfg esp_local_ctrl_proto_sec_cfg_t

Конфігурації безпеки Protocom

typedef struct esp_local_ctrl_config esp_local_ctrl_config_t

Структура конфігурації для переходу `esp_local_ctrl_start()`

Перерахування

enum esp_local_ctrl_proto_sec

Типи безпеки для esp_local_control.

Значення:

> ***enumerator*** **PROTOCOM_SEC0**
>
> ***enumerator*** **PROTOCOM_SEC1**
>
> ***enumerator*** **PROTOCOM_SEC2**
>
> ***enumerator*** **PROTOCOM_SEC_CUSTOM**

3. БЕЗДРОТОВА МЕРЕЖА "EASY NET EVERYWHERE"

Призначення

Бездротова мережа "Easy Net Everywhere" призначена для застосування в територіально-розподілених системах, які потребують наявності бездротового зв'язку з об'єктами управління з гарантованою доставкою команд і даних зі швидкістю 250 Kbps у радіусі до 10 км (залежно від умов навколишньої місцевості та складу обладнання).

Область застосування

Мережа "Easy Net Everywhere" може бути використана для вирішення завдань у проектах різного ступеня складності, на Приклад:

- для управління як окремими об'єктами, так і роєм об'єктів, на Приклад, дронами, роботами, роботизованими платформами та іншими механізмами;
- для використання в якості фреймворку для побудови різних систем і реалізації проектів рівня "Internet Of Things";
- для отримання і пересилання в мережу координат GPS, створення GPS трекерів і маячків тощо;
- для використання в системах зовнішньої та внутрішньої охорони об'єктів і приміщень, управління камерами відеоспостереження, у системах тривожної сигналізації, у метеостанціях і системах моніторингу датчиків показників навколишнього середовища;
- для віддаленого керування станом об'єктів у комунальних службах, на Приклад, водопровідними засувками, кранами, клапанами тощо;
- для збирання та передавання інформації з віддалених автономних датчиків, датчиків пристроїв автоматики, медичних приладів тощо;
- для систем керування дозаторами, зерносушарками, теплицями тощо;
- для систем автоматичного керування мікрокліматом, поливом, освітленням тощо;

- для систем керування міськими світлофорами з метою підвищення комфорту пересування автомобільного транспорту, зниження рівня ДТП тощо;
- для систем управління класу "Smart Home», ввімкнення/вимкнення побутових приладів і електроприводів, для управління потужністю в навантаженні тощо;
- для швидкої реалізації територіально-розподілених DIY-проектів (Arduino тощо) ;
- для створення текстових месенджерів і чатів.

Область застосування мережі обмежується тільки її технічними характеристиками.

Технічні характеристики:

Частотний діапазон: ISM.

Радіоканали: 0-125 (від 2400 до 2525 МГц).

Полоса пропускання радіоканалу - 1МГц.

Трансивери: NRF24L01+, NRF24L01+PA+LNA, E01-ML01DP5, E01-2G4M27D.

Відстань приймання/передавання даних у межах прямої видимості на висоті 1 м на швидкості 250 Kbps:

- трансивер NRF24L01 - 100 м;
- трансивер NRF24L01+PA+LNA, E01-ML01DP5 - 1км;
- трансивер E01-2G4M27D - 2 км.

Швидкість передачі даних: 250 Kbps, 1 Mbps і 2 Mbps.

Максимальна довжина пакета: 256 байт.

Час передачі пакета 256 байт на швидкості 250 Kbps - 50 мс (5 Кб/с).

Interfaceи користувача:

- UART 9600 - 921600 baud;
- Bluetooth: BLE.

Interfaceи контролера мережі:

- I^2C;
- SPI;
- GPS UART 115200 baud.

Кількість функціональних виводів контролера – 14.

Функції виводів контролера (призначаються користувачем):

- ADC - 14;
- DAC - 2;
- PWM - 11;
- Servo - 11;
- GPIO: вихід - 11, вхід - 14, input only - 3;
- GPIO переривання – 14.

Архітектури мережі:

Архітектура "багато-до-багатьох":

- кількість вузлів – 254.

Кластерна архітектура:

- кластерів - 14;
- кількість вузлів у кластері - 15;
- кількість роутерів - 30.

Напруга живлення: 3,3 В, 5 В.

Струм споживання вузла мережі:

- в активному режимі: 40 ма;
- у пасивному режимі: 25 мка.

Діапазон робочих температур: -40ºС...+85ºС.

Особливості мережі

Мережа "Easy Net Everywhere" має кластерну архітектуру з маршрутизацією пакетів і можливістю масштабування.

Вузлами мережі використовуються трансивери невеликої потужності для забезпечення обміну даними тільки там, де це необхідно, не займаючи ефір там, де це небажано.

Таке рішення забезпечує створення оптимальної топології мережі, підвищення порога виявлення роботи трансиверів і зменшення ймовірності виникнення колізій. Розв'язання колізій у мережі реалізується пропрієтарним протоколом.

Вузли мережі можуть одночасно працювати як у режимі точка-точка на невеликій дистанції в межах прямої видимості до 2 км, так і в режимі ретрансляції пакетів, що дає можливість встановлювати зв'язок між об'єктами на відстані до 10 км, а через мережу StarLink, мережу GSM і GSM-Internet - у встановлених межах.

Передавання даних здійснюється каналами, які обираються випадковим чином із числа доступних. Теоретично доступно 125 каналів з шириною смуги пропускання 1 МГц. Практично - залежить від конкретних умов.

Усі вузли мережі рівнозначні і можуть мати повний або обмежений доступ до інших вузлів мережі залежно від використаної архітектури.

Кожен вузол мережі може передати команди і дані будь-якому іншому вузлу.

Адміністратор мережі може встановити необхідні обмеження для кожного вузла. Адміністрування мережі можливе з будь-якого вузла, наділеного повноваженнями.

Кожна наступна транзакція не використовує повторно канал попередньої транзакції, що ускладнює виявлення вузла сканерами ефіру. Тривалість однієї транзакції на одному каналі не перевищує 25 мс.

Девіація каналів передавання даних може бути встановлена адміністратором у межах +/- 2...60 від частоти основного каналу.

Адміністратор може встановити обмежений список робочих каналів, при цьому кожному адресату може бути поставлено у відповідність один або кілька фіксованих/змінних каналів.

Безпека. Кожен вузол мережі має адресу мережі, адресу вузла, пароль і логін мережі, а також пароль доступу до вузла через Bluetooth.

Шифрування даних: пропрієтарний алгоритм маскування даних.

Передача даних у мережі може ініціюватися:

- користувачем;
- вузлом мережі;
- подією;
- перевищенням/зниженням значення параметра відносно заданого порога;
- після закінчення часу таймера одноразово або із заданою періодичністю.

Для передавання даних вузлу-одержувачу використовується маршрутизація відправника. Для кожного вузла-одержувача можна вказати свій маршрут проходження пакетів і записати його в профіль адресата.

Така маршрутизація детермінована і для стаціонарних і малорухомих об'єктів оптимальніша, ніж стохастична маршрутизація в mesh-мережі.

Взаємодія користувача з будь-яким вузлом мережі здійснюється за допомогою послідовного interface UART на швидкості до 921600 baud або за допомогою interface Bluetooth (BLE).

Конфігурація та встановлення параметрів вузлів мережі здійснюється за допомогою AT- команд через бездротовий interface Bluetooth (BLE) планшета або смартфона та дротовий interface UART.

Усі налаштування зберігаються в пам'яті вузла.

Порівняльні характеристики малопотужних бездротових мереж представлені в табл.1

Обмеження:

- обмеження 1 зумовлене частотою роботи трансивера - 2.4 GHz. Зв'язок можливий у межах прямої видимості;
- обмеження 2: за надто великої дистанції між роутерами зв'язок на даній ділянці може перерватися і мережа розпадеться на кілька самостійних сегментів.

Для виключення такої ситуації кількість роутерів на одиницю площі/обсягу ареалу функціонування мережі має бути більшою за мінімально необхідну кількість. У цьому разі існує більше варіантів вибору альтернативних маршрутів проходження пакетів.

Таблиця 1 - Порівняльні характеристики малопотужних бездротових мереж

Параметр	Z-Wave	Zigbee	LoRaWAN	Easy Net Everywhere
Потужність передавача	0 dBm/1 мВт	0 dBm/1 мВт	до 30 dBm/1000 мВт	до 27 dBm/500 мВт
Дальність зв'язку між вузлами мережі	до 100 м.	до 100 м.	до 10 км (модулі класу С) з додатковим обладнанням	до 10 км (модулі класу С) ● без додаткового обладнання
Швидкість передачі	до 100 Кбіт/с *	до 250 Кбіт/с *	до 19,2 Кбіт/с *	● 250 Кбіт/с ; 1 та 2 Mbit/c
Затримка передачі	Tx: 1% Rx: 99%	< 5 мс	від 1 мс до десятків секунд	● < 1 мс
Кількість каналів	16	16	127 (діапазон 2.4 GHz)	● 127
Мобільність	низька	низька	низька	● висока
Складність установки	середня	середня	висока	● низька
Орієнтація	пристрої і датчики	пристрої і датчики	пристрої і датчики	● людина, пристрої і датчики
Обмін текстовими повідомленнями	немає	немає	немає	● є, 256 байт/транзакція
Комунікаційні інтерфейси користувача	немає	немає	немає	● SPI, I2C, UART 921600 baud
Бездротові інтерфейси користувача	немає	немає	немає	● Wi-Fi, Bluetooth/BLE, GSM
Навігація	немає	немає	немає	● GPS/GLONASS/Galileo/Beidou
Управління дронами	непридатна	непридатна	непридатна	● придатна
Ліцензія	не потрібна	потрібна	не потрібна	не потрібна
Вартість	висока	середня	висока	● низька

Примітки: * - залежить від відстані між пристроями; ● - перевага

2. ОПИС КОМПОНЕНТІВ І ВУЗЛІВ МЕРЕЖІ

Трансивери

У всіх вузлах мережі передбачено використання трансиверів різної потужності залежно від розв'язуваних завдань. Трансивер вузла вибирається користувачем. Трансивер або впаюється в плату вузла (рекомендується), або вставляється в роз'єм (для тестування).

Вузли мережі працюють із такими трансиверами:

- трансивери NRF24L01+, NRF24L01+PA+LNA (рис.1);
- трансивери E01-ML01DP5, E01-2G4M27D (рис.2).

Рисунок 1 - Трансивери NRF24L01+ та NRF24L01+PA+LNA

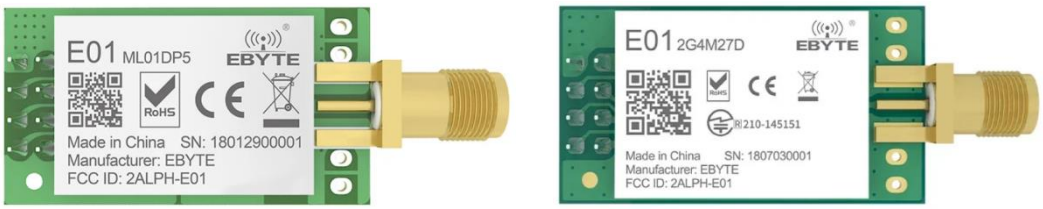


Рисунок 2 – Трансивери E01-ML01DP5 та E01-2G4M27D

Антени

З трансиверами можуть використовуватися різні антени, як штатні, так і спеціальні, з конектором SMA. Від вибору антени залежить відстань, на якій трансивери можуть забезпечити надійний зв'язок.

Штатна антена має коефіцієнт посилення 3 dBi і забезпечує впевнений зв'язок на відстані 1-2 км на відкритій місцевості на висоті 1,5 м з трансивером E01-2G4M27D (рис. 3).

Рисунок 3 - Штатна антена трансивера

Антена Yagi 10 dBi забезпечує впевнений зв'язок на відстані до 5 км на відкритій місцевості на висоті 1,5 м з трансивером E01-2G4M27D (рис. 4).

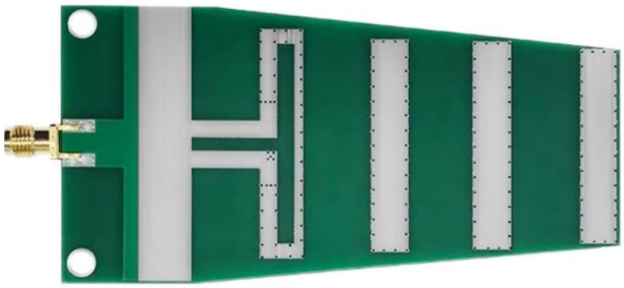


Рисунок 4 - Антена Yagi 10 dBi

Антена Yagi 25 dBi забезпечує впевнений зв'язок на відстані до 30 км на відкритій місцевості на висоті 1,5 м з трансивером E01-2G4M27D (рис. 5).

Рисунок 5 - Антена Yagi 25 dBi

Під час вибору антени слід враховувати діаграму спрямованості, яка вказується в технічних характеристиках конкретних антен.

Керування трансиверами здійснюється мікроконтролером ESP32 фірми Espressif, у керуючій програмі якого реалізовано стек системних, мережевих і користувацьких протоколів. Сукупність контролера і трансивера утворює вузол мережі.

Вибір оптимальних каналів для роботи мережі

Діапазон частот, у якому працює мережа, зумовлює певні вимоги до антен, що використовуються і вибору оптимального каналу (групи каналів) для роботи вузлів мережі.

Антени повинна мати мінімальне значення коефіцієнта стоячої хвилі КСВ (*SWR standing wave ratio*) в обраному діапазоні частот. Цей параметр визначає ККД антени і безпосередньо впливає на дальність зв'язку. Хвильовий опір антен для трансиверів становить 50 Ом.

Для визначення КСВ антен використовуються спеціальні прилади: КСВ-метри. На Приклад, у польових умовах для вимірювання необхідних параметрів антен можна використовувати LiteVNA (Potrable Vector Network Analizer).

Результати вимірювання КСВ штатної антени трансивера (рис. 3) за допомогою приладу LiteVNA представлені на рис. 6.

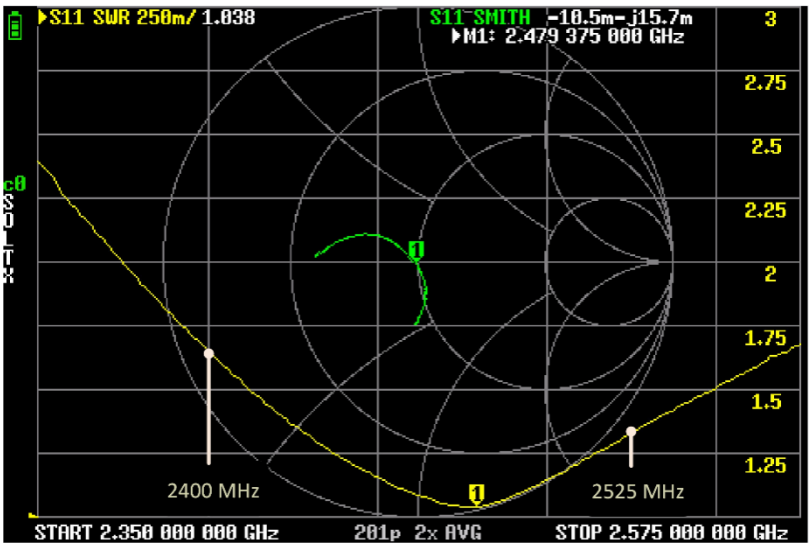


Рисунок 6 - Результати вимірювання КСВ штатної антени трансивера

З графіка випливає, що антена не є широкосмуговою. Мінімальний КСВ антени 1.038 відповідає частоті 2479 MHz або каналу з номером 79.

Прийнятний КСВ відповідає частотам 2450 MHz - 2500 MHz. Це означає, що основний канал для роботи мережі бажано вибирати в діапазоні від 50 до 100.

На рисунку 7 представлено скріншот екрана аналізатора TSA ULTRA (Tiny Spectrum Analyzer) з результатами сканування частот у діапазоні роботи мережі 2400 MHz - 2525 MHz (канали 0 - 125) під час виконання тесту для двох вузлів мережі.

На скріншоті відображено частоти, на яких здійснюється обмін даними між бездротовими пристроями:

- у діапазоні частот 2400 MHz - 2400 MHz працюють пристрої Wi-Fi;
- у діапазоні частот 2400 MHz - 2400 MHz працюють вузли мережі, займаючи канали 58 – 68;
- частота 2485 MHz відповідає основному каналу з номером 85.

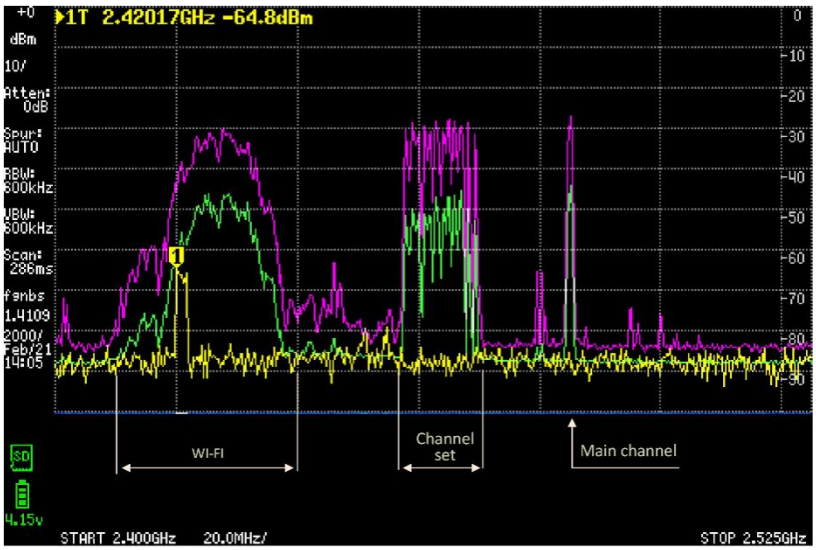


Рисунок 7 - Результат сканування діапазона 2400 MHz - 2525 MHz

Таким чином, знаючи КСВ антени, можна вибрати найбільш оптимальний діапазон частот (каналів) для роботи мережі.

У разі відсутності приладу для вимірювання КСВ антени, рекомендується обирати канали в середині робочого діапазону ISM і надалі коригувати їх на підставі статистичної інформації про помилки передавання даних за всіма обраними каналами (AT- команда AT+ERROR.log).

3. ПРИЗНАЧЕННЯ ТА СТРУКТУРА ВУЗЛІВ МЕРЕЖІ

Вузли мережі

Для побудови та організації роботи мережі використовуються мережеві функціональні вузли:

- "Net Master" (NM);
- "Controller" (C);
- "Light Node" (L);
- "Cluster Admin" (CA);
- "Net Router" (NR);
- "Repeater" (R).

Net Master

Вузол "Net Master" є головним об'єктом мережі та призначений для адміністрування мережі, надсилання команд і даних у мережу та отримання даних із мережі. Взаємодіє з мережею Internet за допомогою Wi-Fi роутера, за допомогою роутера Starlink і GSM-модема, що має підключення до UART.

Структурну схему вузла "Net Master" наведено на рис. 8.

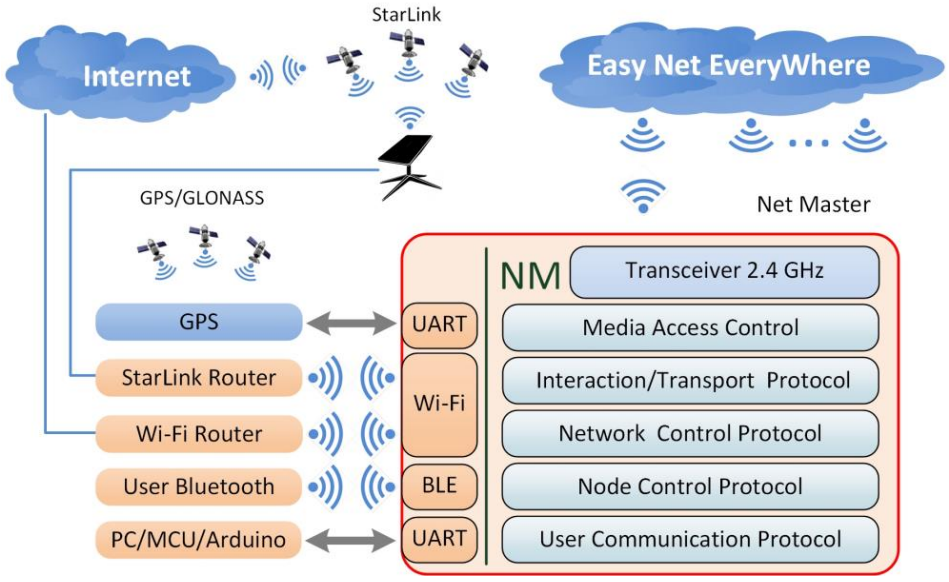


Рисунок 8 - Структурна схема вузла "Net Master"

Controller

Вузол "Controller" призначений:

- для приймання з мережі та передавання в UART команд і даних користувача, включно з текстовими повідомленнями довжиною до 256 байт;
- для виконання команд користувача;
- для збирання та передавання інформації.

Команди користувача являють собою числові значення, яким поставлено у відповідність виконання однієї дії на стороні контролера або послідовності дій, на Приклад:

- увімкнути/вимкнути будь-який пристрій;
- збільшити/зменшити потужність у навантаженні;
- повернути вал одного або декількох сервоприводів на заданий кут з заданою швидкістю;
- встановити на виводах DAC рівень постійної напруги;
- прочитати поточне значення напруги на виводах ADC і передати їх користувачеві;
- встановити тайм-аут таймера для періодичного передавання користувачеві станів виводів і значень ADC тощо.

Вузол "Controller" має можливість підключення модуля GPS. Це дає змогу в реальному масштабі часу визначати координати кожного вузла мережі в просторі, передавати координати вузлу "Net Master" з подальшим відображенням їх на Google-карті (візуалізація топології мережі на смартфоні/планшеті або комп'ютері).

Це дуже корисна можливість для завдання оптимального маршруту проходження пакетів, відображення координат об'єктів, встановлення маячків, GPS-GSM трекерів, навігації дронів, роботизованих платформ тощо.

Структурна схема вузла "Controller" представлена на рис. 9.

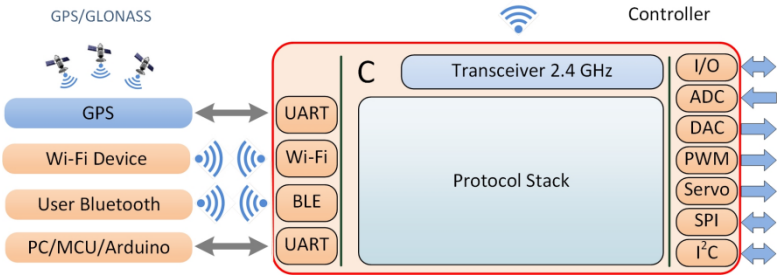


Рисунок 9 - Структурна схема вузла "Controller"

На структурній схемі вузла " Controller" з лівого боку представлені комунікаційні interfaceи UART і Bluetooth для взаємодії користувача з контролером мережі.

З правого боку представлені послідовні системні interfaceи SPI, I2C, і модулі контролера для управління обладнанням і отримання зовнішніх сигналів.

Light Node

Вузол "Light Node" призначений для двостороннього обміну командами і даними між interfaceом UART і мережею зі швидкістю до 921600 baud. Вузол не містить функціональних виводів. Для взаємодії з користувачем вузол "Light Node" має interfaceи UART, Wi-Fi і Bluetooth (BLE).

Структурну схему вузла "Light Node" представлено на рис.10.

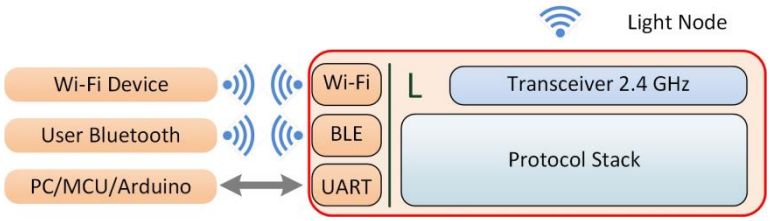


Рисунок 10 - Структурна схема вузла "Light Node"

Cluster Admin

Вузол "Cluster Admin" призначений для двостороннього обміну командами і даними користувача між interfaceом UART і мережею зі швидкістю до 921600 baud. Вузол здійснює адміністрування всіх вузлів кластера, виконує системні та сервісні функції для забезпечення роботи кластера. Для взаємодії з користувачем вузол "Cluster Admin" має interfaceи UART і Bluetooth (BLE).

Структурну схему вузла "Cluster Admin" наведено на рис. 11.

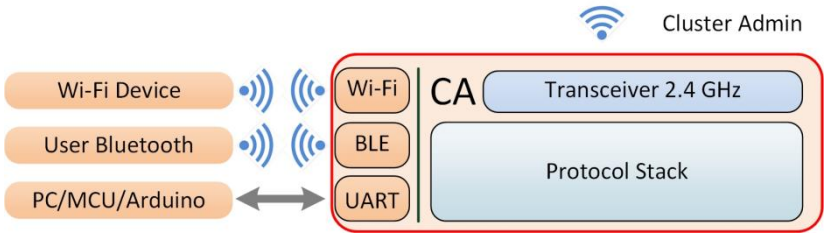


Рисунок 11 - Структурна схема вузла "Cluster Admin"

Net Router

Вузол "Net Router" призначений для маршрутизації пакетів у мережі відповідно до встановленого маршруту. Звільняє інші вузли мережі від участі в процесі маршрутизації та ретрансляції пакетів.

Реалізація вузла "Net Router" має два варіанти: одиночний і здвоєний. Одиночний роутер спочатку приймає дейтаграму, а потім передає. Оскільки режим роботи роутера симплексний, то під час передачі даних приймання даних не здійснюється. Загальний час прийому-передачі дейтаграми становить 50 мс.

У здвоєному роутері /“Fast Router”/ один роутер приймає дейтаграму, потім по interface UART пересилає її другому роутеру, який передає дейтаграму в мережу. Час прийому-передачі дейтаграми становить 27 мс.

Для взаємодії з користувачем вузол "Net Router" має interfaceи UART і Bluetooth (BLE).

Структурну схему одиночного вузла "Net Router" представлено на рис. 12.

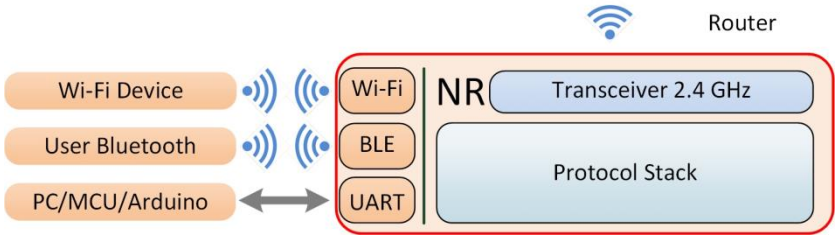


Рисунок 12 - Структурна схема одиночного вузла "Net Router"

Здвоєний роутер має дві мережеві адреси та по два interfaceи UART і Bluetooth (BLE).

Структурна схема здвоєного вузла "Net Router" подана на рис. 13.

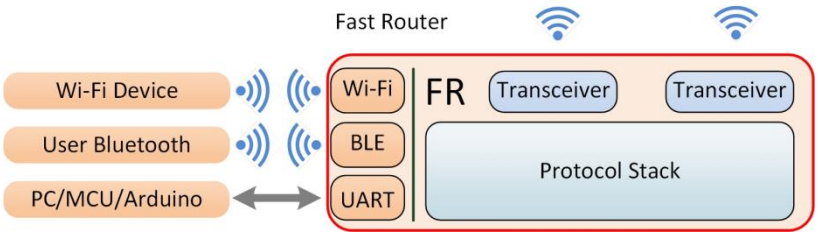


Рисунок 13 - Структурна схема здвоєного вузла "Net Router"

Repeater

Вузол "Repeater" (ретранслятор) призначений для локального розширення зони покриття мережі. Він є повільним пристроєм зі швидкістю передачі 2400 baud. Діапазон частот роботи репітера - 144 і 433 MHz.

Вузол "Repeater" доцільно використовувати в умовах поганого проходження сигналу 2,4 GHz або у відсутності можливості використання вузла "Net Router".

Вузол "Repeater" приймає і передає дейтаграму цілком по interface UART під управлінням спеціального протоколу.

Структурну схему вузла "Repeater" наведено на рис. 14.

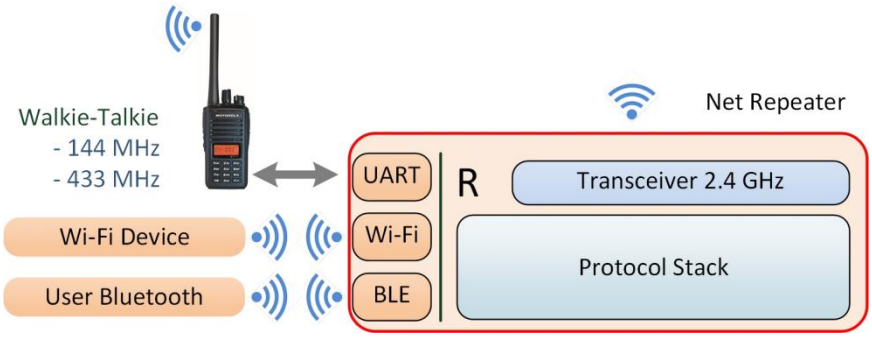


Рисунок 14 - Структурна схема вузла "Repeater"

4. АРХІТЕКТУРА, МАСШТАБУВАННЯ, АДРЕСАЦІЯ ТА РОБОТА МЕРЕЖІ

Архітектура мережі "Easy Net Everywhere" дає змогу користувачеві легко будувати необхідну топологію, оптимально відповідну для вирішення конкретного завдання.

У загальному вигляді архітектура мережі "Easy Net Everywhere" представлена на рис. 15.

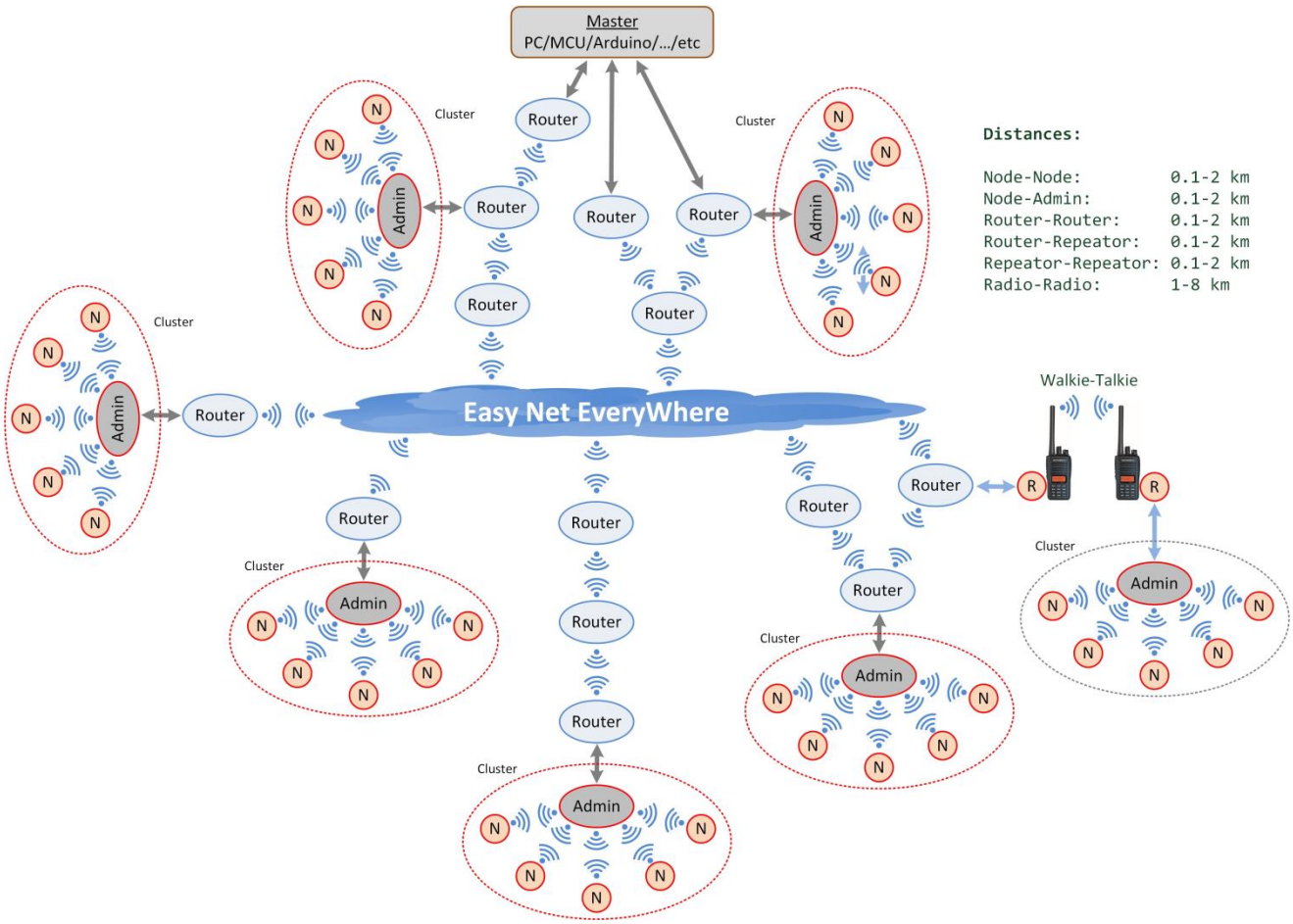


Рисунок 15 - Архітектура мережі "Easy Net Everywhere"

Масштабування мережі

Архітектура мережі "Easy Net Everywhere" є легко масштабованою.

На Приклад, для побудови системи "Розумний дім", що містить невелику кількість вузлів на невеликій площі за умов низького трафіку, застосування роутерів і репітерів недоцільне.

Однак, у територіально розподіленій системі з безліччю вузлів і відстанями між вузлами до 10 км, роутери та репітери є необхідністю.

Масштабування мережі реалізується відповідно до таких моделей:

- Модель "Simple Net";
- Модель "Light Cluster Net";
- Модель "Medium Cluster Net";
- Модель "Union Cluster Net".

Модель мережі "Simple Net" встановлює режим роботи простої мережі з аморфною архітектурою та аморфною топологією без використання роутерів та інших вузлів. У цьому режимі користувач і кожен вузол має прямий доступ до всіх інших вузлів мережі. Усі вузли за замовчуванням рівнозначні. За замовчуванням, Майстром мережі може стати будь-який вузол, до якого в цей момент підключився користувач. Тому мережа може мати кілька Майстрів мережі, якщо це не заборонено адміністратором.

Така модель рекомендується для створення дуже малих локальних мереж з невеликим трафіком.

Архітектура мережі на основі моделі "Simple Net" представлена на рис. 16.

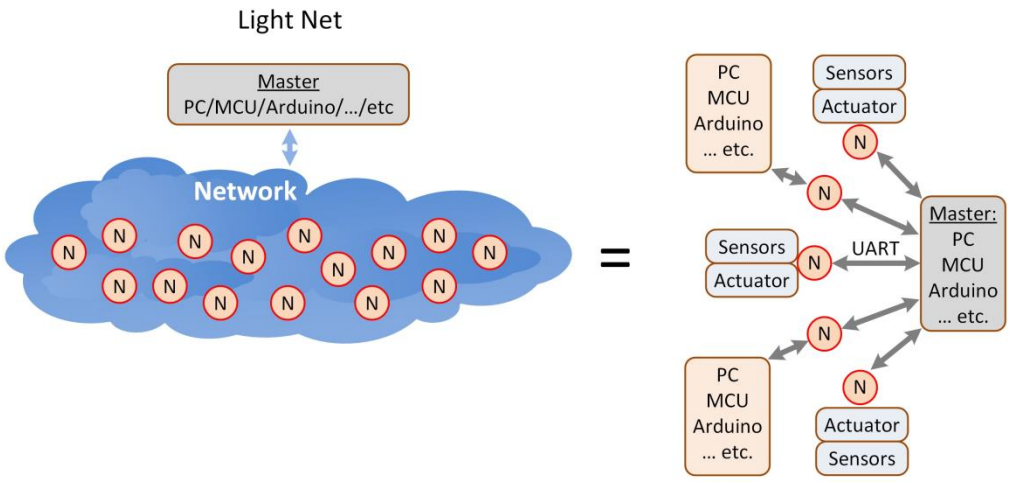


Рисунок 16 - Архітектура мережі на основі моделі "Simple Net"

Фізична реалізація мережі "Simple Net" представлена на рис. 17.

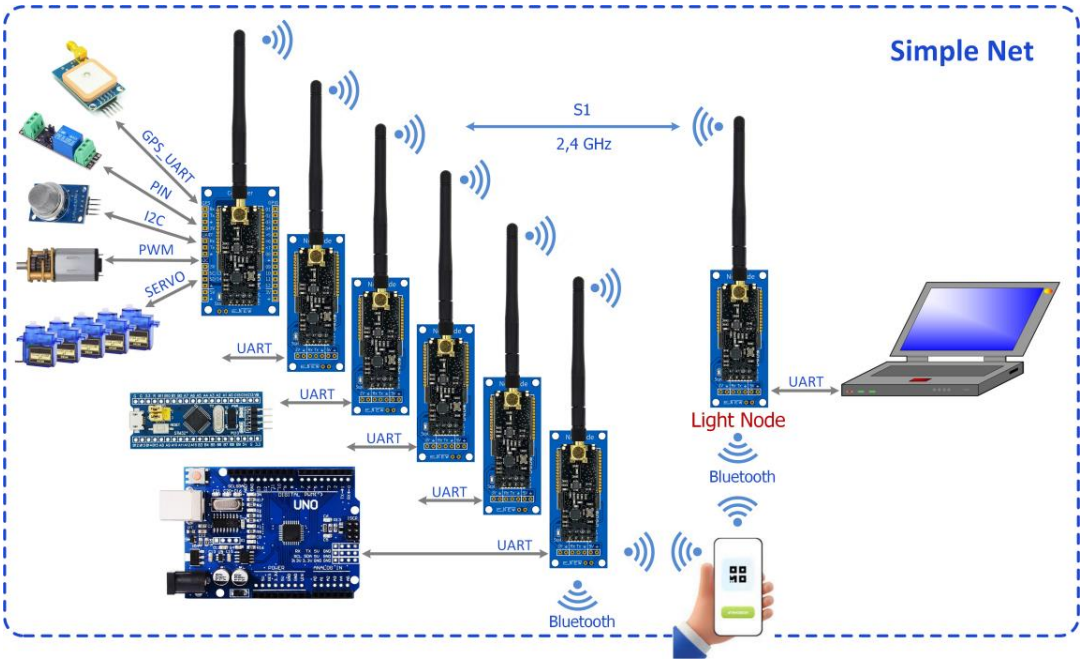


Рисунок 17 - Фізична реалізація мережі "Simple Net"

Значення дистанції S1 залежить від використовуваного трансивера та антени. На Приклад, у разі використання штатної антени і трансивера NRF24L01+PA+LNA дистанція S1 становить 1 км, а в разі використання трансивера E01-2G4M27D - 2 км.

Модель мережі "Light Cluster Net" встановлює режим роботи мережі з одним або кількома кластерами без використання роутерів. У цьому режимі доступ до вузлів кластера здійснюється через вузол "Cluster Admin", який інкапсулює вузли кластера і звільняє користувача від роботи з управління вузлами "Light Node".

Вузли "Cluster Admin" мають прямий зв'язок із вузлом "Net Master" за interfaceом UART зі швидкістю до 921600 baud.

Така модель рекомендується для створення невеликих локальних мереж з невеликим трафіком.

Архітектура мережі на основі моделі "Light Cluster Net" представлена на рис. 18.

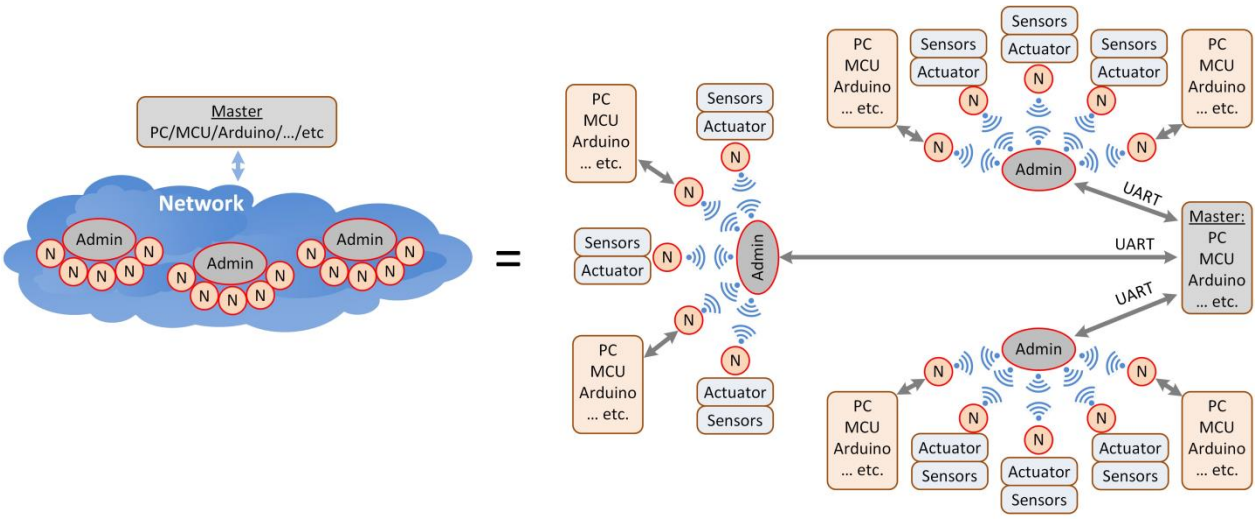


Рисунок 18 - Архітектура мережі на основі моделі "Light Cluster Net"

Фізична реалізація кластера мережі "Light Cluster Net" представлена на рис. 19.

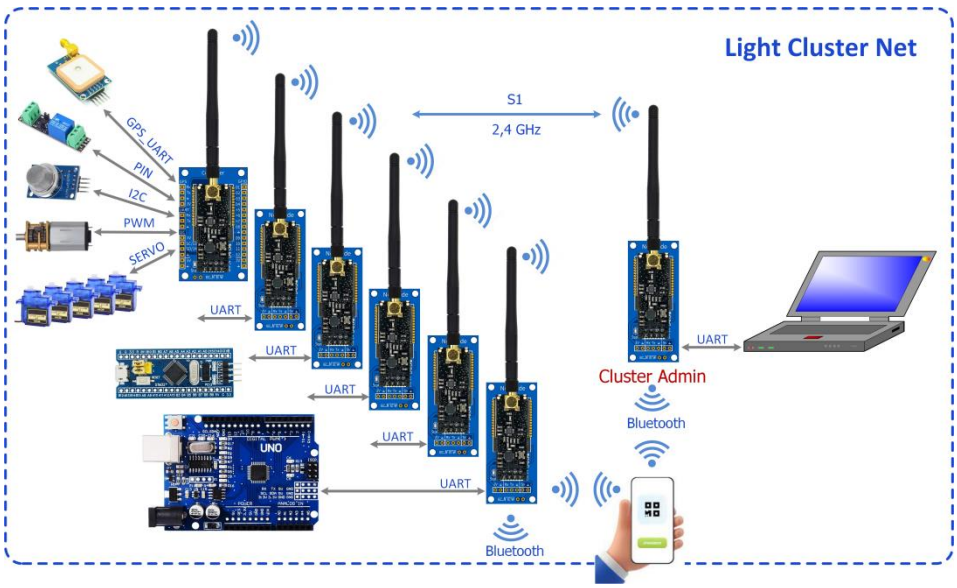


Рисунок 19 - Фізична реалізація кластера мережі "Light Cluster Net"

Модель "Medium Cluster Net" встановлює режим роботи мережі з декількома кластерами з використанням роутерів і репітерів та відносно високим трафіком в умовах наявності перешкод для проходження мережевих пакетів.

Доступ до вузлів мережі також здійснюється через вузол "Cluster Admin".

Така модель рекомендується для створення територіально-розподілених мереж з відносно високим трафіком. Фізична реалізація кластера мережі "Medium Cluster Net" представлена на рис. 20.

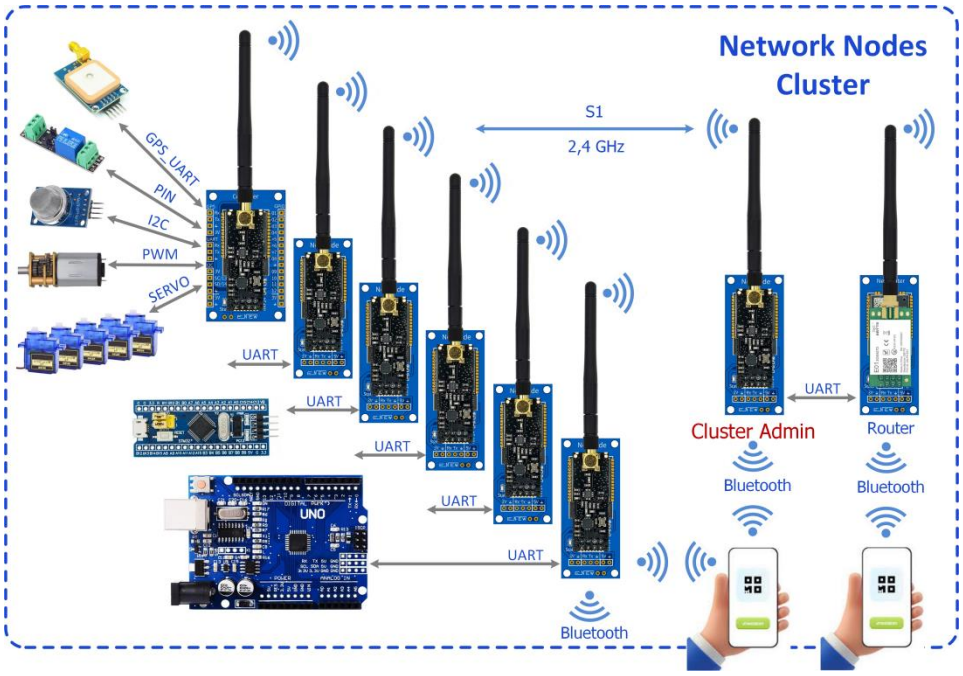


Рисунок 20 - Фізична реалізація кластера мережі "Medium Cluster Net"

Архітектура мережі на основі моделі "Medium Cluster Net" представлена на рис. 21.

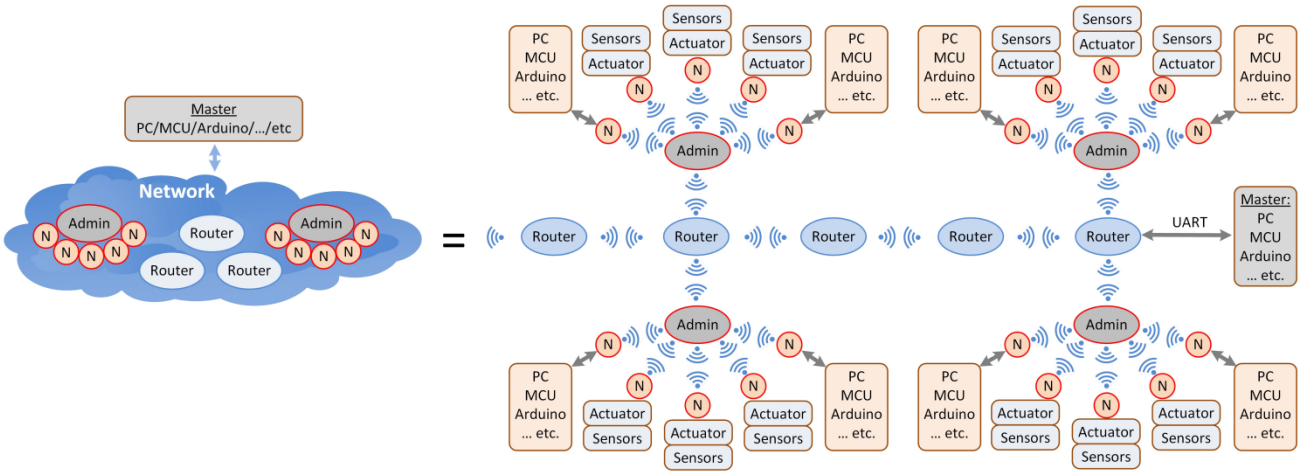


Рисунок 21 - Архітектура мережі на основі моделі "Medium Cluster Net"

Використання роутерів дає змогу користувачеві призначати та змінювати маршрути проходження пакетів, оптимально розподілити мережевий трафік і обійти об'єкти, що перешкоджають обміну даними між об'єктами мережі.

Фізична реалізація мережі "Medium Cluster Net" представлена на рис. 22.

Під час побудови мережі слід враховувати, що в разі спільного використання різних трансиверів дистанція стійкого зв'язку S2 дорівнюватиме найменшому значенню S1,

де: S1 - дистанція для трансивера NRF24L01+PA+LNA (1 км);

S2 - дистанція для трансивера E01-2G4M27D (2 км).

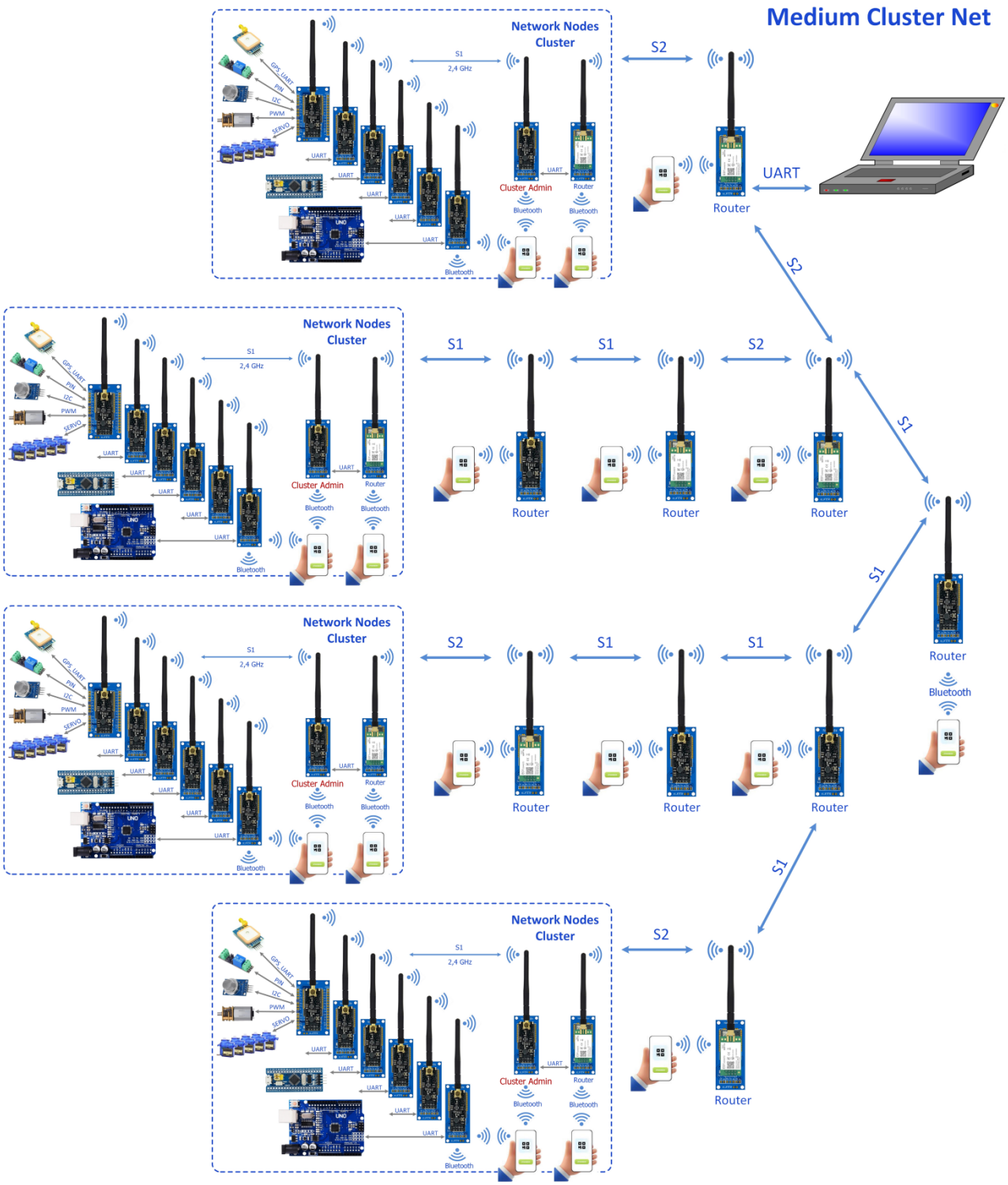


Рисунок 22 - Фізична реалізація мережі "Medium Cluster Net"

Модель "Union Cluster Net" об'єднує декілька локальних мереж різної архітектури в єдину систему. Використання мереж Internet та GSM дає можливість побудови глобальної мережі Easy Net Everywhere з урахуванням певних обмежень.

Архітектура мережі на основі моделі "Union Cluster Net" представлена на рис. 23.

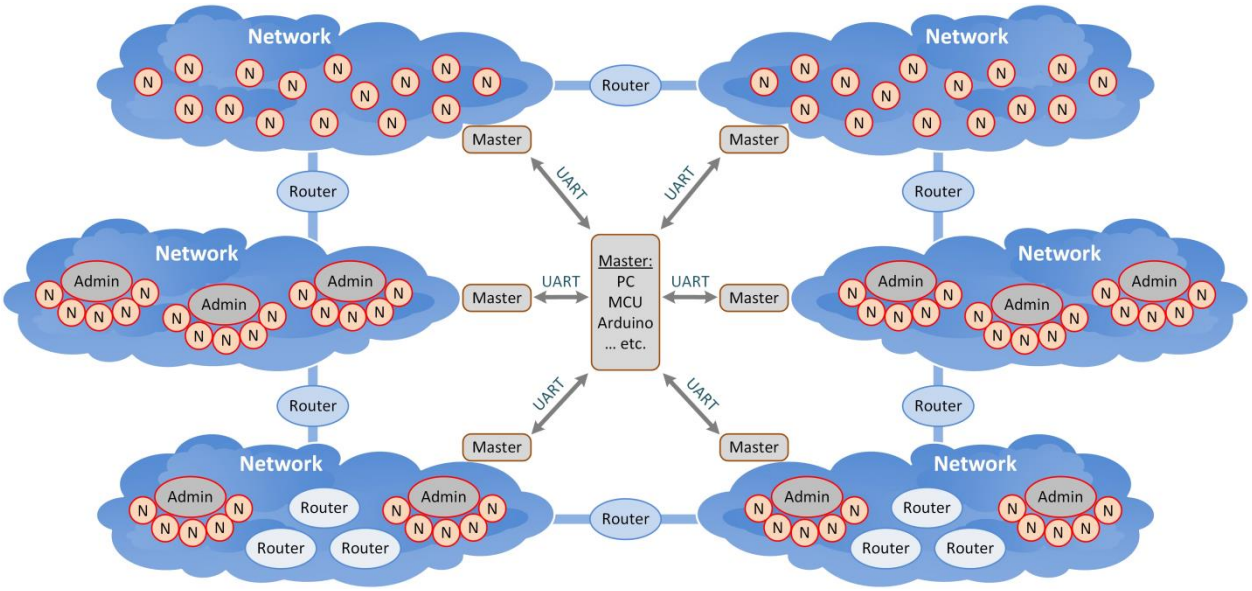


Рисунок 23 - Архітектура мережі на основі моделі "Union Cluster Net"

Адресація вузлів у мережі

Кожен вузол мережі має свою унікальну адресу. Повна адреса вузла складається з трьох компонентів: адреси мережі, адреси кластера і номера вузла в кластері.

Довжина адреси мережі становить 1 байт і адресує 254 мережі.

Довжина адреси кластера становить 4 біти і адресує 14 кластерів.

Довжина номера вузла становить 4 біти і адресує 15 вузлів.

Формат мережевої адреси наведено на рис. 24.

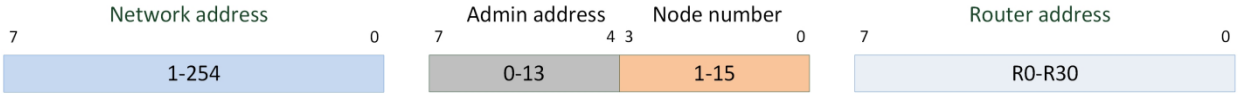


Рисунок 24 - Формат мережевої адреси

Поділ адресного простору мережі

Адресний простір мережі розділено на сегменти за функціональною ознакою пристроїв:

- вузли кластера мають номери: 1-15/0x01-0x0F;
- адміністратори кластера займають діапазон адрес: 0.0-13.0/0x00-0xD0;
- роутери та репітери займають діапазон адрес: R0-R30/0xE0-0xFE/.

Поділ адресного простору мережі на сегменти представлено на рис. 25.

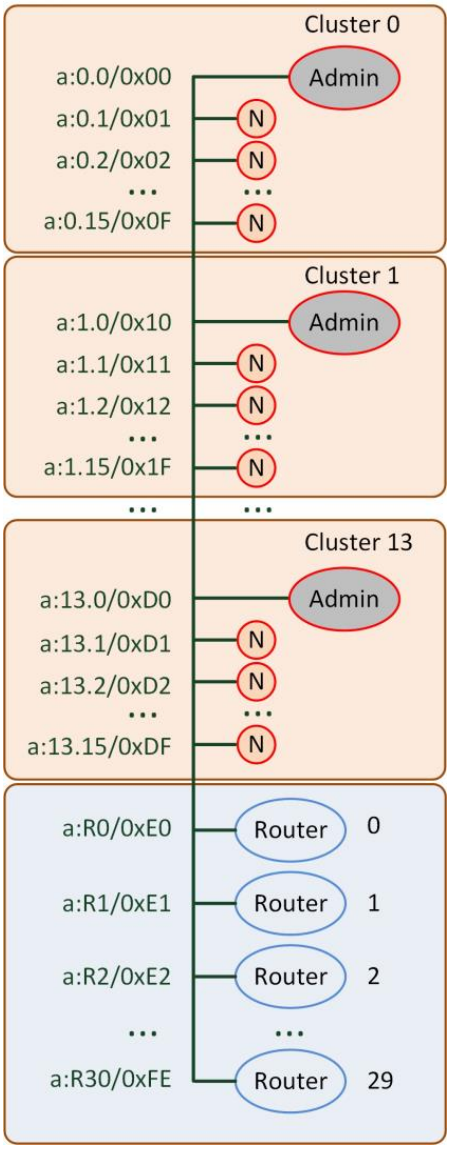


Рисунок 25 - Поділ адресного простору мережі на сегменти

Адреси вузлів мережі наведені в таблиці 2.

Таблиця 2 - Адреси вузлів мережі

Cluster Admin address		Node address		Full Node address		Router address			
alias:	hex:	alias:	hex:	alias:	hex:	alias:	hex:	alias:	hex:
0	0x00	1	0x01	0.1-1.15	0x01-0x0F	R0	0xE0	R16	0xF0
1	0x10	2	0x02	1.1-1.15	0x11-0x1F	R1	0xE1	R17	0xF1
2	0x20	3	0x03	2.1-2.15	0x21-0x2F	R2	0xE2	R18	0xF2
3	0x30	4	0x04	3.1-3.15	0x31-0x3F	R3	0xE3	R19	0xF3
4	0x40	5	0x05	4.1-4.15	0x41-0x4F	R4	0xE4	R20	0xF4
5	0x50	6	0x06	5.1-5.15	0x51-0x5F	R5	0xE5	R21	0xF5
6	0x60	7	0x07	6.1-6.15	0x61-0x6F	R6	0xE6	R22	0xF6
7	0x70	8	0x08	7.1-7.15	0x71-0x7F	R7	0xE7	R23	0xF7
8	0x80	9	0x09	8.1-8.15	0x81-0x8F	R8	0xE8	R24	0xF8
9	0x90	10	0x0A	9.1-9.15	0x91-0x9F	R9	0xE9	R25	0xF9
10	0xA0	11	0x0B	10.1-10.15	0xA1-0xAF	R10	0xEA	R26	0xFA
11	0xB0	12	0x0C	11.1-11.15	0xB1-0xBF	R11	0xEB	R27	0xFB
12	0xC0	13	0x0D	12.1-12.15	0xC1-0xCF	R12	0xEC	R28	0xFC
13	0xD0	14	0x0E	13.1-13.15	0xD1-0xDF	R13	0xED	R29	0xFD
Address range		15	0x0F	Address range		R14	0xEE	R30	0xFE
0.0-13.0	0x00-0xD0			0.1-13.15	0x01-0xDF	R15	0xEF		

Приклад адресації вузлів мережі

Якщо вузли належать до однієї мережі, то адреса мережі не вказується.

Приклад адресації вузлів мережі наведено на рис. 26.

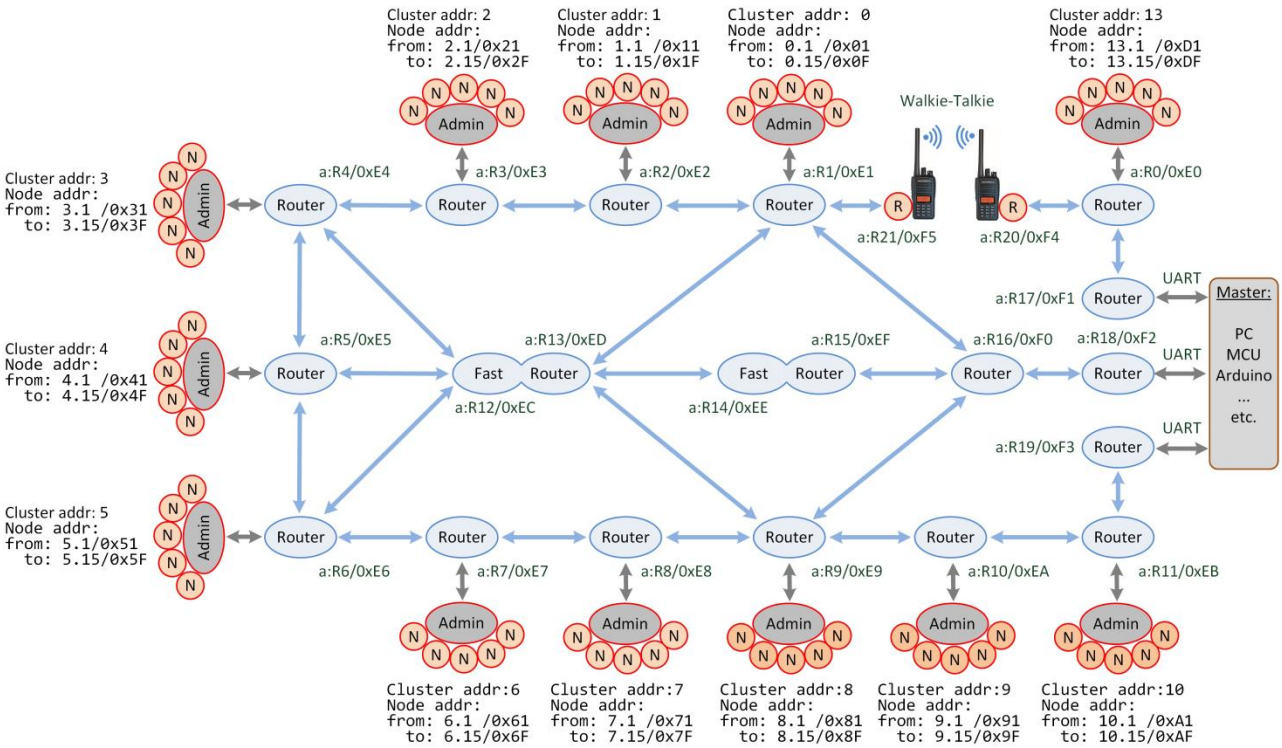


Рисунок 26 - Приклад адресації вузлів мережі

5. ПІДГОТОВКА ВУЗЛІВ МЕРЕЖІ ДО РОБОТИ

Для підготовки вузла мережі до роботи необхідно під'єднати його до джерела живлення 5 В або 3,3 В, як зазначено на рисунках 15 і 16.

Для взаємодії користувача з вузлом через комп'ютер, через мікроконтролер або через плату Arduino, необхідно роз'єм UART вузла під'єднати до роз'єму UART відповідного пристрою і встановити швидкість 115200 baud.

Для підключення вузла до комп'ютера використовується будь-який конвертор UART-USB. До комп'ютера через USB-Hub можна підключити кілька вузлів мережі.

Після цього можна увімкнути живлення вузла і виконати відповідні налаштування вузла за допомогою будь-якої термінальної програми.

Налаштування вузла можна виконати через планшет/смартфон за допомогою програми "Serial Bluetooth Terminal" (автор Kai Morich).

Для цього достатньо з'єднатися з вузлом по interface Bluetooth (BLE) і виконати відповідні налаштування.

Цю процедуру необхідно виконати для кожного вузла мережі.

Перше ввімкнення вузла

Якщо всі підключення виконано правильно, то під час увімкнення живлення у вікні терміналу з'явиться таке повідомлення:

```
> Node 123 ready to start. Wait...
**********************
*  Easy Net EveryWhere
*  -------------------
*  Network.mode:   SIMPLE_NET
*  Radio.bitrate:  250Kb
*  UART.baud:      115200
*  Bluetooth:      ON 123 NET_NODE
**********************
>> Controller 123 started
```

Це означає, що вузол працездатний і готовий до роботи. Після цього необхідно виконати налаштування параметрів вузла.

Для налаштування вузла необхідно скористатися АТ-командами, які можуть бути надіслані вузлу з вікна терміналу або з планшета/смартфона.

АТ-команди виконують функції interface пристрою з користувачем, що дає змогу програмі керування пристроєм інкапсулювати виклики системних функцій.

Програма "Serial Bluetooth Terminal". Базові відомості

Адміністрування мережі, обмін інформацією в мережі та налаштування параметрів вузлів мережі здійснюється не тільки через порт UART комп'ютера/мікроконтролера, а й через планшет/смартфон за допомогою програми "Serial Bluetooth Terminal" (автор Kai Morich).

Користувач може використовувати будь-яку іншу аналогічну програму.

Програма завантажується з Internet і встановлюється на смартфон/планшет за допомогою програми Google Play.

Програма "Serial Bluetooth Terminal" дає змогу виконувати завдання:

1. Керування доступом до модуля Bluetooth (BLE) вузла мережі.
2. Встановлення режимів і параметрів вузла мережі за допомогою АТ-команд.
3. Приймання та відображення даних від вузла мережі.
4. Передавання даних вузлу мережі.
5. Адміністрування мережі.

Базовий набір функціональних кнопок

Програма "Serial Bluetooth Terminal" використовується для керування кількома різними пристроями за допомогою функціональних кнопок. Користувач може привласнити кожній кнопці ім'я і поставити у відповідність імені певну команду, яка відсилається керованому пристрою під час натискання кнопки. Кожен пристрій виконує свої функції зі своїми параметрами. Тому для керування кожним пристроєм необхідно мати окремий набір кнопок із командами керування, що не завжди зручно.

З метою спрощення interface користувача, відповідно до парадигми: "Багато пристроїв - один interface", функціональні кнопки interface називаються "F1", "F2" і т.д., а команди кнопок починаються з символу '~' (тільда).

Таким чином формується проксі-меню, яке в кожному конкретному пристрої має свою реалізацію, що дає змогу без зміни interface користувача керувати функціями різних пристроїв.

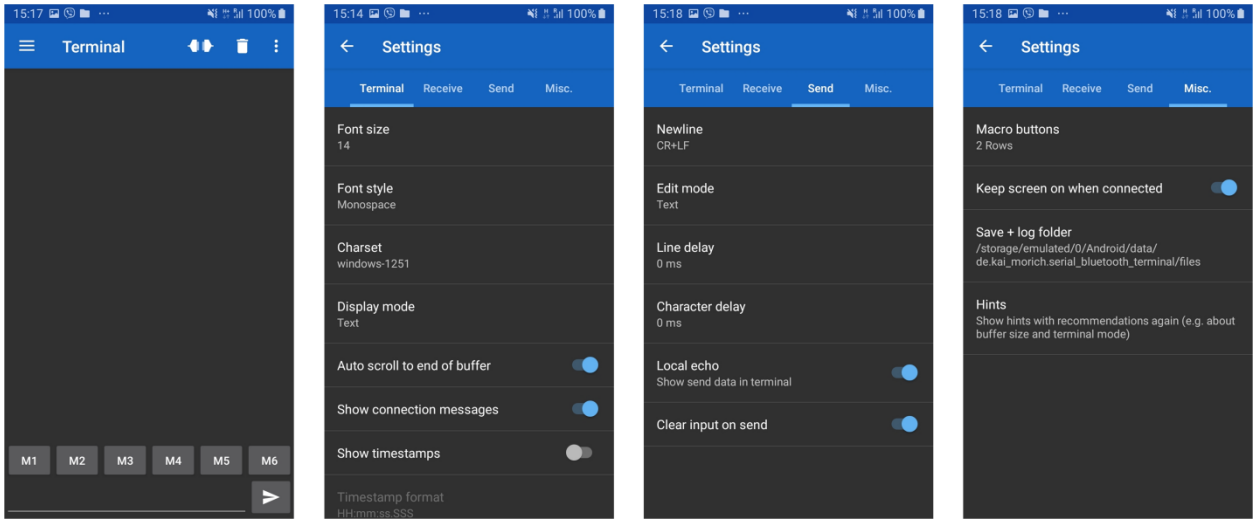
Базовий набір функціональних кнопок містить кнопки керування доступом до модуля Bluetooth (BLE) пристрою.

Розширений набір функціональних кнопок

Якщо базового набору функціональних кнопок недостатньо, то користувач створює додаткові кнопки, присвоює їм імена і ставить їм у відповідність виклики AT-команд.

Налаштування програми "Serial Bluetooth Terminal"

Після першого ввімкнення програми з'явиться вікно з набором кнопок, як показано на рис. 29.a. Користувачеві необхідно налаштувати interface взаємодії програми з вузлами мережі. Для цього у вкладках "Terminal", "Send" і "Misc." встановити параметри програми, як показано на рис. 29.b, c, d.



a) b) c) d)

Рисунок 29 - Установка параметрів програми

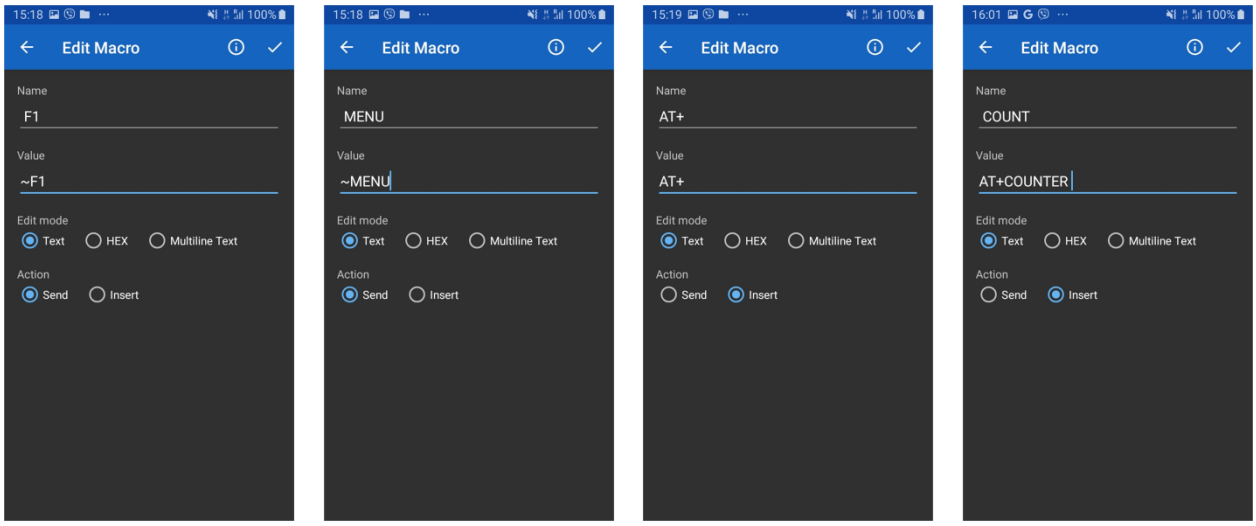
Потім необхідно присвоїти кнопкам імена і поставити їм у відповідність команди керування, як показано на рис. 30.a, b, c, d.

Порядок призначення імені кнопки і команди керування для неї:

1. Натискати на кнопку до появи вікна, як показано на рис. 30.a, b, c, d.

2. У верхньому рядку встановити ім'я кнопки, а в нижньому встановити команду керування.

Якщо команда не містить параметрів, то радіокнопку "Action" слід установити в положення "Send" (рис. 30.a, b), а якщо є параметри, то в положення "Insert" (рис. 30.c, d).



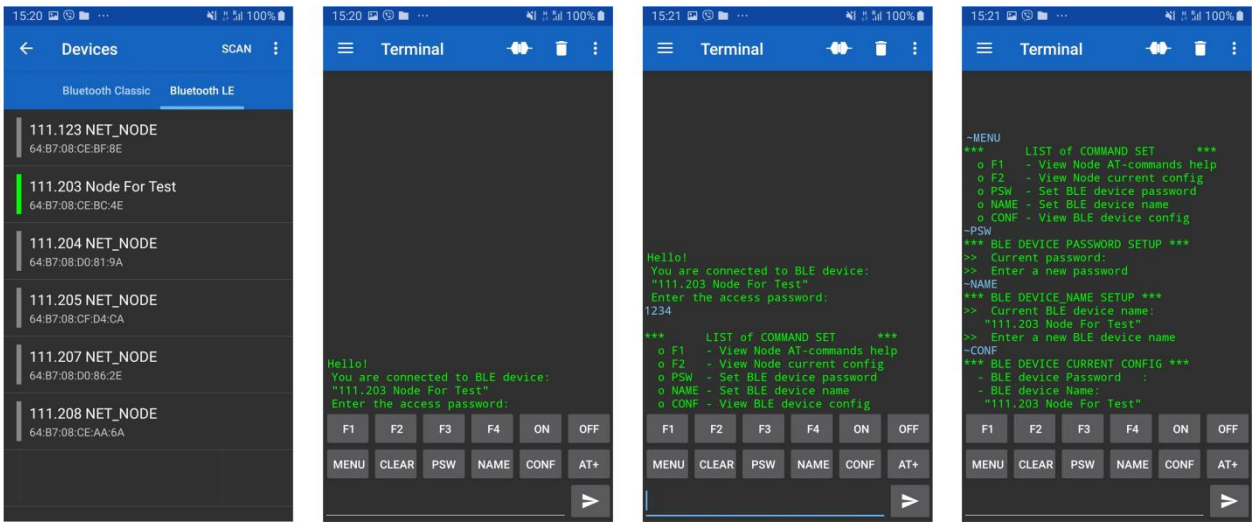
a) b) c) d)

Рисунок 30 - Присвоєння кнопкам імен і команд

Після присвоєння кнопці імені та команди необхідно натиснути галочку у верхньому правому куті екрана програми. Присвоїти імена та команди всім кнопкам, як показано на рис. 31.b.

Під час натискання на вкладку "Scan" програма просканує ефір і знайде пристрої Bluetooth (BLE) та відобразить на екрані у вигляді списку, як показано на рис. 31.a.

Для підключення до вузла мережі потрібно натиснути відповідний рядок у списку пристроїв.



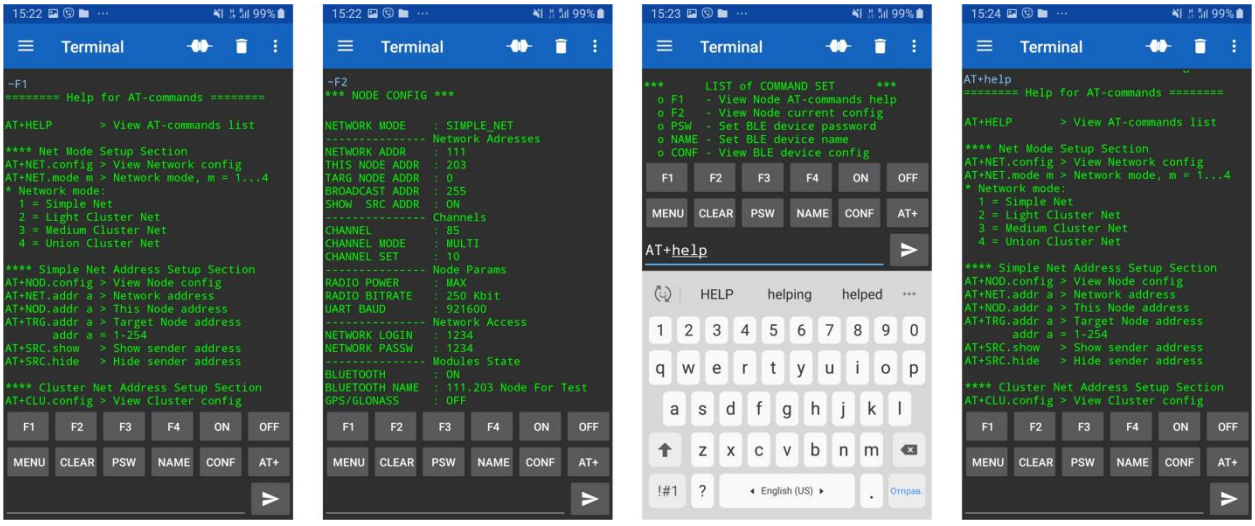
a) b) c) d)

Рисунок 31 - Приклад адресації вузлів мережі

У вікні термінала з'явиться повідомлення про під'єднання до вузла мережі та запит пароля доступу, як показано на рис. 31.b. Пароль за замовчуванням: "1234". Після введення пароля у вікні термінала з'явиться список кнопок і виконуваних команд, як показано на рис. 31.c.

На рис. 31.d показано виведення у вікно термінала меню командних кнопок.

На рис. 32.a і b показано виведення у вікно термінала результату натискання кнопок "F1" і "F2".



a) b) c) d)

Рисунок 32 - Виконання команд під час натискання кнопок

AT-команди мають префікс "AT+", який для зручності введення присвоєно відповідній кнопці. При натисканні кнопки "AT+" у рядку введення з'явиться префікс "AT+".

Для формування команди необхідно ввести частину AT-команди, що залишилася (на Приклад: "AT+HELP"), як показано на рис. 32.c і натиснути кнопку " > ".

Результат виконання AT-команди "AT+HELP" представлено на рис. 32.d. Аналогічним чином працюють усі інші командні кнопки.

Описані вище налаштування командних клавіш мають рекомендаційний характер.

Користувач на свій розсуд може встановлювати імена і команди для наявних командних кнопок, а також створювати додаткові кнопки для побудови зручного interface керування вузлами мережі.

6. ТЕСТУВАННЯ МЕРЕЖІ. ПЕРЕДАЧА КОМАНД І ДАНИХ

Для перевірки правильності роботи командних кнопок і працездатності вузлів мережі необхідно під'єднати вузли відповідно до рис. 33. На комп'ютері має бути встановлена будь-яка термінальна програма.

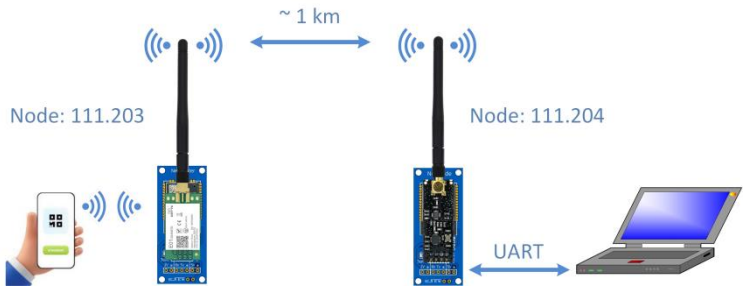


Рисунок 33 - Підключення вузлів мережі для тестування

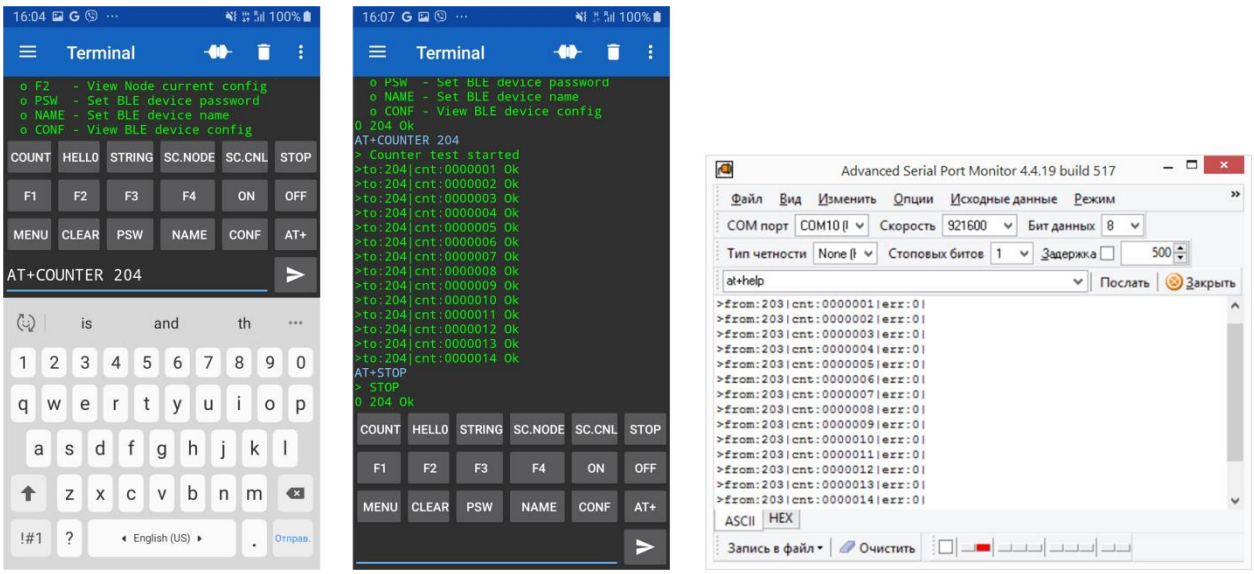
Тест якості зв'язку між двома вузлами запускається кнопкою "COUNT", яка передає вузлу AT- команду "AT+COUNTER target[,period]". Слід врахувати, що AT-команда має обов'язковий параметр, який необхідно ввести, і необов'язковий. Його можна не вказувати.

Обов'язковим параметром є адреса вузла-одержувача, до якого буде звернення.

Приклад введення показано на рис. 34.a.

Після правильного введення команди та натискання кнопки ">" запуститься тест інкрементного лічильника, значення якого передаватиметься вузлу-одержувачу. Процес виконання тесту відображатиметься у вікні терміналу на смартфоні, як показано на рис.34.b і у вікні терміналу на комп'ютері (рис.34.c).

Кількість помилок слугує критерієм для визначення якості зв'язку на обраному каналі.



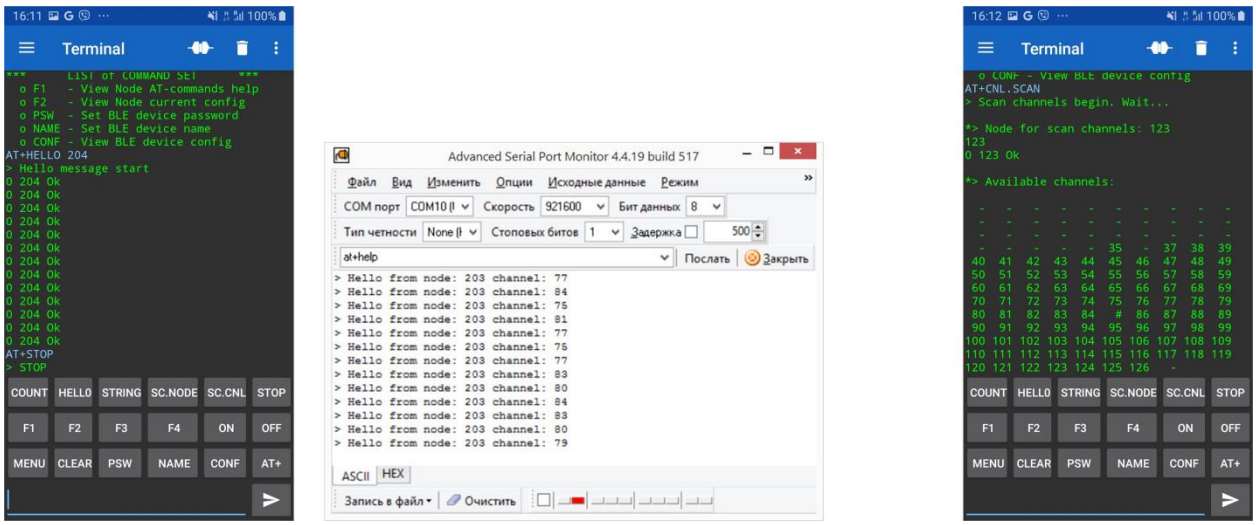
a) b) c)

Рисунок 34 - Запуск і виконання тестової команди "AT+COUNTER target[,period]"

Для тестування пропускної здатності каналу використовується тест "HELLO". Тест запускається на кількох вузлах, які надсилають повідомлення "Hello from node" одному вузлу.

Після введення команди "AT+HELLO target[,period]" і натискання кнопки ">" запуститься тест. Процес виконання тесту відображатиметься у вікні терміналу на смартфоні, як показано на рис. 35.a і у вікні терміналу на комп'ютері (рис. 35.b). Кількість колізій, що виникають, слугує критерієм визначення пропускної здатності каналу.

На рис. 35.c відображено виведення у вікно термінала смартфона результату виконання команди сканування каналів "AT+CNL.scan". З результату сканування випливає, що зайнятими каналами є канали 0-34, а канали 37-126 можна використовувати. Символ " # " у списку каналів позначає головний канал мережі.



a) b) c)

Рисунок 35 - Виконання тестової команди "AT+HELLO target[,period]"

На рисунку 36 представлено виведення у вікно терміналу планшета повідомлень тестової команди "AT+HELLO target[,period]" у процесі виконання. У тесті беруть участь 6 вузлів.

Вузол 203 - одержувач, а вузли 204, 205, 206, 207, 208 - відправники.

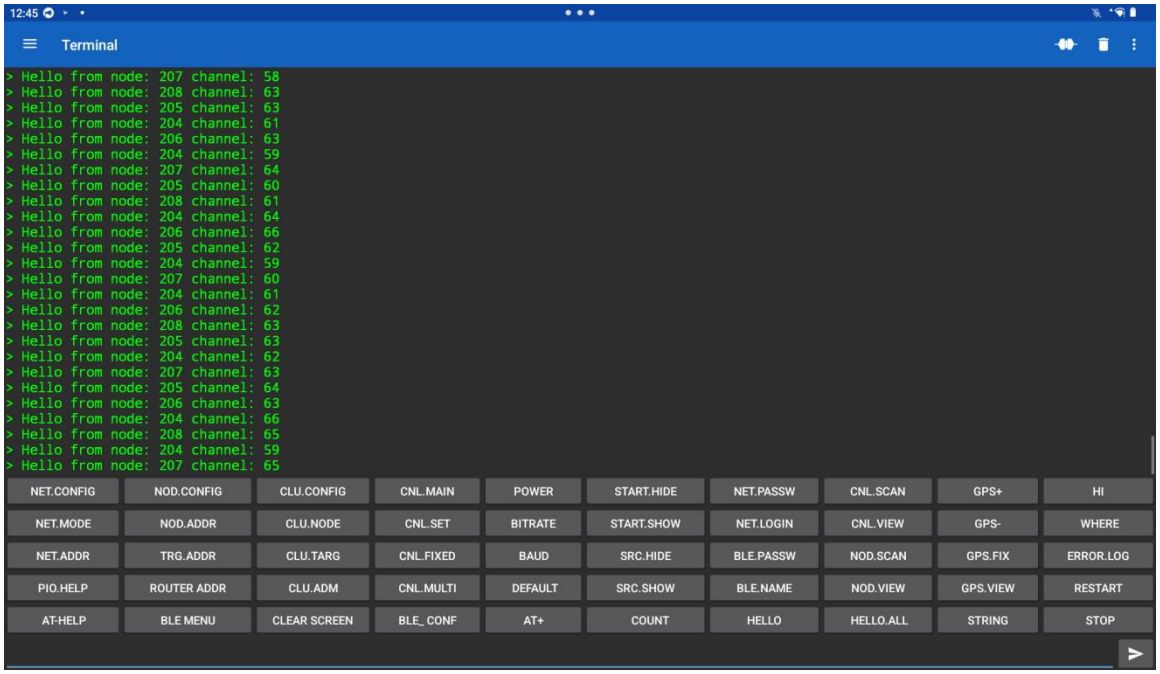


Рисунок 36 - Виконання тестової команди "AT+HELLO target[,period]" на 6 вузлах

На рисунку 37 представлено виведення повідомлень того самого тесту у вікна терміналів на комп'ютері через порти UART.

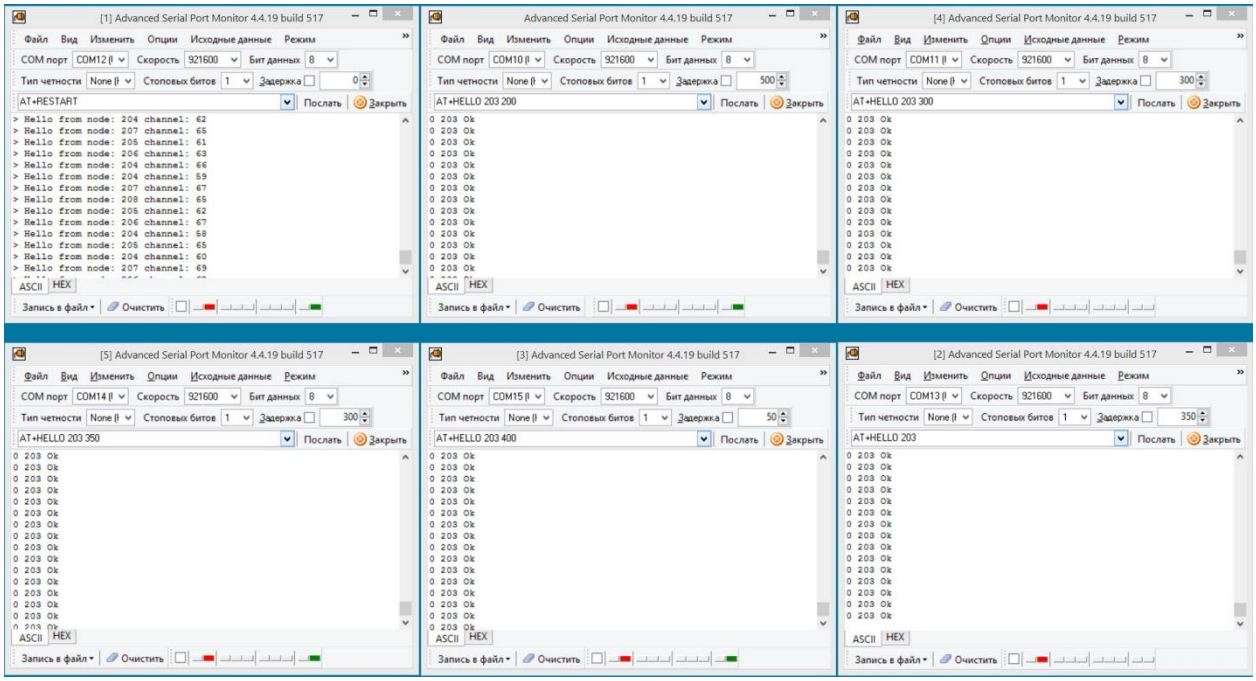


Рисунок 37 - Виконання тестової команди "AT+HELLO target[,period]" на 6 вузлах

Рядок "0 203 Ok" у вікні термінала (рис. 35) позначає:

0 - код квитанції успішної доставки пакета;

203 - адреса вузла-одержувача;

Ok - символьний еквівалент коду квитанції.

Рекомендація:

Перед початком роботи в мережі рекомендується встановити такі основні параметри:

Режим роботи мережі: команда "AT+NET.mode m";

Для всіх режимів мережі встановити:

Адреса мережі: команда "AT+NET.addr a";

Номер каналу мережі: команда "AT+CNL.main n";

Швидкість порту UART: команда "AT+BAUD bd";

Якщо обрано режим роботи мережі "Simple Net", то встановити:

Адреса вузла: команда "AT+NOD.addr a";

Якщо обрано режим роботи мережі "Light Cluster Net", додатково встановити:

Номер адміна кластера: команда "AT+CLU.adm n";

Номер вузла кластера: команда "AT+CLU.node n".

Якщо обрано режим роботи мережі "Medium Cluster Net", додатково встановити:

Адреси роутерів: команда "AT+ROUT.num n".

Адреси репітерів можна встановлювати в будь-якому режимі роботи мережі.

Подивитися поточні параметри вузла можна за допомогою команди: "AT+NOD.config"

Підключення обладнання для роботи в мережі

Підключення обладнання та організація процесу обміну даними в мережі є простим завданням.

Усі пристрої підключаються до вузлів мережі через послідовний interface UART відповідно до рис. 38 и 39.

Комп'ютери та контролери можна підключати в будь-якій комбінації. Приклад під'єднання комп'ютерів до вузлів мережі наведено на рис. 36.

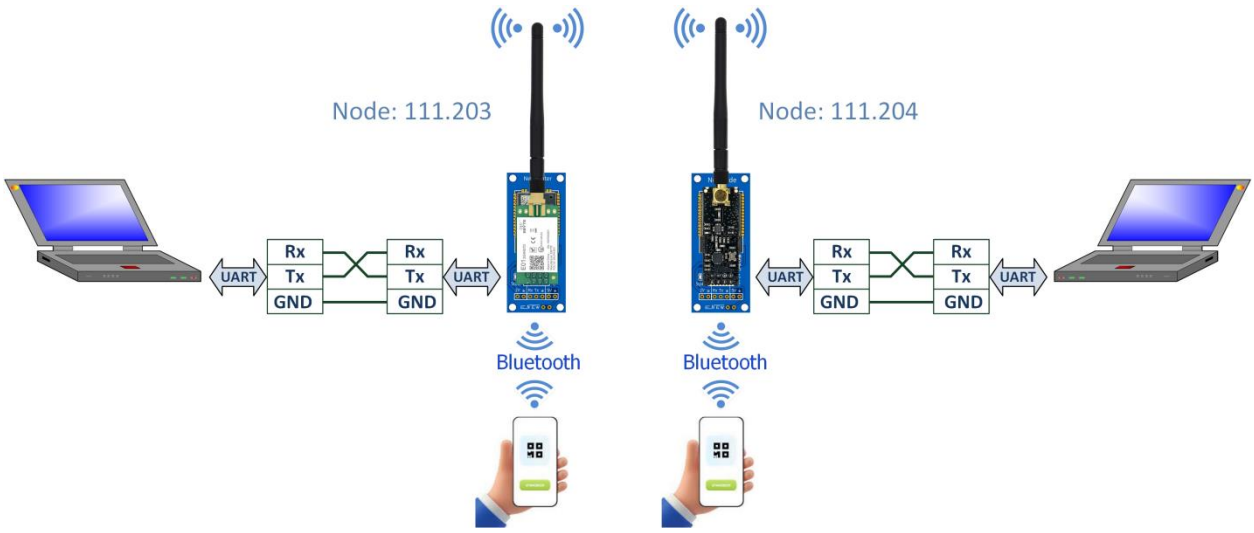


Рисунок 38 - Підключення комп'ютерів до вузлів мережі

Приклад підключення контролерів до вузлів мережі подано на рис. 37.

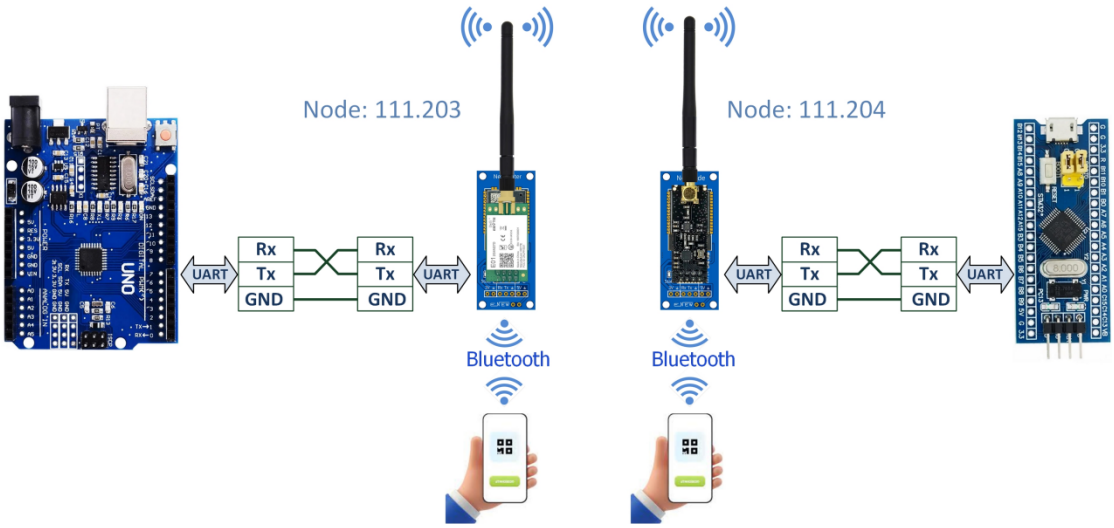


Рисунок 39 - Підключення контролерів до вузлів мережі

Якщо параметри вузлів мережі встановлено правильно, то мережа починає функціонувати. Перед початком експлуатації мережі рекомендується:

1. Просканувати канали та встановити номер головного каналу в середині найбільшої ділянки доступних каналів.

2. Виконати тестування за допомогою набору вбудованих тестів.

7. ОБМІН ДАНИМИ В МЕРЕЖІ. ПрикладИ ВИКОРИСТАННЯ

Порядок проходження байтів у переданому повідомленні не змінюється. Вузол-одержувач приймає дані з мережі та виводить їх у порт UART у тому самому порядку, у якому вони надійшли в порт UART вузла-відправника.

Тобто для користувача процес передачі даних є прозорим, як показано на рис. 40.

Швидкість портів UART вузлів встановлюється користувачем з урахуванням особливостей приймальних і передавальних пристроїв.

На Приклад, швидкість порту UART вузла-відправника може бути 921600 baud, а швидкість порту UART вузла-приймача - 115200 baud.

Рисунок 40 – Порядок проходження байтів у переданому повідомленні

Встановлення адреси вузла-одержувача

Перед надсиланням даних у мережі вузол-відправник має встановити адресу вузла одержувача в символьному або цифровому форматі. У цифровому форматі повідомлення більш компактне.

Встановлення адреси вузла-одержувача здійснюється трьома способами:

Спосіб 1: Використання AT-команди AT+TRG.addr a.

Адреса вузла-одержувача буде збережена в EEPROM. Усі наступні транзакції за замовчуванням здійснюватимуться тільки з цим вузлом.

Приклад:

```
AT+TRG.addr 203                    //target address = 203
Hello! This is a simple message    /ag/messe to node 203
AT+CLU.targ 01.05                  //target address:01.05(Cluster = 1 node
= 5)
Hello! This is a simple message    //message to node 01.05
```

Спосіб 2: Встановлення адреси вузла-одержувача на початку повідомлення. Адреса в EEPROM не зберігається. Усі наступні транзакції будуть здійснюватися тільки з цим вузлом.

Приклад:

Символьний формат:

```
*203*Hello! This is a simple message     //message to node 203
*205*Hello! This is a simple message     //message to node 205
//
*01.05*Hello! This is a simple message   //message to node 01.05
*02.07*Hello! This is a simple message   //message to node 02.07
```

Цифровий формат:

```
#CB#Hello! This is a simple message      //message to node 203 = 0xCB
#CD#Hello! This is a simple message      //message to node 205 = 0xCD
//
#25#Hello! This is a simple message      //message to node 02.05
#27#Hello! This is a simple message      //message to node 02.07
```

Спосіб 3: Встановлення адреси вузла-одержувача в окремому повідомленні. Адреса в EEPROM не зберігається. Усі наступні транзакції здійснюватимуться тільки з цим вузлом.

Приклад:

Символьний формат:

```
*203*                                  //target address = 203
Hello! This is a simple message        //message to node 203
//
*02.07*
Hello! This is a simple message        //message to node 02.07
```

Цифровой формат:

```
#CB#                                   //target address = 203/0xCB
Hello! This is a simple message        //message to node 203
```

```
#25#
Hello! This is a simple message          //message to node 02.05
```

Надсилання даних вузлу-одержувачу

Вузлу-одержувачу можуть бути надіслані будь-які дані в символьному та цифровому форматі.

Дані можуть бути відформатовані, якщо дані є командним рядком зі списком параметрів або бути простим текстовим рядком.

Завдання парсингу даних лежить на користувачеві. Вузол-одержувач виводить в UART дані в тому ж порядку, в якому вони були відправлені вузлом-відправником.

Приклад 1:

Передання вузлу 203 команди !XXXX! з параметрами в символьному форматі.

```
String DATA_OUT = "*203*!XXXX!12,12345,234567,345.678";
printf(DATA_OUT);                  //send data to network
```

Передача вузлу 01.05 команди !XXXX! із параметрами у символьному форматі.

```
String DATA_OUT = "*01.05*!XXXX!12,12345,234567,345.678";
printf(DATA_OUT);                  //send data to network
```

На стороні вузла-одержувача дані можна прийняти в такий спосіб:

```
#include "Parser.h"
enum package{_cmd,_parm_1,_parm_2,parm_3,_parm_4}; //data format
byte DATA_IN[150];
String cmd;
int8 var_8; int16 var_16;
int32 var_32;float var_fl;

//== input data parsing
parse_data(DATA_IN);

//== get parameters
cmd = get_token_string(_cmd);           // cmd = !XXXX!
var_8 = get_token_int8(_parm_1);        // var_8 = 12
var_16 = get_token_int16(_parm_2);      // var_16 = 12345
var_32 = get_token_int32(_parm_3);      // var_32 = 234567
        var_fl = get_token_float(_parm_4);      // var_fl = 345.678
```

8. РОБОТА З ВУЗЛОМ "CONTROLLER"

Призначення виводів вузла "Controller"

На друкованій платі вузла "Controller" для зручності використання всі виводи (пін, pin) позначені у форматі логічних номерів. Кожен пін може виконувати кілька функцій.

Функції пінів і відповідність їхніх логічних номерів фізичним номерам наведено в табл. 2.

Таблиця 2 - Вузол "Controller". Функції пінів і відповідність їхніх логічних номерів фізичним номерам

LOG_PIN	GPIO	ADC/PIN	DAC/PIN	SPI	I^2C	PWM/PIN	SERVO/PIN	GPS
01	27	2/1				1/1	1/1	
02	26	2/2	2/2			2/2	2/2	
03	25	2/3	1/3			3/3	3/3	
04	33	1/4				4/4	4/4	
05	35*	1/5						
06	34*	1/6						
07	36*	1/7						Rx
08	02	2/8				5/8	5/8	Tx
09	15	2/9		CS		6/9	6/9	
10	13	2/10		MOSI		7/10	7/10	
11	12	2/11		MISO		8/11	8/11	
12	14	2/12		SCK		9/12	9/12	
13	22	2/13			SCL	10/13	10/13	
14	21	2/14			SDA	11/14	11/14	

* - піни тільки для читання.

Команди управління вузлом Controller оперують з пінами, які мають логічні номери 01-14.

Класи команд управління функціями вузла "Controller"

Робота вузла «Controller» полягає у виконанні AT-команд, що надходять із мережі/Bluetooth/Wi-Fi та передачі їх у порт UART. AT-команди для вузла "Controller" мають три класи: "GET", "SET" та "CANCEL".

AT-команди класу GET повідомляють вузол Controller про те, що потрібно передати в мережу дані, зазначені в команді.

AT-команди класу "SET" встановлюють режим роботи вузла "Controller", вказаний у команді.

AT-команди класу "CANCEL" скасовують режим роботи вузла "Controller", вказаний у команді.

AT-команди мають два формати: повний та короткий. Повний формат зручно використовувати при ручному введенні команд, а короткий - для передачі команди в мережу. Формати AT-команд описані у Додатку 1.

8.1. МОДУЛЬ PIO. КОМАНДИ УПРАВЛІННЯ

Управління логічним рівнем пінів PIO

Модуль PIO може встановлювати на пінах вузла стан логічної "1" або логічного "0", а також надсилати в мережу поточне логічне значення одного або декількох пінів вузла, зазначених у команді.

Задача: Вузлу 203 встановити пін 1 у стан "0", пін 4 у стан "1", пін 8 у стан "0":

```
AT-команда    [*addr*]AT+PIN.SET
              pin:state[,pin:state][,pin:state]…[,pin:state]
Повернення    addr result  list old_pin_state>new_pin_state
              result: 0 cmd_Ok |1 cmd_ Error
 Приклад      *203*AT+PIN.SET 1:0,4:1,8:0
              203 0 cmd_Ok 1:1>0,4:1>1,8:1>0  //результат виконання команди
              203 1 cmd_Error                 //команду не виконано
```

Задача: Отримати стан пінів 1, 4, 8 вузла 203:

```
AT-команда    [*addr*]AT+PIN.GET pin[,pin][,pin]…[,pin]
Повернення    addr pin:state[,pin:state][,pin:state]…[,pin:state]|cmd_ Error
 Приклад      *203*AT+PIN.GET 1,4,8
Повернення    203 1:0,4:1,8:0
              203                   //номер вузла-відправника
              1:0,4:1,8:0           //номер піна та його стан
              203 1 cmd_Error       //команду не виконано
```

Встановлення режиму передавання повідомлень про стан пінів PIO у разі зміни їхнього логічного стану

Вузол "Controller" може надсилати повідомлення іншому вузлу при кожній зміні логічного стану одного або декількох пінів. Повідомлення передається тому вузлу, який встановив режим.

Задача: Встановити для вузла 203 режим передавання повідомлення при зміні логічного стану пінів:

```
АТ-команда    [*addr*]AT+PIN.SET_CHANGE_NOTIFY  pin[,pin][,pin]…[,pin]
Повернення    addr  0 cmd_Ok |1 cmd_ Error
 Приклад      *203*AT+PIN.SET_CHANGE_NOTIFY 1,4,8
Повернення    203 0 cmd_Ok           //команду виконано
              203 1 cmd_Error        //команду не виконано
```

Після встановлення режиму вузол буде відсилати в мережу повідомлення при кожній зміні логічного рівня зазначених пінів:

```
Приклад       203 1:0>1          //зміна логічного стану піна 1 з 0 на 1
              203 4:0>1          //зміна логічного стану піна 4 з 0 на 1
              203 4:1>0,8:0>1    //зміна логічного стану пінів 4 і 8
```

Відміна режиму передавання повідомлень про стан пінів PIO у разі зміни їхнього логічного стану

Задача: Відмінити для вузла 203 режим передавання повідомлення у разі зміни логічного стану пінів:

```
АТ-команда    [*addr*]AT+PIN.CANCEL_CHANGE_NOTIFY
Повернення    addr 0 cmd_Ok |1 cmd_ Error
 Приклад      *203*AT+PIN.CANCEL_CHANGE_NOTIFY
Повернення    203 0 cmd_Ok | 203 1 cmd_Error
```

Встановлення режиму передавання повідомлень про стан пінів PIO із зазначеним періодом

Вузол "Controller" може надсилати повідомлення іншому вузлу про стан пінів PIO із заданим періодом period. Повідомлення передається тому вузлу, який встановив режим

Задача: Встановити для вузла 203 режим передавання повідомлень про стан пінів із зазначеним періодом. Період у мілісекундах 1-10000 або в секундах 1-1000:

```
АТ-команда    [*addr*]AT+PIN.SET_PERIOD_STATE_NOTIFY_MS
              period,pin[,pin][,pin]…[,pin]
              [*addr*]AT+PIN.SET_PERIOD_STATE_NOTIFY_SEC
              period,pin[,pin][,pin]…[,pin]
```

Повернення addr 0 cmd_Ok | 1 cmd_ Error

Приклад *203*AT+PIN.SET_PERIOD_STATE_NOTIFY_MS 300,1,4,8 //період:300 ms
*203*AT+PIN.SET_PERIOD_STATE_NOTIFY_SEC 600,1,4,8//період:10 min.

Повернення 203 0 cmd_Ok | 203 1 cmd_Error

Після встановлення режиму вузол надсилатиме в мережу повідомлення про стан пінів із періодом 300 ms/10 min:

Приклад 203 1:0,4:1,8:0
203 1:0,4:1,8:0
203 1:0,4:1,8:1

Відміна режиму передавання повідомлень про стан пінів PIO із зазначеним періодом

Задача: Відмінити для вузла 203 режим передавання повідомлень про стан пінів із зазначеним періодом:

AT-команда [*addr*]AT+PIN.CANCEL_PERIOD_STATE_NOTIFY

Повернення addr 0 cmd_Ok | 1 cmd_ Error

Приклад *203*AT+PIN.CANCEL_PERIOD_STATE_NOTIFY

Повернення 203 0 cmd_Ok | 203 1 cmd_Error

8.2. МОДУЛЬ ADC. КОМАНДИ УПРАВЛІННЯ МОДУЛЕМ

Контролер ESP32 має два канали ADC: ADC_1 та ADC_2. Канал ADC_2 можна використовувати тільки тоді, коли модуль Wi-Fi вимкнений.

Отримання значення напруги на пінах ADC

Модуль ADC може надсилати в мережу поточне значення в мілівольтах одного або декількох пінів вузла, зазначених у команді.

Задача: Отримати значення напруги на пінах 4,5,6,7 вузла 203:

```
AT-команда    [*addr*]AT+ADC.GET_ADC pin[,pin][,pin]…[,pin]
Повернення    addr pin:val[,pin:val][,pin:val]…[,pin:val] |cmd_ Error
 Приклад      *203*AT+ADC.GET_ADC 4,5,6,7
Повернення    203 4:105,5:2700,6:1350,7:3500
              4:105,5:2700,6:1350,7:3500 //напруга на входах піна в mv
              203 1 cmd_Error
```

Встановлення режиму передавання повідомлень у разі зміни рівня напруги на пінах ADC

Вузол "Controller" може надсилати повідомлення іншому вузлу при кожній зміні рівня напруги на вході одного або декількох пінів. Повідомлення передається тому вузлу, який встановив режим.

Задача: Встановити для вузла 203 режим передавання повідомлення в разі зміни рівня напруги на вході одного або декількох пінів у разі перевищення заданого порога **threshold**:

```
AT-команда    [*addr*]AT+ADC.SET_CHANGE_NOTIFY  threshold_mv,
              pin[,pin][,pin]…[,pin]
Повернення    addr  0 cmd_Ok |1 cmd_ Error
 Приклад      *203*AT+ADC.SET_CHANGE_NOTIFY 50,4,5,6,7 //порог 50 mv
Повернення    203 0 cmd_Ok | 203 1 cmd_Error
```

Після встановлення режиму вузол буде відсилати в мережу повідомлення при кожній зміні рівня напруги на вході зазначених пінів:

```
Приклад       203 4:105>170 //зміна рівня напруги піна 4
              203 5:2700>21750 //зміна рівня напруги піна 5
              203 7:3500>3450 //зміна рівня напруги піна 7
```

Відміна режиму передавання повідомлень у разі зміни рівня напруги на пінах ADC

Задача: Відмінити для вузла 203 режим передавання повідомлення у разі зміни рівня напруги на пінах ADC:

АТ-команда	[*addr*]AT+ADC.CANCEL_CHANGE_NOTIFY
Повернення	addr 0 cmd_Ok |1 cmd_ Error
Приклад	*203*AT+ADC.CANCEL_CHANGE_NOTIFY
Повернення	203 0 cmd_Ok | 203 1 cmd_Error

Встановлення режиму передавання повідомлень про рівні напруги на пінах ADC із заданим періодом

Вузол "Controller" може надсилати повідомлення про рівні напруги на пінах ADC іншому вузлу із заданим періодом **period**. Повідомлення передається тому вузлу, який встановив режим.

Задача: Встановити для вузла 203 режим передавання повідомлень про рівні напруги на пінах ADC із заданим періодом. Період у мілісекундах 1-10000 або в секундах 1-1000:

АТ-команда	[*addr*]AT+ADC.SET_PERIOD_VALUE_NOTIFY_MS period,pin[,pin][,pin]…[,pin] [*addr*]AT+ADC.SET_PERIOD_VALUE_NOTIFY_SEC period,pin[,pin][,pin]…[,pin]
Повернення	addr 0 cmd_Ok | 1 cmd_ Error
Приклад	*203*AT+ADC.SET_PERIOD_VALUE_NOTIFY_MS 300,4,5,7 //період:300 ms *203*AT+ADC.SET_PERIOD_VALUE_NOTIFY_SEC 600,4,5,7//період:10 min.
Повернення	203 0 cmd_Ok | 203 1 cmd_Error

Після встановлення режиму вузол буде відсилати в мережу повідомлення про рівні напруги на пінах ADC з періодом 300 ms/10 min:

Пример	203 4:105 //рівень напруги на піні 4 203 5:2700 //рівень напруги на піні 5 203 7:3500 //рівень напруги на піні 7

Відміна режиму передавання повідомлень про рівні напруги на пінах ADC із заданим періодом

Задача: Відмінити для вузла 203 режим передавання повідомлень про рівні напруги на пінах ADC із зазначеним періодом:

```
АТ-команда    [*addr*]AT+ADC.CANCEL_PERIOD_VALUE_NOTIFY
Повернення    addr 0 cmd_Ok | 1 cmd_ Error
 Приклад      *203*AT+ADC.CANCEL_PERIOD_VALUE_NOTIFY
Повернення    203 0 cmd_Ok | 203 1 cmd_Error
```

8.3. МОДУЛЬ DAC. КОМАНДИ КЕРУВАННЯ МОДУЛЕМ

Вузол "Controller" має два піни DAC: DAC_1/PIN_3 і DAC_2/PIN_2. На пінах встановлюється значення заданої напруги з розрядністю 8 біт.

Встановлення рівня напруги на пінах DAC

Модуль DAC може встановлювати на пінах 2 і 3 заданий у команді рівень напруги в мілівольтах.

Задача: Вузлу 203 встановити на піні 2 напругу 1500 mv, а на піні 3 напругу 2500 mv:

```
АТ-команда    [*addr*]AT+DAC.SET pin:value[,pin:value]
Повернення    addr 0 cmd_Ok | 1 cmd_ Error
 Приклад      *203*AT+DAC.SET 2:1500,3:2500
              203 0 cmd_Ok | 203 1 cmd_Error
```

8.4. МОДУЛЬ PWM. КОМАНДИ УПРАВЛІННЯ МОДУЛЕМ

Вузол "Controller" має 11 пінів PWM. Для ініціалізації модуля PWM необхідно встановити такі параметри:

- частоту **frequency** у діапазоні 20-5000 Hz;

- розрядність **resolution** у діапазоні 8-12 bit;

- пін **pin** для виведення сигналу PWM (1 - 4; 8-14).

Управління роботою модуля PWM на зазначеному піні здійснюється встановленням параметра **duty** в діапазоні 0-1000, де 1000 = 100%.

Ініціалізація модуля PWM

Задача: Ініціалізувати модуль PWM вузла 203 з параметрами:

- **frequency** = 1000;

- **resolution** = 8;

- **pin** = 1,2,3.

AT-команда	`[*addr*]AT+PWM.SET frequency,resolution,pin[,pin][,pin]…[,pin]`
Повернення	`addr 0 cmd_Ok | 1 cmd_ Error`
Приклад	`*203*AT+PWM.SET 1000,8,1,2,3` `203 0 cmd_Ok | 203 1 cmd_Error`

Управління скважністю на пінах модуля PWM

Задача: На піні 1 вузла 203 встановити скважність 10%, на піні 2 - 25%, на піні 3 - 75%:

AT-команда	`[*addr*]AT+PWM.DUTY` `pin:value[,pin:value][,pin:value]…[,pin:value]`
Повернення	`addr 0 cmd_Ok | 1 cmd_ Error`
Приклад	`*203*AT+PWM.DUTY 1:100,2:250,3:750` `203 0 cmd_Ok | 203 1 cmd_Error`

Відміна режиму PWM

Задача: Відмінити для вузла 203 режим PWM

AT-команда	`[*addr*]AT+PWM.CANCEL`
Повернення	`addr 0 cmd_Ok | 1 cmd_ Error`

8.5. МОДУЛЬ SERVO. КОМАНДИ УПРАВЛІННЯ МОДУЛЕМ

Вузол "Controller" має 11 пінів для управління сервоприводами. Для ініціалізації модуля SERVO необхідно встановити такі параметри:

- пін **pin** для управління сервоприводом (1 - 4; 8-14).

Управління роботою модуля SERVO на зазначеному піні здійснюється встановленням параметрів:

- **degree** = 0 - 180; //кут установки вала сервопривода в градусах

- **time** = 0 - 10000 ms або 1-1000 sec. //час встановлення

Ініціалізація модуля SERVO

Задача: Ініціалізувати модуль SERVO вузла 203 для пінів 1,2,3,4:

```
AT-команда    [*addr*]AT+SERVO.SET pin[,pin][,pin]…[,pin]
Повернення    addr 0 cmd_Ok | 1 cmd_ Error
 Приклад      *203*AT+SERVO.SET 1,2,3,4
              203 0 cmd_Ok | 203 1 cmd_Error
```

Встановлення кута повороту вала сервоприводів на пінах модуля SERVO

Задача: Вал сервоприводу 1 встановити в положення 90 градусів,

Вал сервоприводу 2 встановити в положення 30 градусів за час 1500 ms,

Вал сервоприводу 3 встановити в положення 120 градусів за час 2000 ms,

Вал сервоприводу 4 встановити в положення 180 градусів за час 500 ms:

```
AT-команда    [*addr*]AT+SERVO.DEGREE
              pin:degree:time[,pin:degree:time]…[,pin:degree:time]
Повернення    addr 0 cmd_Ok | 1 cmd_ Error
 Приклад      *203*AT+SERVO.DEGREE 1:900,2:300:1500,3:1200:2000,4:1800:500
              203 0 cmd_Ok | 203 1 cmd_Error
```

Встановлення режиму сканування сервоприводів на пінах модуля SERVO

У режимі сканування вал сервоприводу циклічно переміщується від початкового значення кута повороту **beg_degree** до кінцевого значення кута

повороту **end_degree** за час **time** у секундах із зупинками в кінцевих точках на час **pause** у секундах.

Задача: Встановити режим сканування для двох сервоприводів на пінах 1 і 3 з параметрами:

Сервопривід 1: **beg_degree** = 0, **end_degree** = 1800, **time** = 10, **pause** = 0;

Сервопривід 2: beg_degree = 450, **end_degree** = 1350, **time** = 30, **pause** = 5:

AT-команда	[*addr*]AT+SERVO.SET_SCAN pin:beg_degree:end_degree:time:pause…
Повернення	addr 0 cmd_Ok | 1 cmd_ Error
Приклад	*203*AT+SERVO.SET_SCAN 1:0:1800:10,2:450:1350:30:5 203 0 cmd_Ok | 203 1 cmd_Error

Відміна режиму сканування сервоприводів на пінах модуля SERVO

Задача: Відмінити для вузла 203 режим сканування

AT-команда	[*addr*]AT+SERVO.CANCEL_SCAN
Повернення	addr 0 cmd_Ok | 1 cmd_ Error

Відміна режиму SERVO

Задача: Відмінити для вузла 203 режим SERVO

AT-команда	[*addr*]AT+SERVO.CANCEL
Повернення	addr 0 cmd_Ok | 1 cmd_ Error

8.6. МОДУЛЬ GPS. КОМАНДИ КЕРУВАННЯ МОДУЛЕМ

Модуль GPS є зовнішнім модулем і під'єднується у відповідний роз'єм вузла "Controller".

Модуль GPS управляється за допомогою AT-команд.

Підключення модуля GPS

Задача: Підключити модуль GPS:

AT-команда `[*addr*]AT+GPS+`
Повернення `addr  0 cmd_Ok | 1 cmd_ Error`
Приклад `*203*AT+GPS+`

Повернення `203 0 cmd_Ok | 203 1 cmd_Error`

Відключення модуля GPS

Задача: Відключити модуль GPS:

AT-команда `[*addr*]AT+GPS-`
Повернення `addr  0 cmd_Ok | 1 cmd_ Error`
Приклад `*203*AT+GPS-`

Повернення `203 0 cmd_Ok | 203 1 cmd_Error`

Отримання поточних координат GPS

Задача: Отримати короткі поточні координати модуля GPS вузла 203:

AT-команда `[*addr*]AT+GPS.GET_COORDINATE`
Повернення `ret_val | addr 0 cmd_Ok | 1 cmd_ Error`
Приклад `*203*AT+GPS.GET`

Повернення
```
//Latitude,Longitude
N48.09'47.6574",E17.08'11.2991"
203 0 cmd_Ok | 203 1 cmd_Error
```

Отримання розширених координат модуля GPS

Задача: Отримати розширені координати модуля GPS вузла 203:

AT-команда `[*addr*]AT+GPS.GET_DATA`
Повернення `ret_val | addr 0 cmd_Ok | 1 cmd_ Error`
Приклад `*203*AT+GPS.GET`

Повернення
```
//Latitude,Longitude,Distance(m),Altitude(m),Speed(m/s)
N48.09'47.6574",E17.08'11.2991",3500,160,3
203 0 cmd_Ok | 203 1 cmd_Error
```

Фіксація поточних координат модуля GPS

Задача: Зафіксувати поточні координати модуля GPS вузла 203:

```
АТ-команда    [*addr*]AT+GPS.FIX
Повернення    ret_val | addr 0 cmd_Ok | 1 cmd_ Error
 Приклад      *203*AT+GPS.GET
Повернення    //Latitude,Longitude
              N48.09'47.6574",E17.08'11.2991"
              203 0 cmd_Ok | 203 1 cmd_Error
```

Встановлення режиму передавання повідомлень у мережу повідомлення про зміну координат GPS

Вузол "Controller" може надсилати повідомлення іншому вузлу в разі зміни координат GPS, що перевищують вказане значення. Повідомлення передається тому вузлу, який встановив режим.

Задача: Встановити для вузла 203 режим передавання повідомлення при зміні координат GPS у разі перевищення заданого значення **threshold** (1-100 m):

```
АТ-команда    [*addr*]AT+GPS.SET_CHANGE_NOTIFY  threshold
Повернення    addr  0 cmd_Ok |1 cmd_ Error
 Приклад      *203*AT+GPS.SET_CHANGE_NOTIFY 10  //порог = 10 метров
Повернення    203 0 cmd_Ok | 203 1 cmd_Error
```

Після встановлення режиму вузол надсилатиме в мережу повідомлення при кожній зміні координат GPS, що перевищують зазначене значення:

```
 Приклад      N48.09'47.6574",E17.08'11.2991"
              N48.09'47.6751",E17.08'11.5622"
              N48.09'47.6751",E17.08'11.7988"
              N48.09'47.6554",E17.08'12.0528"
```

Відміна режиму передавання повідомлень у мережу повідомлення про зміну координат GPS

Задача: Відмінити для вузла 203 режим передавання повідомлення у разі зміни координат GPS:

АТ-команда	[*addr*]AT+GPS.CANCEL_CHANGE_NOTIFY
Повернення	addr 0 cmd_Ok |1 cmd_ Error
Приклад	*203*AT+GPS.CANCEL_CHANGE_NOTIFY
Повернення	203 0 cmd_Ok | 203 1 cmd_Error

Встановлення режиму передавання повідомлень у мережу координат GPS із заданим періодом

Вузол "Controller" може надсилати повідомлення з координатами GPS іншому вузлу із заданим періодом **period**. Повідомлення передається тому вузлу, який встановив режим.

Задача: Встановити для вузла 203 режим передавання повідомлень із координатами GPS із заданим періодом. Період у мілісекундах 100-10000 (0.1-10 sec):

АТ-команда	[*addr*]AT+GPS.SET_PERIOD_NOTIFY period
Повернення	addr 0 cmd_Ok | 1 cmd_ Error
Приклад	*203*AT+GPS.SET_PERIOD_NOTIFY 200 //период: 200 ms
Повернення	203 0 cmd_Ok | 203 1 cmd_Error

Після встановлення режиму вузол буде відсилати в мережу повідомлення з координатами GPS з періодом 200 ms:

Приклад	N48.09'47.6574",E17.08'11.2991"
	N48.09'47.6574",E17.08'11.2991"
	N48.09'47.6574",E17.08'11.2991"

Відміна режиму передавання повідомлень у мережу координат GPS із заданим періодом

Задача: Скасувати для вузла 203 режим передавання повідомлень із координатами GPS із заданим періодом:

АТ-команда	[*addr*]AT+GPS.CANCEL_PERIOD_NOTIFY
Повернення	addr 0 cmd_Ok | 1 cmd_ Error
Приклад	*203*AT+GPS.CANCEL_PERIOD_NOTIFY
Повернення	203 0 cmd_Ok | 203 1 cmd_Error

Додаток 1

АТ-КОМАНДИ КЕРУВАННЯ ПАРАМЕТРАМИ ВУЗЛІВ МЕРЕЖІ

Команда «AT+HELP»

Призначення: показати список АТ-команд. Регістр введення не має значення.

Приклад:

```
AT+HELP
```

У вікні терміналу відобразиться список АТ-команд:

```
======== Help for AT-commands ========
AT+HELP       > View AT-commands list

**** Net Mode Setup Section
AT+NET.config > View Network config
AT+NET.mode m > Network mode, m = 1...4
  Network mode:
  1 = Simple Net
  2 = Light Cluster Net
  3 = Medium Cluster Net
  4 = Union Cluster Net

**** Simple Net Address Setup Section
AT+NOD.config > View Node config
AT+NET.addr a > Network address
AT+NOD.addr a > This Node address
AT+TRG.addr a > Target Node address
       addr a = 1-254
AT+SRC.show   > Show sender address
AT+SRC.hide   > Hide sender address

**** Cluster Net Address Setup Section
AT+CLU.config > View Cluster config
AT+CLU.adm n  > This Admin number:  0-13
AT+CLU.node n > This Node number:   1-15
AT+CLU.targ n > Target Node number: 1-15
AT+ROUT.num n > Router number, n:R0-R30

**** Channel Mode Setup Section
AT+CNL.main n > Main channel number
            n = 1-125
AT+CNL.fixed  > Use one fixed channel
AT+CNL.multi  > Use many channels
AT+CNL.set s  > Channel set, s:1-50
AT+CNL.scan   > Scan and view channels
AT+CNL.view   > View available channels
```

```
**** Node Params Setup Section
AT+POWER pow  > Radio power,
   pow:1/2/3  = MIN/AVR/MAX
AT+BITRATE br > Radio bitrate,
   br:1/2/3   = 250Kb/1Mb/2Mb
AT+BAUD bd    > UART baud,
   bd:9600/115200/230400/460800/921600
AT+UART.flow+ > UART flow control ON
AT+UART.flow- > UART flow control OFF

**** Node Control Section
AT+START.hide > Hide start message
AT+START.show > Show start message
AT+DEFAULT    > Reset to default settings
AT+RESTART    > Node restart

**** Network Access Setup
AT+LOGIN log  > Login,    log:1-8 chars
AT+PASSW psw  > Password, psw:1-8 chars

**** Network Service Section
AT+HI         > Flash BLE connected node
AT+WHERE node > Flash node looking for
AT+ERROR.log  > View Error log
AT+NOD.scan   > Scan and view Net nodes
AT+NOD.view   > View available Net nodes

**** GPS Module Control Section
AT+GPS+       > GPS switch ON
AT+GPS-       > GPS switch OFF
AT+GPS.fix    > Fix GPS coordinates
AT+GPS.view   > View GPS coordinates

**** BLE Module Control Section
AT+BLE+       > BLE switch ON
AT+BLE-       > BLE switch OFF
AT+BLE.config > View BLE config
AT+BLE.name n > Set BLE name, n:name
AT+BLE.psw p  > Set BLE password, p:passw

****Test Mode Section
AT+STOP       > Test stop
AT+COUNTER target[,period]
  target   = target_addr,
  period   = 50...10000 ms, default: 500

AT+HELLO target[,period]
  target   = target_addr,
  period   = 50...10000 ms, default: 500
```

```
AT+HELLO.all [period]
  period   = 50...10000 ms, default: 500

AT+STRING target[,period][,length]
  target  = target_addr,
  period  = 50...1000 ms, default: 500
  length  = 1...150 bytes, default: 50

**** Message Format Section
*target addr* message 1...256 bytes
```

****** Net Mode Setup Section

Команда «AT+NET.config»

Призначення: показати конфігурацію мережі. Буде відображено список вузлів мережі відповідно до поточної моделі мережі.

Приклад:

```
AT+NET.config
```

У вікні терміналу відобразиться конфігурація мережі:

```
===== Network Configuration =====

> CLUSTER   ADDR    NODE_TYPE   STATE
  0         0.0     Admin       ON
  0         0.1     Lite Node   ON
  0         0.2     Lite Node   ON
  0         0.3     Controller  ON
  0         0.4     Controller  ON
  0         0.5     Controller  ON
  0         0.6     Lite Node   OFF
  ---------------------------
> CLUSTER   ADDR    NODE_TYPE   STATE
  1         1.0     Admin       ON
  1         1.1     Controller  ON
  1         1.2     Controller  ON
  1         1.3     Controller  ON
  1         1.4     Controller  ON
  1         1.5     Controller  ON
  ---------------------------
> ROUTER    ADDR    NODE_TYPE   STATE
  1         R0      Single      ON
  2         R1/R2   Fast        ON
  3         R3/R4   Fast        ON
  4         R5      Single      ON
  ---------------------------
> REPEATOR  ADDR   TYPE         STATE
  1         R17    Repeator     OFF
```

Команда «AT+NET.mode m»

Призначення: встановити конфігурацію мережі.

Параметр m набуває значень:

1 = Simple Net

2 = Light Cluster Net

3 = Medium Cluster Net

4 = Union Cluster Net

Приклад:

```
AT+NET.mode 2
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> mode LIGHT_CLUSTER_NET: Ok
```

Якщо параметр AT-команд введено неправильно, то буде відображено повідомлення про помилку.

Приклад:

```
AT+NET.mode 5

> Error: Incorrect value: 5
```

******** Simple Net Addresses Setup Section**

Команда «AT+NOD.config»

Призначення: показати конфігурацію вузла.

Приклад:

AT+NOD.config

У вікні терміналу відобразиться поточна конфігурація вузла:

```
*** NODE CONFIG ***

NETWORK MODE    : SIMPLE_NET
--------------- Network Adresses
NETWORK ADDR    : 111
THIS NODE ADDR  : 203
TARG NODE ADDR  : 0
BROADCAST ADDR  : 255
```

```
SHOW  SRC ADDR  : ON
--------------- Channels
CHANNEL         : 85
CHANNEL MODE    : MULTI
CHANNEL SET     : 10
--------------- Node Params
RADIO POWER     : MAX
RADIO BITRATE   : 250 Kbit
UART BAUD       : 921600
--------------- Network Access
NETWORK LOGIN   : 1234
NETWORK PASSW   : 1234
--------------- Modules State
BLUETOOTH       : ON
BLUETOOTH NAME  : 111.203 Node For Test
GPS/GLONASS     : OFF
```

Команда «AT+NET.addr a»

Призначення: присвоїти адресу мережі.

Параметр a приймає значення: 1-254.

Приклад:

```
AT+NET.addr 234
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Network Address 234: Ok
```

Команда «AT+NOD.addr a»

Призначення: присвоїти адресу вузлам "Net Master", "Controller" і "Light Node".

Параметр **a** приймає значення: 1-254.

Приклад:

```
AT+NOD.addr 204
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> This Node Address 204: Ok
```

Команда «AT+TRG.addr a»

Призначення: присвоїти адресу вузлу-одержувачу.

Параметр **a** приймає значення: 1-254.

Приклад:

```
AT+ TRG.addr 127
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Target Node Address 127: Ok
```

Команда «AT+SRC.show»

Призначення: дозволяє виведення адреси вузла-відправника в UART вузла-приймача. Сервісна функція. Служить для візуальної ідентифікації користувачем вузла-відправника.

Приклад:

```
AT+SRC.show
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Sender address put to UART: Ok
```

Адреса вузла-відправника буде виводитися в UART вузла-приймача.

Команда «AT+SRC.hide»

Призначення: скасовує виведення в UART вузла-приймача адреси вузла-відправника.

Приклад:

```
AT+SRC.hide
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Sender address don't put to UART: Ok
```

Адреса вузла-відправника не буде виводитися в UART вузла-приймача.

******** Cluster Net Addresses Setup Section**

Команда «AT+CLU.config»

Призначення: показати конфігурацію поточного кластера.

Приклад:

```
AT+CLU.config
```

У вікні терміналу відобразиться поточна конфігурація кластера:

```
> Cluster Admin address: 08.00
   Node addresses:
   08.01
   08.02
   08.03
   08.04
```

Команда «AT+CLU.adm n»

Призначення: присвоїти номер вузлу "Cluster Admin".

Параметр **n** приймає значення: 1-13.

Приклад:

```
AT+CLU.adm 12
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> This Cluster Admin number 12: Ok
```

Команда «AT+CLU.node n»

Призначення: присвоїти номер вузлам кластера "Controller" або "Light Node".

Параметр **n** приймає значення: 1-15.

Приклад:

```
AT+CLU.node 12.15
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> This Cluster Node number 12.15: Ok
```

Команда «AT+CLU.targ n»

Призначення: присвоїти номер вузлам кластера-одержувача "Controller" або "Light Node".

Параметр **n** приймає значення: 0-13.1-15.

Приклад:

```
AT+CLU.targ 12.15
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Target Cluster Node number 12.15: Ok
```

Команда «AT+ROUT.num n»

Призначення: присвоїти номер вузлу "Net Router".

Параметр **n** приймає значення: R0-R30.

Приклад:

```
AT+ AT+ROUT.num R15
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> This Router number 15: Ok
```

******** Channel Mode Setup Section**

Команда «AT+CNL.main n»

Призначення: встановити для мережі номер основного каналу.

Параметр **n** приймає значення: 1-125.

Приклад:

```
AT+CNL.main 85
```

У вікні термінала відобразиться повідомлення про те, що АТ-команду виконано успішно.

```
> Main channel #85: Ok
```

Команда «AT+CNL.fixed»

Призначення: встановити для мережі один фіксований номер каналу.

Приклад:

```
AT+CNL.fixed
```

У вікні термінала відобразиться повідомлення про те, що АТ-команду виконано успішно.

```
> Fixed channel mode: Ok
```

Команда «AT+CNL.multi»

Призначення: дозволити вузлам мережі використовувати безліч каналів.

Приклад:

```
AT+CNL.multi
```

У вікні термінала відобразиться повідомлення про те, що АТ-команду виконано успішно.

```
> Multi channels mode: Ok
```

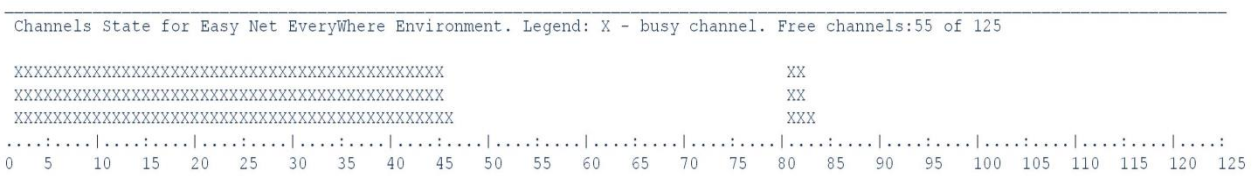
Команда «AT+CNL.scan»

Призначення: сканувати і відобразити всі канали мережі з метою визначення вільних і зайнятих каналів.

Приклад:

```
AT+CNL.scan
```

У вікні терміналу відобразиться результат сканування каналів:



Команда «AT+CNL.view»

Призначення: показати доступні канали.

Приклад:

```
AT+CNL.view
```

У вікні терміналу відобразиться список доступних каналів:

```
*> Available Channels:91

 31  35  37  38  39  40  41  42  43  44
 45  46  47  48  49  50  51  52  53  54
 55  56  57  58  59  60  61  62  63  64
 65  66  67  68  69  70  71  72  73  74
 75  76  77  78  79  80  81  82  83  84
 86  87  88  89  90  91  92  93  94  95
 96  97  98  99 100 101 102 103 104 105
106 107 108 109 110 111 112 113 114 115
116 117 118 119 120 121 122 123 124 125
126
```

Команда «AT+CNL.set s»

Призначення: виділити для мережі безліч робочих каналів s.

Параметр **s** приймає значення: 1-60.

Приклад:

```
AT+CNL.set 10
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Installed set of 10 channels: Ok
```

******** Node Params Setup Section**

Команда «AT+POWER pow»

Призначення: встановити потужність трансивера.

Параметр **pow** приймає значення: 1/2/3 = MIN/AVR/MAX.

Приклад:

```
AT+POWER 3
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> RF power MAX: Ok
```

Команда «AT+BITRATE br»

Призначення: встановити швидкість передавання даних трансивера.

Параметр br набуває значень: 1/2/3 = 250Kb/1Mb/2Mb.

Приклад:

```
AT+BITRATE 1
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно:

```
> RF bitrate 250 Kbit: Ok
```

Команда «AT+BAUD bd»

Призначення: встановити швидкість передавання даних порту UART.

Параметр bd приймає значення: 9600/115200/230400/460800/921600 baud.

Приклад:

```
AT+BAUD 921600
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> UART baud 921600: Ok
Don't forget to switch to installed baud
```

Команда «AT+UART.flow+»

Призначення: увімкнути керування потоком даних UART XON/XOFF. Відправник буде призупиняти передачу даних в UART при отриманні байта XOFF і відновлювати при отриманні байта XON.

Управління потоком даних запобігає переповненню вхідного буфера UART.

Приклад:

```
AT+UART.flow+
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно:

```
> UART flow control ON: Ok
```

Команда «AT+UART.flow-»

Призначення: вимкнути керування потоком даних UART XON/XOFF.

Приклад:

```
AT+UART.flow-
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно:

```
> UART flow control OFF: Ok
```

******** Node Control Section**

Команда «AT+START.hide»

Призначення: заборонити виведення повідомлення з параметрами вузла під час старту.

Приклад:

```
AT+START.hide
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Start message hide: Ok
```

Під час старту вузла у вікні термінала відобразиться повідомлення:

```
>> Node 203 started
```

Команда «AT+START.show»

Призначення: дозволити виведення повідомлення з параметрами вузла під час старту.

Приклад:

```
AT+START.show
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Start message show: Ok
```

Під час старту вузла у вікні термінала відобразиться повідомлення:

```
> Node 203 ready to start. Wait..
**********************
*  Easy Net EveryWhere
*  -------------------
*  Network.mode:   SIMPLE_NET
*  Radio.bitrate:  250Kb
*  UART.baud:      921600
*  Bluetooth:      ON
*  Bluetooth name: 111.203 Node For Test
**********************
>> Node 203 started
```

Команда «AT+DEFAULT»

Призначення: скинути налаштування параметрів вузла до початкових значень.

Приклад:

```
AT+DEFAULT
```

Вузол встановить параметри за замовчуванням.

Команда «AT+RESTART»

Призначення: перезавантажити вузол.

Приклад:

```
AT+RESTART
```

У вікні термінала відобразиться повідомлення про перезавантаження, після чого вузол буде перезавантажено.

```
> The node will reboot now
**********************
*  Easy Net EveryWhere
*  -------------------
*  Network.model:  SIMPLE_NET
*  Radio.bitrate:  250Kb
*  UART.baud:      115200
*  Bluetooth:      ON
*  Bluetooth name: 111.203 NET_NODE
**********************
      >> Node 203 started
```

******** Network Access Setup**

Команда «AT+LOGIN log»

Призначення: встановити мережевий логін.

Параметр **log** набуває значень: будь-які символи і цифри довжиною до 8 байт.

Приклад:

```
AT+LOGIN 12345678
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Network Login Ok: 12345678
```

Команда «AT+PASSW psw»

Призначення: встановити мережевий пароль.

Параметр **psw** набуває значень: будь-які символи і цифри довжиною до 8 байт.

Приклад:

```
AT+PASSW 876954321
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Network Password Ok: 876954321
```

******** Network Service Section**

Команда «AT+HI»

Призначення: знайти вузол, з яким встановлено з'єднання через Bluetooth (BLE).

Світлодіод "State" шуканого вузла блиматиме протягом двох секунд.

Приклад:

```
AT+HI
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> The node you are connected is flashing
```

Команда «AT+WHERE node»

Призначення: знайти вузол із зазначеною адресою.

Параметр **node** приймає значення: 1-254/00.01-13.15

Світлодіод "State" шуканого вузла блиматиме протягом двох секунд.

Приклад:

```
AT+WHERE 205 или AT+WHERE 12.14
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> The node you are looking for is flashing
```

Команда «AT+ERROR.log»

Призначення: показати журнал із помилками мережі та статистикою.

Приклад:

```
AT+ERROR.log 01.00
```

У вікні термінала відобразиться вміст журналу помилок кластера 1. Якщо параметр node не вказувати, то вузол "Net Master" відобразить вміст журналу помилок усієї мережі.

```
===== Network Channel Error =====
> SENDER  TARG    CNL     ERROR   QUALITY
  1.0     1.1     62      12      ***
  1.1     1.0     62      9       ****
  1.0     1.5     62      14      **
  1.5     1.0     63      16      **
```

Із записів журналу випливає, що канали 62 і 63 мають помилки.

При досягненні порога помилок у 20% мережа автоматично виключає канал зі списку доступних.

Команда «AT+NOD.scan»

Призначення: сканувати вузли мережі.

Приклад:

```
AT+NOD.scan
```

У вікні термінала відобразиться список знайдених вузлів мережі:

```
> Scan nodes begin. Wait...
*> Nodes found: 14
 20  21  22  23  24  25  26  27  28  29
123 204 207 208
```

Команда «AT+NOD.view»

Призначення: показати всі вузли в мережі.

Приклад:

```
AT+NOD.view
```

У вікні термінала відобразиться список знайдених вузлів мережі:

```
*> Nodes found: 15
 20  21  22  23  24  25  26  27  28  29
123 204 205 207 208
```

****** GPS Module Control Section**

Команда «AT+GPS+»

Призначення: увімкнути функцію роботи з GPS модулем.

Приклад:

```
AT+GPS+
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> GPS state: ON
```

Команда «AT+GPS-»

Призначення: вимкнути функцію роботи з GPS модулем.

Приклад:

```
AT+GPS-
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> GPS state: OFF
```

Команда «AT+GPS.fix»

Призначення: зафіксувати поточні координати GPS модуля.

Приклад:

```
AT+GPS.fix
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> GPS coordinates fixed: зафіксовані координати широти і довготи
```

Якщо зафіксувати поточні координати GPS не вдалося, то з'явиться повідомлення:

```
> GPS fix error
> GPS Coordinates NOT fixed
```

Команда «AT+GPS.view»

Призначення: показати поточні координати GPS модуля.

Приклад:

```
AT+GPS.view
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> GPS current coordinates: поточні координати широти та довготи
```

****** BLE Module Control Section**

Команда «AT+BLE+»

Призначення: увімкнути функцію роботи з Bluetooth (BLE) модулем.

Приклад:

```
AT+BLE+
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Bluetooth state: ON
```

Команда «AT+BLE-»

Призначення: вимкнути функцію роботи з Bluetooth (BLE) модулем.

Приклад:

```
AT+BLE-
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> Bluetooth state: OFF
```

Команда «AT+BLE.config»

Призначення: подивитися поточні установки BLE модуля вузла.

Приклад:

```
AT+BLE.config
```

У вікні терміналу будуть відображені поточні установки BLE модуля:

```
*** BLE DEVICE CURRENT CONFIG ***
    BLE device Password: 12345678
    BLE device Name:
    111.203 Node For Test
```

Команда «AT+BLE.name n»

Призначення: присвоїти ім'я BLE модулю вузла.

Параметр n приймає значення: будь-які символи і цифри довжиною до 24 байт.

Приклад:

```
AT+BLE.name Node For Test
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> BLE Node name: Ok  111.203 Node For Test
```

Команда «AT+BLE.psw p»

Призначення: встановити пароль доступу до BLE модуля вузла.

Параметр **p** приймає значення: будь-які символи і цифри довжиною до 8 байт.

Приклад:

```
AT+BLE.psw 12345678
```

У вікні термінала відобразиться повідомлення про те, що АТ-команду виконано успішно.

```
> BLE password: Ok  12345678
```

******** Test Mode Section**

Команда «AT+STOP»

Призначення: зупинити поточний тест.

Приклад:

```
AT+STOP
```

У вікні термінала відобразиться повідомлення про те, що AT-команду виконано успішно.

```
> STOP
```

Поточний тест буде зупинено.

Команда «AT+COUNTER target[,period]»

Призначення: запустити тест для перевірки передавання даних між двома вузлами.

У тесті вузлом-відправником надсилається поточне значення лічильника вузлу-одержувачу.

Якщо вузол-одержувач отримав значення лічильника без помилок, то вузлу-відправнику надсилається квитанція ACK і вузол-відправник інкрементує значення лічильника.

Вузол-приймач фіксує кількість помилок лічильника і виводить на екран терміналу поточні значення.

Параметр **target** - адреса вузла-одержувача.

Параметр **period** - період надсилання тестових пакетів: 50...10000 ms. За замовчуванням: 500 ms.

Приклад:

```
AT+COUNTER 203,100
```

У вікні термінала вузла-відправника з'явиться повідомлення про те, що AT-команду виконано успішно, відправлене поточне значення лічильника і значення лічильника помилок, передане вузлом-одержувачем.

Якщо до вузла під'єднано та активовано модуль GPS, то відстань між вузлами також відображатиметься з похибкою 2 метри:

```
> Counter test started
>to:203|cnt:0000001|err:0|s=623 м|
>to:203|cnt:0000002|err:0|s=623 м|
>to:203|cnt:0000003|err:0|s=623 м|
>to:203|cnt:0000004|err:0|s=623 м|
```

У вікні термінала вузла-одержувача виводитиметься отримане значення лічильника:

```
>from:204|cnt:0000001|err:0|
>from:204|cnt:0000002|err:0|
>from:204|cnt:0000003|err:0|
>from:204|cnt:0000004|err:0|
```

Команда «AT+HELLO target[,period]»

Призначення: запустити тест для перевірки передачі даних між двома вузлами.

У тесті вузлом-відправником надсилається повідомлення "Hello from node".

Якщо вузол-одержувач отримав повідомлення, то у вікні терміналу вузла-одержувача буде відображатися повідомлення, адреса вузла-відправника і номер каналу, по якому було прийнято повідомлення.

Параметр **target** - адреса вузла-одержувача.

Параметр **period** - період надсилання тестових пакетів: 50...10000 ms. За замовчуванням: 500 ms.

Приклад:

```
AT+ HELLO 203,300
```

У вікні термінала вузла-одержувача з адресою 204 відобразиться повідомлення про те, що АТ-команду виконано успішно:

```
> Hello message start
0 203 Ok
0 203 Ok
0 203 Ok
```

У вікні термінала вузла-одержувача з адресою 203 виводитиметься повідомлення:

```
> Hello from node: 204 channel: 61
> Hello from node: 204 channel: 59
> Hello from node: 204 channel: 58
```

Команда «AT+HELLO.all [period]»

Призначення: запустити тест для перевірки передавання даних між кількома вузлами.

У тесті вузлом-відправником усім вузлам мережі по черзі надсилається повідомлення "Hello from node".

Якщо вузол-одержувач отримав повідомлення, то у вікні терміналу вузла-одержувача відображатиметься повідомлення, адреса вузла-відправника і номер каналу, яким було прийнято повідомлення.

Параметр **target** - адреса вузла-одержувача.

Параметр **period** - період надсилання тестових пакетів: 50...10000 ms. За замовчуванням: 500 ms.

Приклад:

```
AT+HELLO.all 100
```

У вікні термінала вузла-відправника відобразиться повідомлення про те, що AT-команду виконано успішно:

```
> Hello to all message start
0 28 Ok
1 29 NO_CONNECTION
0 123 Ok
```

У вікні терміналу вузлів-одержувачів виводитиметься повідомлення:

```
> Hello from node: 203 channel: 76
> Hello from node: 203 channel: 80
> Hello from node: 203 channel: 79
```

Команда «AT+STRING target[,period][,length]»

Призначення: запустити тест для перевірки передавання даних між двома вузлами.

У тесті вузлом-відправником вузлу-одержувачу надсилається текстовий рядок довжиною, заданою користувачем.

Якщо вузол-одержувач отримав повідомлення, то у вікні терміналу вузла-одержувача відображатиметься прийнятий рядок.

Параметр **target** - адреса вузла-одержувача.

Параметр **period** - період надсилання тестових пакетів: 50...10000 ms. За замовчуванням: 500 ms.

Параметр **length** - довжина рядка: 1-150 байт. За замовчуванням: 50 байт.

Приклад:

```
AT+STRING 203
```

У вікні термінала вузла-відправника відобразиться повідомлення про те, що АТ-команду виконано успішно:

```
> Test Sequence start
0 203 Ok
0 203 Ok
0 203 Ok
0 203 Ok
```

У вікні терміналу вузлів-одержувачів виводитиметься повідомлення:

```
123456789_123456789_123456789_123456789_123456789_
123456789_123456789_123456789_123456789_123456789_
123456789_123456789_123456789_123456789_123456789_
123456789_123456789_123456789_123456789_123456789_
```

Додаток 2

Формати AT-команд

Повний формат	**Короткий формат:**
AT+HELP	011
AT+NET.config	012
AT+NET.mode	013
AT+NOD.config	014
AT+NET.addr	015
AT+NOD.addr	015
AT+TRG.addr	017
AT+SRC.show	018
AT+SRC.hide	019
AT+CLU.config	021
AT+CLU.adm	022
AT+CLU.node	023
AT+CLU.targ	024
AT+ROUT.num	025
AT+CNL.main	026
AT+CNL.fixed	027
AT+CNL.multi	028
AT+CNL.scan	029
AT+CNL.view	02A
AT+CNL.set	02B
AT+POWER	031
AT+BITRATE	032
AT+BAUD	033
AT+UART.flow+	034
AT+UART.flow-	035
AT+START.hide	036
AT+START.show	037
AT+DEFAULT	038
AT+RESTART	039
AT+LOGIN	03A
AT+PASSW	03B
AT+HI	041
AT+WHERE	042
AT+ERROR.log	043
AT+NOD.scan	044
AT+NOD.view	045
AT+BLE+	046
AT+BLE-	047
AT+BLE.config	048
AT+BLE.name	049
AT+BLE.psw	04A
AT+STOP	051

AT+COUNTER	052
AT+HELLO	053
AT+HELLO.all	054
AT+STRING	055
AT+PIN.SET	061
AT+PIN.GET	062
AT+PIN.SET_CHANGE_NOTIFY	063
AT+PIN.CANCEL_CHANGE_NOTIFY	064
AT+PIN.SET_PERIOD_STATE_NOTIFY_MS	065
AT+PIN.SET_PERIOD_STATE_NOTIFY_SEC	066
AT+PIN.CANCEL_PERIOD_STATE_NOTIFY	067
AT+ADC.GET_ADC	071
AT+ADC.SET_CHANGE_NOTIFY	072
AT+ADC.CANCEL_CHANGE_NOTIFY	073
AT+ADC.SET_PERIOD_VALUE_NOTIFY_MS	074
AT+ADC.SET_PERIOD_VALUE_NOTIFY_SEC	075
AT+ADC.CANCEL_PERIOD_VALUE_NOTIFY	076
AT+DAC.SET	077
AT+PWM.SET	081
AT+PWM.DUTY	082
AT+PWM.CANCEL	083
AT+SERVO.SET	086
AT+SERVO.DEGREE	087
AT+SERVO.SET_SCAN	088
AT+SERVO.CANCEL_SCAN	089
AT+SERVO.CANCEL	08A
AT+GPS+	091
AT+GPS-	092
AT+GPS.GET_COORDINATE	093
AT+GPS.GET_DATA	094
AT+GPS.FIX	095
AT+GPS.SET_CHANGE_NOTIFY	096
AT+GPS.CANCEL_CHANGE_NOTIFY	097
AT+GPS.SET_PERIOD_NOTIFY	098
AT+GPS.CANCEL_PERIOD_NOTIFY	099

Список літератури

1. https://ru.wikipedia.org/wiki/ESP32/(дата звернення: 12.03.2024).
2. Смірнов В.В., Смірнова Н.В., Пархоменко Ю.М. Апаратно-програмний комплекс «Internet Of Things Development Board». Матеріали iii форуму «Автоматизація, електроніка та робототехніка. Стратегії розвитку та інноваційні технології» AERT-2021 29 жовтня 2021 р., Харківський національний університет радіоелектроніки, Харків, 2021. – С. 18 - 21.
3. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols /index.html/ (дата звернення: 12.03.2024).
4. https://coderlessons.com/tutorials/stsenarii/vyuchi-lua/lua-tutorial/(дата звернення: 12.03.2024).
5. IEEE 802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks. Standards Committee : C/LM - LAN/MAN Standards Committee. 2020.05.06. URL: https://standards.ieee.org/standard/802_15_4-2020.html (дата звернення: 12.03.2024).
6. Recommendation G.9959. URL: https://www.itu.int/rec/T-REC-G.9959-201310-S!Amd1/en (дата звернення: 12.03.2024).
7. LoRaWAN™ Specification, N.Sornin (Semtech), M.Luis (Semtech), T.Eirich (IBM), T.Kramp (IBM), O.Hersent (Actility), V1.0, 2015 January.
8. Смирнов В.В., Смирнова Н.В. Бездротова локальна мережа класу Smart Home на базі модулів сплітерів-репітерів. Загальнодержавний міжвідомчий науково-технічний збірник Конструювання, виробництво та експлуатація сільськогосподарських машин. Кропивницький: ЦНТУ, 2021. Вип. 51. С. 195-202 (Фахове видання).
9. Смірнов В.В., Смірнова Н.В. Мобільна mesh-мережа для управління роєм об'єктів. Центральноукраїнський науковий вісник. Технічні науки. 2023. Вип. 7(38), ч.II. С. 3-11 (Фахове видання).
10. Смірнов В.В., Смірнова Н.В., Практична реалізація безпровідної мережі «Easy Net Everywhere». M 58 International security studios: managerial, technical, legal, environmental, informative and psychological aspects. International collective monograph. Volume I. NMBU, Research and Education. 2024. - 700 p. Oslo, Kingdom of Norway - 2024. С.665 - 697.

Навчальне електронне видання комбінованого використання.
Можна використовувати в локальному та мережному режимах

Смірнов Володимир Вікторович
кандидат технічних наук, доцент

Смірнова Наталія Володимирівна
кандидат технічних наук, доцент

Програмування пристроїв Internet Of Things на базі мікроконтролера ESP8266 і мові програмування LUA

Навчальний посібник

Редактор В.В. Смірнов
Технічний редактор В.В. Смірнов
Верстальник Н.В. Смірнова

Видавець
Центральноукраїнський національний технічний університет
25006, м. Кропивницький. пр. Університетський, 8,

тел. +38-091-608-75-02
E-mail: swckntu@gmail.com