Read the Docs Template Documentation

Release v1.0-156-gcf50274

Read the Docs

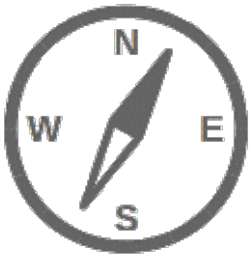
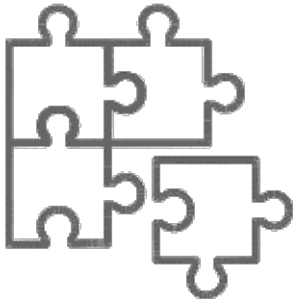
Mar 09, 2023

Contents

[]

This is the documentation for Espressif Mesh Development Framework(esp-mdf).

The documentation has different language versions (English, , *How to switch between languages?*). However, please refer to the English version if there is any discrepancy.

		
Get Started	API Reference	H/W Reference
		
API Guides	Contribute	Resources

CHAPTER 1

Introduction

[]

It is intended to guide users to build a software environment for ESP-MDF. ESP-MDF is a development framework based on ESP-WIFI-MESH which is encapsulated by ESP-IDF. Therefore, the building of the software environment for ESP-MDF is similar to the building for ESP-IDF. This document focuses on the software environment difference between ESP-MDF and ESP-IDF, as well as provides some related notes. Before developing with ESP-MDF, please read ESP-IDF Get Started.

1.1 What you need

To develop applications for ESP32 you need:

- **Router**: it is used to connect to the external network
- **Mobile phone**: install an ESP-WIFI-MESH network configuration app
- **ESP32 development board**: at least two ESP32 development boards are required to build an ESP-WIFI-MESH network

1.2 Development board guides

We provide development boards specially designed for the development and testing of ESP-WIFI-MESH.

1.2.1 ESP32-Buddy

ESP32-Buddy is a development board specifically designed to test the development of ESP-WIFI-MESH. With its small size and USB power input, the board can be conveniently used for testing a large number of devices and measure distances between them.

- Purchase link: coming soon

- **Functions:**
  - 16 MB flash: stores logs
  - OLED screen: displays information about the device, such as its layer, connection status, etc.
  - LED: indicates the board's status
  - Temperature & humidity sensor: collects environmental parameters

1.2.2 ESP32-MeshKit

ESP32-MeshKit offers a complete [ESP-Mesh Lighting Solution](https://www.espressif.com/en/products/software/esp-mesh/overview), complemented by ESP-Mesh App for research, development and better understanding of ESP-WIFI-MESH.

- Purchase link: [taobao](https://item.taobao.com/item.htm?spm=a230r.1.14.1.55a83647K8jlrh&id=573310711489&ns=1&abbucket=3#detail)
- **Products:**
  - ESP32-MeshKit-Light: The RGBW smart lights that show control results visually. They can be used to test network configuration time, response speed, stability performance, and measure distance, etc.
  - ESP32-MeshKit-Sense: This kit is equipped with a light sensor as well as a temperature & humidity sensor. It can measure power consumption and develop low power applications. The kit may also be used with ESP-Prog for firmware downloading and debugging.
  - ESP32-MeshKit-Button: Serves as an on/off controller, ready for the development of low power applications. It can be used with ESP-Prog for firmware downloading and debugging.

1.3 Get ESP-MDF

Besides the toolchain (that contains programs to compile and build the application), you also need ESP32 specific API / libraries. They are provided by Espressif in ESP-MDF repository.

To obtain a local copy: open terminal, navigate to the directory you want to put ESP-MDF, and clone the repository using `git clone` command:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-mdf.git
```

ESP-MDF will be downloaded into `~/esp/esp-mdf`.

Note: This command will clone the master branch, which has the latest development ("bleeding edge") version of ESP-MDF. It is fully functional and updated on weekly basis with the most recent features and bugfixes.

Note: GitHub's "Download zip file" feature does not work with ESP-MDF.

1.4 Set up path to ESP-MDF

The toolchain progarms access ESP-MDF using `MDF_PATH` environment variable. This variable should be set up on your PC, otherwise projects will not build. The setup method is same as `IDF_PATH`.

1.5 Configure

In the terminal window, go to the directory of `get-started` by typing `cd ~/esp/get-started`, and then start project configuation utility `menuconfig`:

```
cd ~/esp/get-started
make menuconfig
```

- Configure `examples` under the submenu `Example Configuration`
- Configure the function modules under the submenu starting with `MDF` in `Component config`

1.6 Build and Flash

You can configure the serial port with *make menuconfig*, or directly use `ESPPORT` and `ESPBAUD` environment variable on the command line to specify the serial port and baud rate:

```
make erase_flash flash -j5 monitor ESPBAUD=921600 ESPPORT=/dev/ttyUSB0
```

1.7 Tools

You can use the scripts under the `tool` directory to simplify your development process.

You can build and flash with `gen_misc.sh`, which adds timestamp and log saving functions to `make monitor`:

```
cp $MDF_PATH/tools/gen_misc.sh .
./gen_misc.sh /dev/ttyUSB0
```

`multi_downloads.sh` and `multi_downloads.sh` can be used to simultaneously flash multiple devices:

```
cp $MDF_PATH/tools/multi_*.sh .
./multi_downloads.sh 49
./multi_open_serials.sh 49
```

`format.sh` can be used to format the codes:

```
$MDF_PATH/tools/format.sh .
```

CHAPTER 2

API Reference

2.1 Mcommmon API

`Mcommon` (Mesh common) is a module shared by all ESP-MDF components, and it features the followings:

1. Memory Management: manages memory allocation and release, and can help find a memory leak;
2. Error Codes: checks the error codes of all the modules, and therefore can help find out what could possibly go wrong;
3. Event Loop: uses an identical function for all the modules to deal with an event.
4. Data persistence: provide an API to save any type of data on Flash.

2.1.1 Memory Management

Header File

- mcommon/include/mdf_mem.h

Functions

void **`mdf_mem_add_record`**(void *_ptr_, int _size_, **`const`** char *_tag_, int _line_)

Add to memory record.

Parameters

- `ptr`: Memory pointer
- `size`: Memory size
- `tag`: Description tag
- `line`: Line number

void **mdf_mem_remove_record**(void *_ptr_, **const** char *_tag_, int _line_)
Remove from memory record.

Parameters

- `ptr`: Memory pointer
- `tag`: Description tag
- `line`: Line number

void **mdf_mem_print_record**(void)
Print the all allocation but not released memory.

Attention Must configure CONFIG_MDF_MEM_DEBUG == y annd esp_log_level_set(mdf_mem, ESP_LOG_INFO);

void **mdf_mem_print_heap**(void)
Print memory and free space on the stack.

void **mdf_mem_print_task**(void)
Print the state of tasks in the system.

Macros

MDF_MEM_DEBUG
< _cplusplus CONFIG_MDF_MEM_DEBUG

CONFIG_MDF_MEM_DBG_INFO_MAX
CONFIG_MDF_MEM_DBG_INFO_MAX

MDF_MEM_DBG_INFO_MAX

MALLOC_CAP_INDICATE

MDF_MALLOC(size)
Malloc memory.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `size`: Memory size

MDF_CALLOC(n, size)
Calloc memory.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `n`: Number of block
- `size`: Block memory size

MDF_REALLOC(ptr, size)
Reallocate memory.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `ptr`: Memory pointer
- `size`: Block memory size

MDF_REALLOC_RETRY(ptr, size)
Reallocate memory, If it fails, it will retry until it succeeds.

Return

- valid pointer on success
- NULL when any errors

Parameters

- `ptr`: Memory pointer
- `size`: Block memory size

MDF_FREE(ptr)
Free memory.

Parameters

- `ptr`: Memory pointer_cplusplus **MDF_MEM_H**

2.1.2 Error Codes

Header File

- mcommon/include/mdf_err.h

Functions

const char ***mdf_err_to_name**(*mdf_err_t code*)
Returns string for mdf_err_t error codes.

This function finds the error code in a pre-generated lookup-table and returns its string representation.

The function is generated by the Python script tools/gen_mdf_err_to_name.py which should be run each time an mdf_err_t error is modified, created or removed from the IDF project.

Return string error message

Parameters

- `code`: mdf_err_t error code

Macros

MDF_OK
mdf_err_t value indicating success (no error)

MDF_FAIL
Generic mdf_err_t code indicating failure

MDF_ERR_NO_MEM
Out of memory

MDF_ERR_INVALID_ARG
Invalid argument

MDF_ERR_INVALID_STATE
Invalid state

MDF_ERR_INVALID_SIZE
Invalid size

MDF_ERR_NOT_FOUND
Requested resource not found

MDF_ERR_NOT_SUPPORTED
Operation or feature not supported

MDF_ERR_TIMEOUT
Operation timed out

MDF_ERR_INVALID_RESPONSE
Received response was invalid

MDF_ERR_INVALID_CRC
CRC or checksum was invalid

MDF_ERR_INVALID_VERSION
Version was invalid

MDF_ERR_INVALID_MAC
MAC address was invalid

MDF_ERR_NOT_INIT
MAC address was invalid

MDF_ERR_BUF
The buffer is too small

MDF_ERR_MWIFI_BASE
Starting number of MWIFI error codes

MDF_ERR_MESPNOW_BASE
Starting number of MESPNOW error codes

MDF_ERR_MCONFIG_BASE
Starting number of MCONFIG error codes

MDF_ERR_MUPGRADE_BASE
Starting number of MUPGRADE error codes

MDF_ERR_MDEBUG_BASE
Starting number of MDEBUG error codes

MDF_ERR_MLINK_BASE
Starting number of MLINK error codes

MDF_ERR_CUSTOM_BASE
Starting number of COUSTOM error codes

CONFIG_MDF_LOG_LEVEL
CONFIG_MDF_LOG_LEVEL

MDF_LOG_LEVEL

MDF_LOG_FORMAT(letter, format)

MDF_LOGE(format, ...)

MDF_LOGW(format, ...)

MDF_LOGI(format, ...)

MDF_LOGD(format, ...)

MDF_LOGV(format, ...)

MDF_ERROR_CHECK(con, err, format, ...)
Macro which can be used to check the error code, and terminate the program in case the code is not MDF_OK. Prints the error code, error location, and the failed statement to serial output.

Disabled if assertions are disabled.

MDF_ERROR_ASSERT(err)
Macro serves similar purpose as `assert`, except that it checks `esp_err_t` value rather than a `bool` condition. If the argument of `MDF_ERROR_ASSERT` is not equal `MDF_OK`, then an error message is printed on the console, and `abort()` is called.

Note If `IDF monitor` is used, addresses in the backtrace will be converted to file names and line numbers.

Return [description]

Parameters

- `err`: [description]

MDF_ERROR_GOTO(con, lable, format, ...)

MDF_ERROR_CONTINUE(con, format, ...)

MDF_ERROR_BREAK(con, format, ...)

MDF_PARAM_CHECK(con)
_cplusplus **MDF_ERR_H**

Type Definitions

typedef int **mdf_err_t**
< _cplusplus

2.1.3 Event Loop

Header File

- mcommon/include/mdf_event_loop.h

Functions

mdf_err_t **mdf_event_loop_init** (*mdf_event_loop_cb_t cb*)
Initialize event loop, create the event handler and task.

Attention Because all the callbacks are dispatched from the same task, it is recommended to only do the minimal possible amount of work from the callback itself, posting an event to a lower priority task using a queue instead.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `cb`: Application specified event callback, it can be modified by call mdf_event_loop_set

mdf_err_t **mdf_event_loop_deinit** ()
Deinitialize event loop, delete the event handler and task.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdf_event_loop** (*mdf_event_loop_t event*, void **ctx*)
Call event callback function directly.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Event type defined in this file
- `ctx`: Reserved for user

mdf_event_loop_cb_t **mdf_event_loop_set** (*mdf_event_loop_cb_t cb*)
Set event loop callback function.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `cb`: Set application event callback

mdf_err_t **mdf_event_loop_send** (*mdf_event_loop_t event*, void **ctx*)
Send the event to the event handler.

Return

- MDF_OK

- MDF_FAIL

Parameters

- `event`: Generated events
- `ctx`: Reserved for user

mdf_err_t **mdf_event_loop_delay_send**(*mdf_event_loop_t event*, void **ctx*, TickType_t *delay_ticks*)
Delay send the event to the event handler.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Generated events
- `ctx`: Reserved for user
- `delay_ticks`: Delay time

Macros

CONFIG_EVENT_QUEUE_NUM
CONFIG_EVENT_QUEUE_NUM

EVENT_QUEUE_NUM

CONFIG_MDF_EVENT_TASK_NAME
CONFIG_MDF_EVENT_TASK_NAME

MDF_EVENT_TASK_NAME

CONFIG_MDF_EVENT_TASK_STACK_SIZE
CONFIG_MDF_EVENT_TASK_STACK

MDF_EVENT_TASK_STACK

CONFIG_MDF_EVENT_TASK_PRIOTY
CONFIG_MDF_EVENT_TASK_PRIOTY

MDF_EVENT_TASK_PRIOTY

MDF_EVENT_MWIFI_BASE

MDF_EVENT_MESPNOW_BASE

MDF_EVENT_MCONFIG_BASE

MDF_EVENT_MUPGRADE_BASE

MDF_EVENT_MDEBUG_BASE

MDF_EVENT_MLINK_BASE

MDF_EVENT_CUSTOM_BASE

Type Definitions

typedef uint32_t **mdf_event_loop_t**

typedef *mdf_err_t* (***mdf_event_loop_cb_t**) (*mdf_event_loop_t* event, void *ctx)
Application specified event callback function.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `event`: Event type defined in this file
- `ctx`: Reserved for user

2.1.4 Info Store

Header File

- mcommon/include/mdf_info_store.h

Functions

esp_err_t **mdf_info_init** (void)
Initialize the default NVS partition.

Return

- ESP_FAIL
- ESP_OK

esp_err_t **mdf_info_save** (**const** char *_key_, **const** void *_value_, size_t _length_)
save the information with given key

Return

- ESP_FAIL
- ESP_OK

Parameters

- `key`: Key name. Maximal length is 15 characters. Shouldn't be empty.
- `value`: The value to set.
- `length`: length of binary value to set, in bytes; Maximum length is 1984 bytes (508000 bytes or (97.6% of the partition size - 4000) bytes whichever is lower, in case multi-page blob support is enabled).

esp_err_t **__mdf_info_load** (**const** char *_key_, void *_value_, size_t _len_, uint32_t _type_)

esp_err_t **mdf_info_erase** (**const** char *_key_)

Macros

MDF_SPACE_NAME

LENGTH_TYPE_NUMBER

Load the information, esp_err_t mdf_info_load(const char *key, void *value, size_t *length); esp_err_t mdf_info_load(const char *key, void *value, size_t length);.

Attention The interface of this api supports size_t and size_t * types. When the length parameter of the pass is size_t, it is only the length of the value. When the length parameter of the pass is size_t *, the length of the saved information can be obtained.

Return

- ESP_FAIL
- ESP_OK

Parameters

- `key`: The corresponding key of the information that want to load
- `value`: The corresponding value of key
- `length`: The length of the value, Pointer type will return length

LENGTH_TYPE_POINTER

mdf_info_load(key, value, len)

2.2 Mconfig API

`Mconfig` (Mesh Network Configuration) is a network configuration solution for ESP-WIFI-MESH, which sends network configuration information to ESP-WIFI-MESH devices in a convenient and efficient manner.

2.2.1 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mconfig, which includes:

- Connect to the external network: This can be achieved by the root node via MQTT and HTTP.

2.2.2 Mconfig Blufi

Header File

- mconfig/include/mconfig_blufi.h

Functions

mdf_err_t **mconfig_blufi_init**(**const** *mconfig_blufi_config_t* **config*)

initialize Bluetooth network configuratxion

Attention The BLE stack must be enabled first through menuconfig configuration.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

mdf_err_t **mconfig_blufi_deinit** (void)
de-initialize Bluetooth network configuration

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_blufi_send** (uint8_t **data*, size_t *size*)
This function is called to custom data, send a custom request to the APP to verify the device.

Return

- MDF_OK
- MDF_FAIL
- MDF_ERR_INVALID_ARG__MCONFIG_BLUFI_H__

Parameters

- `data`: Custom data value
- `size`: The length of custom data

Structures

struct mconfig_blufi_config_t
Bluetooth configuration network related configuration.

Public Members

char **name**[**MCONFIG_BLUFI_NAME_SIZE**]
Local device & peripheral name, If the length of name is greater than 10 bytes, it will overwrite custom_data, and custom_data will not be available.

uint16_t **company_id**
Company Identifiers (https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers)

uint16_t **tid**
Type of device

uint8_t **custom_size**
Custom data size

uint8_t **custom_data**[**MCONFIG_BLUFI_CUSTOM_SIZE**]
Placed in a Bluetooth broadcast package

bool **only_beacon**
Send only beacon does not support connection

struct mconfig_blufi_data_t
Mconfig_blufi event callback parameters.

Public Members

uint8_t *__data__
Custom data value

size_t __size__
The length of custom data

Macros

__MCONFIG_BLUFI_NAME_SIZE__
Contains the ending character

__MCONFIG_BLUFI_CUSTOM_SIZE__
BLE broadcast data packets have a valid length of up to 31 bytes

__CONFIG_BLUFI_BROADCAST_OUI__
Used to filter other Bluetooth broadcast packets, 3 bytes CONFIG_BLUFI_FILTER_OUI

__MDF_EVENT_MCONFIG_BLUFI_STARTED__

__MDF_EVENT_MCONFIG_BLUFI_STOPED__

__MDF_EVENT_MCONFIG_BLUFI_CONNECTED__

__MDF_EVENT_MCONFIG_BLUFI_DISCONNECTED__

__MDF_EVENT_MCONFIG_BLUFI_STA_CONNECTED__

__MDF_EVENT_MCONFIG_BLUFI_STA_DISCONNECTED__

__MDF_EVENT_MCONFIG_BLUFI_FINISH__

__MDF_EVENT_MCONFIG_BLUFI_RECV__

2.2.3 Mconfig Chain

Header File

- mconfig/include/mconfig_chain.h

Functions

mdf_err_t __mconfig_chain_slave_init__(void)
Chain configuration network slave initialization for obtaining network configuration information.

Attention The received network configuration information is sent to mconfig_queue

Return

- MDF_OK
- MDF_FAIL

mdf_err_t __mconfig_chain_slave_channel_switch_disable__(void)
Disable slave to switch wifi channel.

Attention Chain configuration network slaves, will constantly switch channels to find the master, if you need to make a wifi connection, You must first disable the switch of the slave wifi channel.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_chain_slave_channel_switch_enable**(void)
Enable slave to switch wifi channel.

Attention The slave will continuously switch the wifi channel for scanning, which is enabled by default.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mconfig_chain_slave_deinit**(void)
Free all resource allocated in mconfig_chain_slave_init and stop chain_slave task.

Return

- MDF_OK
- MDF_ERR_NOT_SUPPORTED

mdf_err_t **mconfig_chain_master**(**const** *mconfig_data_t* **config*, TickType_t *duration_ticks*)
Chain configuration network host initialization, sending network configuration information to the slave.

Return

- MDF_OK
- MDF_ERR_NOT_SUPPORTED

Parameters

- `config`: This configuration information will be sent to the slave device.
- `duration_ticks`: Stop chain_master after the set time

mdf_err_t **mconfig_chain_filter_rssi**(int8_t *rssi*)
Chain Devices with weak rssi are not allowed to join the network.

Note If you use it, you must call after `mconfig_chain_master`

Return

- MDF_OK
- MDF_FAIL__MCONFIG_CHAIN_H__

Parameters

- `rssi`: When the device signal strength is less than this value, it will not join the network.

Macros

MDF_EVENT_MCONFIG_CHAIN_SLAVE_STARTED

MDF_EVENT_MCONFIG_CHAIN_SLAVE_STOPED

MDF_EVENT_MCONFIG_CHAIN_FINISH

MDF_EVENT_MCONFIG_CHAIN_MASTER_STARTED

MDF_EVENT_MCONFIG_CHAIN_MASTER_STOPED

MDF_EVENT_MCONFIG_CHAIN_FOUND_ROUTER

2.2.4 Mconfig Queue

Header File

- mconfig/include/mconfig_queue.h

Functions

mdf_err_t **mconfig_queue_write**(**const** *mconfig_data_t* **mconfig_data*, TickType_t *wait_ticks*)
Write data to the queue of mconfig.

Return

- MDF_OK
- MDF_ERR_TIMEOUT

Parameters

- `mconfig_data`: information that points to the configuration of the network
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mconfig_queue_read**(*mconfig_data_t* ***mconfig_data*, TickType_t *wait_ticks*)
READ data to the queue of mconfig.

Return

- MDF_OK
- MDF_ERR_TIMEOUT__MCONFIG_QUEUE_H__

Parameters

- `mconfig_data`: mconfig_data is a secondary pointer and must be called after MDF_FREE is used
- `wait_ticks`: wait time if a packet isn't immediately available

Structures

struct mconfig_whitelist_t
List of configured networks for each device.

Public Members

uint8_t **addr**[**MWIFI_ADDR_LEN**]
the address of the device

struct mconfig_data_t
Network configuration information.

Public Members

mwifi_config_t **config**
Mwifi AP configuration

mwifi_init_config_t **init_config**
Mwifi initialization configuration, Used only during debugging

uint8_t **custom**[32 + **CONFIG_MCONFIG_CUSTOM_EXTERN_LEN**]
Custom data for specific applications, such as: uuid, token, username, etc

uint16_t **whitelist_size**
The size of the device's whitelist

mconfig_whitelist_t **whitelist_data**[0]
Whitelist of devices

2.2.5 Mconfig Security

Header File

- mconfig/include/mconfig_security.h

Functions

mdf_err_t **mconfig_random**(void **rng_state*, uint8_t **output*, size_t *len*)
Generate an array of random numbers.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG

Parameters

- `rng_state`: The seed of the random number, not used temporarily, you can pass NULL
- `output`: Pointer to an array of random numbers
- `len`: The length of the array of random numbers

mdf_err_t **mconfig_dhm_gen_key**(uint8_t **param*, ssize_t *param_size*, uint8_t **privkey*, uint8_t **pubkey*)
Generate a public and private key using the DHM algorithm.

Return

- MDF_OK

- MDF_ERR_INVALID_ARG

Parameters

- `param`: DHM configuration parameters
- `param_size`: The length of the DHM configuration parameter
- `privkey`: DHM's public key
- `pubkey`: DHM's private key

mdf_err_t **mconfig_rsa_gen_key**(char **privkey_pem*, char **pubkey_pem*)

Generate a public and private key using the RSA algorithm.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `privkey_pem`: RSA public key in pem format
- `pubkey_pem`: RSA private key in pem format

mdf_err_t **mconfig_rsa_decrypt**(**const** uint8_t **ciphertext*, **const** char **privkey_pem*, void **plaintext*, size_t *plaintext_size*)

Use RSA's public key to encrypt the data.

Return

- ESP_OK
- ESP_FAIL
- MDF_ERR_INVALID_ARG

Parameters

- `ciphertext`: Encrypted data
- `privkey_pem`: RSA's private key in pem format
- `plaintext`: Unencrypted data
- `plaintext_size`: The length of unencrypted data

mdf_err_t **mconfig_rsa_encrypt**(**const** void **plaintext*, size_t *plaintext_size*, **const** char **pubkey_pem*, uint8_t **ciphertext*)

use RSA's public key to encrypt the data

Return

- ESP_OK
- ESP_FAIL
- MDF_ERR_INVALID_ARG__MCONFIG_SECURITY_H__

Parameters

- `plaintext`: Unencrypted data
- `plaintext_size`: The length of unencrypted data

- `pubkey_pem`: RSA's public key in pem format
- `ciphertext`: Encrypted data

Macros

MCONFIG_RSA_PRIVKEY_PEM_SIZE

MCONFIG_RSA_PUBKEY_PEM_SIZE

MCONFIG_RSA_KEY_BITS

MCONFIG_RSA_CIPHERTEXT_SIZE

MCONFIG_RSA_PLAINTEXT_MAX_SIZE

MCONFIG_RSA_EXPONENT

MCONFIG_RSA_PUBKEY_PEM_DATA_SIZE

PEM_BEGIN_PUBLIC_KEY

PEM_END_PUBLIC_KEY

PEM_BEGIN_PRIVATE_KEY

PEM_END_PRIVATE_KEY

MCONFIG_DH_PRIVKEY_LEN

MCONFIG_DH_PUBKEY_LEN

MCONFIG_AES_KEY_LEN

2.3 Mespnow API

`Mespnow` (Mesh ESP-NOW) is the encapsulation of ESP-NOW APIs, and it adds to ESP-NOW the retransmission filter, Cyclic Redundancy Check (CRC), and data fragmentation features.

2.3.1 Features

1. **Retransmission filter**: Mespnow adds a 16-bit ID to each fragment, and the redundant fragments with the same ID will be discarded.
2. **Fragmented transmission**: When the data packet exceeds the limit of the maximum packet size, Mespnow splits it into fragments before they are transmitted to the target device for reassembly.
3. **Cyclic Redundancy Check**: Mespnow implements CRC when it receives the data packet to ensure the packet is transmitted correctly.

2.3.2 Writing Applications

1. Prior to the use of Mwifi, Wi-Fi must be initialized;
2. Max number of the devices allowed in the network: No more than 20 devices, among which 6 encrypted devices at most, are allowed to be network configured;
3. Security: CCMP is used for encrpytion with a 16-byte key.

2.3.3 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mespnow, which includes:

- A simple application that demonstrates how to use Mespnow for the communication between two devices.

2.3.4 API Reference

Header File

- mespnow/include/mespnow.h

Functions

mdf_err_t **mespnow_add_peer**(wifi_interface_t *ifx*, **const** uint8_t **addr*, **const** uint8_t **lmk*)
add a peer to espnow peer list based on esp_now_add_peer(...). It is convenient to use simplified MACRO follows.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `ifx`: Wi-Fi interface that peer uses to send/receive ESPNOW data
- `addr`: peer mac address
- `lmk`: local master key. If lmk is NULL, ESPNOW data that this peer sends/receives isn't encrypted

mdf_err_t **mespnow_del_peer**(**const** uint8_t **addr*)
delete a peer from espnow peer list.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `addr`: peer mac address

mdf_err_t **mespnow_read**(*mespnow_trans_pipe_e pipe*, uint8_t **src_addr*, void **data*, size_t **size*, TickType_t *wait_ticks*)
read data from espnow

Return

- ESP_OK
- ESP_FAIL

Parameters

- `pipe`: mespnow packet type
- `src_addr`: source address

- `data`: point to received data buffer
- `*size`: A non-zero pointer to the variable holding the length of out_value. In case out_value is not zero, will be set to the actual length of the value written.
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mespnow_write**(*mespnow_trans_pipe_e pipe*, **const** uint8_t **dest_addr*, **const** void **data*, size_t *size*, TickType_t *wait_ticks*)

write date package to espnow.

1. It is necessary to add device to espnow_peer befor send data to dest_addr.
2. When data_len to write is too long, it may fail duration some package and and the return value is the data len that actually sended.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `pipe`: Pipe of data from espnnow
- `dest_addr`: Destination address
- `data`: Point to send data buffer
- `size`: send data len
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mespnow_deinit**(void)

deinit mespnow

Return

- ESP_OK
- ESP_FAIL

mdf_err_t **mespnow_init**(void)

init mespnow

Return

- ESP_OK
- ESP_FAIL_cplusplus **MESPNOW_H**

Macros

MESPNOW_PAYLOAD_LEN

< _cplusplus

MDF_EVENT_MESPNOW_RECV

MDF_EVENT_MESPNOW_SEND

Enumerations

enum mespnow_trans_pipe_e
Divide espnnow data into multiple pipes.

Values:

MESPNOW_TRANS_PIPE_DEBUG
debug packet: log, coredump, espnow_debug config, ack

MESPNOW_TRANS_PIPE_CONTROL
control packet

MESPNOW_TRANS_PIPE_MCONFIG
network configuration packet

MESPNOW_TRANS_PIPE_RESERVED
reserved for other functiond

MESPNOW_TRANS_PIPE_MAX

2.4 Mlink API

`Mlink` (MESH LAN communication protocol) is a solution for controlling ESP-WIFI-MESH network devices through APP, including: device discovery, control, upgrade, etc.

2.4.1 Application Examples

For Mlink examples, please refer to the directory development_kit, which includes:

- Smart light: Mesh App can control the color of the lamp, distribution network, upgrade, etc.

2.4.2 Mlink

Header File

- mlink/include/mlink.h

Macros

MDF_EVENT_MLINK_SYSTEM_RESET
< _cplusplus

MDF_EVENT_MLINK_SYSTEM_REBOOT

MDF_EVENT_MLINK_SET_STATUS

MDF_EVENT_MLINK_GET_STATUS

MDF_EVENT_MLINK_SET_TRIGGER

MDF_EVENT_MLINK_BUFFER_FULL

Enumerations

enum mlink_protocol
Type of protocol.

Values:

MLINK_PROTO_HTTPD = 0
Http protocol communication

MLINK_PROTO_NOTICE = 1
UDP protocol communication

2.4.3 Mlink Handle

Header File

- mlink/include/mlink_handle.h

Functions

mdf_err_t **mlink_add_device**(uint32_t *tid*, **const** char ***name*, **const** char ***version*)
Configuring basic information about the device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `tid`: Unique identifier for the device type
- `name`: The name of device
- `version`: The version of device

mdf_err_t **mlink_device_set_name**(**const** char ***name*)
Set device name.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `name`: The name of device

mdf_err_t **mlink_device_set_position**(**const** char ***position*)
Set device version.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `position`: The position of device

const char ***mlink_device_get_name**()
Get device name.

Return Name of the device

const char ***mlink_device_get_position**()
Get device version.

Return Version of the device

int **mlink_device_get_tid**()
Get device tid.

Return Unique identifier for the device type

mdf_err_t **mlink_add_characteristic**(uint16_t *cid*, **const** char **name*, *characteristic_format_t format*, *characteristic_perms_t perms*, int *min*, int *max*, uint16_t *step*)
Add device characteristic information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cid`: Unique identifier for the characteristic
- `name`: The name of the characteristic
- `format`: The format of the characteristic
- `perms`: The permissions of the characteristic
- `min`: The min of the characteristic
- `max`: The max of the characteristic
- `step`: The step of the characteristic

mdf_err_t **mlink_add_characteristic_handle**(*mlink_characteristic_func_t get_value_func*, *mlink_characteristic_func_t set_value_func*)
Increase the device's characteristic handler.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `get_value_func`: [description]
- `set_value_func`: [description]

mdf_err_t **mlink_handle**(**const** uint8_t **src_addr*, **const** *mlink_httpd_type_t* **type*, **const** void **data*, size_t *size*)

Handling requests from the APP.

Note This function is deprecated. Use 'mlink_handle_request' function

Return

- ESP_OK
- ESP_FAIL

Parameters

- `src_addr`: Source address of the device
- `type`: Type of data
- `data`: Requested message
- `size`: The length of the requested data

mdf_err_t **mlink_set_handle**(**const** char **name*, **const** *mlink_handle_func_t* *func*)

Add or modify a request handler.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `name`: The name of the handler
- `func`: The pointer of the handler

mdf_err_t **mlink_handle_request**(*mlink_handle_data_t* **handle_data*)

Call the handler in the request list.

Return

- ESP_OK
- ESP_FAIL
- MDF_ERR_NOT_SUPPORTED_cplusplus **MLINK_HANDLE_H**

Parameters

- `handle_data`: The data type of the parameter of the handler

Structures

struct mlink_handle_data_t

The data type of the parameter of the handler.

Public Members

const char ***req_data**
Received request data

ssize_t **req_size**
The length of the received request data

mlink_httpd_format_t **req_fromat**
The format of the received request data

char ***resp_data**
Response data to be sent

ssize_t **resp_size**
The length of response data to be sent

mlink_httpd_format_t **resp_fromat**
The format of response data to be sent

Type Definitions

typedef *mdf_err_t* (***mlink_handle_func_t**) (*mlink_handle_data_t* *data)
Type of request handler.

typedef *mdf_err_t* (***mlink_characteristic_func_t**) (uint16_t cid, void *value)
Get the type of callback function that sets the characteristic value.

Enumerations

enum characteristic_perms_t
Permissions for the characteristics.

< _cplusplus

Values:

CHARACTERISTIC_PERMS_READ = 1 << 0
The characteristic of the device are readable

CHARACTERISTIC_PERMS_WRITE = 1 << 1
The characteristic of the device are writable

CHARACTERISTIC_PERMS_TRIGGER = 1 << 2
The characteristic of the device can be triggered

CHARACTERISTIC_PERMS_RW = CHARACTERISTIC_PERMS_READ | CHARACTERISTIC_PERMS_WRITE
The characteristic of the device are readable & writable

CHARACTERISTIC_PERMS_RT = CHARACTERISTIC_PERMS_READ | CHARACTERISTIC_PERMS_TRIGGER
The characteristic of the device are readable & triggered

CHARACTERISTIC_PERMS_WT = CHARACTERISTIC_PERMS_WRITE | CHARACTERISTIC_PERMS_TRIGGER
The characteristic of the device are writable & triggered

CHARACTERISTIC_PERMS_RWT = CHARACTERISTIC_PERMS_RW | CHARACTERISTIC_PERMS_TRIGGER
The characteristic of the device are readable & writable & triggered

enum characteristic_format_t
Format for the characteristic.

Values:

CHARACTERISTIC_FORMAT_NONE
Invalid format

CHARACTERISTIC_FORMAT_INT
characteristic is a number format

CHARACTERISTIC_FORMAT_DOUBLE
characteristic is a double format

CHARACTERISTIC_FORMAT_STRING
characteristic is a string format

2.4.4 Mlink Httpd

Header File

- mlink/include/mlink_httpd.h

Functions

mdf_err_t **mlink_httpd_start**(void)
Start http server.

Return

- ESP_OK
- ESP_FAIL

mdf_err_t **mlink_httpd_stop**(void)
Stop http server.

Return

- ESP_OK
- ESP_FAIL

mdf_err_t **mlink_httpd_read**(*mlink_httpd_t* ***request*, TickType_t *wait_ticks*)
Receive data from http server.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `request`: Request data
- `wait_ticks`: Waiting timeout

mdf_err_t **mlink_httpd_write**(**const** *mlink_httpd_t* **response*, TickType_t *wait_ticks*)
Send data to http server.

Return

- ESP_OK
- ESP_FAIL_cplusplus **MLINK_HTTPD_H**

Parameters

- `response`: Response data
- `wait_ticks`: Waiting timeout

Structures

struct mlink_httpd_type_t
Http server data type.

Public Members

uint16_t **sockfd**
File descriptor for tcp

uint8_t **format**
Http body data format

uint8_t **from**
Data request source

bool **resp**
Whether to respond to request data

uint16_t **received**
Received

struct mlink_httpd_t
Http server data.

Public Members

mlink_httpd_type_t **type**
Http server data type

bool **group**
Send a package as a group

size_t **addrs_num**
Number of addresses

uint8_t ***addrs_list**
List of addresses

size_t **size**
Length of data

char ***data**
Pointer of Data

Enumerations

enum mlink_httpd_from_t
Data request source.

< _cplusplus

Values:

MLINK_HTTPD_FROM_DEVICE = 1
Request from within the ESP-WIFI-MESH network

MLINK_HTTPD_FROM_SERVER = 2
Request from server

enum mlink_httpd_format_t
Http body data format.

Values:

MLINK_HTTPD_FORMAT_NONE
Invalid format

MLINK_HTTPD_FORMAT_HEX
data is a hex format

MLINK_HTTPD_FORMAT_JSON
data is a json format

MLINK_HTTPD_FORMAT_HTML
data is a html format

2.4.5 Mlink Notice

Header File

- mlink/include/mlink_notice.h

Functions

mdf_err_t **mlink_notice_init**()
Initialize mlink mdns and udp.

< _cplusplus

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mlink_notice_deinit**()
Deinitialize mlink mdns and udp.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mlink_notice_write**(**const** char *_message_, size_t _size_, **const** uint8_t *_addr_)
Inform Mesh-App to initiate a request.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL_cplusplus **MLINK_NOTICE_H**

Parameters

- `message`: Type of message requested
- `size`: Length of the message
- `addr`: Address of the device to be requested

2.4.6 Mlink Utils

Header File

- mlink/include/mlink_utils.h

Functions

uint8_t *__mlink_mac_str2hex__(**const** char *_mac_str_, uint8_t *_mac_hex_)
Convert mac from string format to hex.

< _cplusplus

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

Parameters

- `mac_str`: String format mac
- `mac_hex`: Hex format mac

char *__mlink_mac_hex2str__(**const** uint8_t *_mac_hex_, char *_mac_str_)
Convert mac from hex format to string.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

Parameters

- `mac_hex`: Hex format mac
- `mac_str`: String format mac

uint8_t *__mlink_mac_ap2sta__(__const__ uint8_t *_ap_mac_, uint8_t *_sta_mac_)
Convert mac address from ap to sta.

Each ESP32 chip has MAC addresses for Station (STA), Access Point (AP), Bluetooth low energy (BT) and Local Area Network (LAN). Their address values are incremented by one, i.e. LAN Mac = BT Mac + 1 = AP Mac + 2 = STA Mac + 3.

For example:

- MAC for STA: `xx:xx:xx:xx:xx:00`
- MAC for AP : `xx:xx:xx:xx:xx:01`
- MAC for BT : `xx:xx:xx:xx:xx:02`
- MAC for LAN: `xx:xx:xx:xx:xx:03`

The device's STA address is used For ESP-WIFI-MESH communication.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL

Parameters

- `ap_mac`: Access Point address in hexadecimal format
- `sta_mac`: Station address in hexadecimal format

uint8_t *__mlink_mac_bt2sta__(__const__ uint8_t *_bt_mac_, uint8_t *_sta_mac_)
Convert mac address from bt to sta.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_FAIL_cplusplus **MLINK_UTILS_H**

Parameters

- `bt_mac`: Access Point address in hexadecimal format
- `sta_mac`: Station address in hexadecimal format

2.4.7 Mlink Json

Header File

- mlink/include/mlink_json.h

Functions

esp_err_t ___mlink_json_parse__(__const__ char *_json_str_, __const__ char *_key_, void *_value_, int _value_type_)
esp_err_t mlink_json_parse(const char *json_str, const char *key, void *value) Parse the json formatted string

Note does not support the analysis of array types

Return

- ESP_OK
- ESP_FAIL

Parameters

- `json_str`: The string pointer to be parsed
- `key`: Build value pairs
- `value`: You must ensure that the incoming type is consistent with the post-resolution type
- `value_type`: Type of parameter

esp_err_t **__mlink_json_pack**(char **_json_str_, **const** char *_key_, int _value_, int _value_type_)
mlink_json_pack(char *json_str, const char *key, int/double/char value); Create a json string

Note if the value is double or float type only retains the integer part, requires complete data calling mlink_json_pack_double()

Return

- lenght: generates the length of the json string
- ESP_FAIL

Parameters

- `json_str`: Save the generated json string
- `key`: Build value pairs
- `value`: This is a generic, support long / int / float / char / char* / char []
- `value_type`: Type of parameter

ssize_t **mlink_json_pack_double**(char **_json_ptr_, **const** char *_key_, double _value_)
Create a double type json string, Make up for the lack of mdf_json_pack()

Return

- lenght: generates the length of the json string
- ESP_FAIL_cplusplus **MLINK_JOSN_H**

Parameters

- `json_ptr`: Save the generated json string
- `key`: Build value pairs
- `value`: The value to be stored

Macros

mlink_json_parse(json_str, key, value)

mlink_json_pack(json_str, key, value)

Enumerations

enum mlink_json_type
The type of the parameter.

< _cplusplus

Values:

MLINK_JSON_TYPE_NONE = 0
Invalid parameter type

MLINK_JSON_TYPE_INT8 = 1
The type of the parameter is int8_t / uint8_t

MLINK_JSON_TYPE_INT16 = 10
The type of the parameter is int16_t / uint16_t

MLINK_JSON_TYPE_INT32 = 100
The type of the parameter is int32_t / uint32_t

MLINK_JSON_TYPE_FLOAT = 1000
The type of the parameter is float

MLINK_JSON_TYPE_DOUBLE = 10000
The type of the parameter is double

MLINK_JSON_TYPE_STRING = 100000
The type of the parameter is string

MLINK_JSON_TYPE_POINTER = 1000000
The type of the parameter is point

2.5 Mupgrade API

`Mupgrade` (Mesh Upgrade) is a solution for simultaneous over-the-air (OTA) upgrading of multiple ESP-WIFI-MESH devices on the same wireless network by efficient routing of data flows. It features automatic retransmission of failed fragments, data compression, reverting to an earlier version, firmware check, etc.

2.5.1 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mupgrade, which includes:

- Send a request for data to server via AP, in order to complete the upgrading of the overall Mesh network.

2.5.2 API Reference

Header File

- mupgrade/include/mupgrade.h

Functions

mdf_err_t **mupgrade_firmware_init**(**const** char ***name*, size_t *size*)
Initialize the upgrade status and erase the upgrade partition.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_ERR_MUPGRADE_FIRMWARE_PARTITION

Parameters

- `name`: Unique identifier of the firmware
- `size`: Total length of firmware, If image size is not yet known, pass OTA_SIZE_UNKNOWN and call `mupgrade_firmware_download_finished()` after the firmware download is complete.

mdf_err_t **mupgrade_firmware_download**(**const** void ***data*, size_t *size*)
Write firmware to flash.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT
- MDF_ERR_MUPGRADE_FIRMWARE_FINISH

Parameters

- `data`: Pointer to the firmware
- `size`: The length of the data

mdf_err_t **mupgrade_firmware_download_finished**(size_t *image_size*)
Finish OTA update and validate newly written app image.

Attention Only called `mupgrade_firmware_init()` firmware size unknown at root node

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT

Parameters

- `image_size`: of new OTA app image

mdf_err_t **mupgrade_firmware_check**(**const** esp_partition_t ***partition*)
Check if the firmware is generated by this project.

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_INVALID

Parameters

- `partition`: The partition where the firmware to be checked is located

mdf_err_t **mupgrade_firmware_send**(**const** uint8_t **dest_addrs*, size_t *dest_addrs_num*, *mupgrade_result_t* **result*)

Root sends firmware to other nodes.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT
- MDF_ERR_MUPGRADE_DEVICE_NO_EXIST

Parameters

- `dest_addrs`: Destination nodes of mac
- `dest_addrs_num`: Number of destination nodes
- `result`: Must call mupgrade_result_free to free memory

mdf_err_t **mupgrade_firmware_stop**()

Stop Root to send firmware to other nodes.

- MDF_ERR_NOT_SUPPORTED

Return

- MDF_OK

mdf_err_t **mupgrade_result_free**(*mupgrade_result_t* **result*)

Free memory in the results list.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG

Parameters

- `result`: Pointer to device upgrade status

mdf_err_t **mupgrade_handle**(**const** uint8_t **addr*, **const** void **data*, size_t *size*)

Handling upgrade data sent by ROOT.

Attention Only called at the non-root

Return

- MDF_OK
- MDF_ERR_MUPGRADE_FIRMWARE_DOWNLOAD
- MDF_ERR_MUPGRADE_NOT_INIT

Parameters

- `addr`: root address

- `data`: Upgrade instructions or firmware
- `size`: The length of the data

mdf_err_t **mupgrade_root_handle**(**const** uint8_t **addr*, **const** void **data*, size_t *size*)
Handling the status of non-root responses.

Attention Only called at the root

Return

- MDF_OK
- MDF_ERR_TIMEOUT

Parameters

- `addr`: Non-root address
- `data`: State of non-root response
- `size`: The length of the data

mdf_err_t **mupgrade_get_status**(*mupgrade_status_t *status*)
Get the status of the upgrade.

Return

- MDF_OK
- MDF_ERR_INVALID_ARG
- MDF_ERR_NOT_SUPPORTED

Parameters

- `status`: The status of the upgrade

mdf_err_t **mupgrade_version_fallback**()
Upgrade error back to previous version.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mupgrade_stop**()
Stop upgrading.

Return

- MDF_OK
- MDF_FAIL_cplusplus **MUPGRADE_H**

Structures

struct mupgrade_packet_t
Firmware packet.

Public Members

uint8_t **type**
Type of packet, MUPGRADE_TYPE_DATA

uint16_t **seq**
Sequence

uint16_t **size**
Size

uint8_t **data**[**MUPGRADE_PACKET_MAX_SIZE**]
Firmware

struct mupgrade_status_t
Status packet.

Public Members

uint8_t **type**
Type of packet, MUPGRADE_TYPE_STATUS

char **name**[32]
Unique identifier of the firmware

mdf_err_t **error_code**
Upgrade status

size_t **total_size**
Total length of firmware

size_t **written_size**
The length of the flash has been written

uint8_t **progress_array**[0]
Identify if each packet of data has been written

struct mupgrade_config_t
Mupgrade config.

Public Members

QueueHandle_t **queue**
Queue handle

esp_ota_handle_t **handle**
OTA handle

const esp_partition_t ***partition**
Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get.

uint32_t **start_time**
Start time of the upgrade

mupgrade_status_t **status**
Upgrade status

struct mupgrade_result_t
List of devices' status during the upgrade process.

Public Members

size_t **unfinished_num**
The number of devices to be upgraded

uint8_t ***unfinished_addr**
MAC address of devices to be upgraded

size_t **successed_num**
The number of devices that succeeded to upgrade

uint8_t ***successed_addr**
MAC address of devices that succeeded to upgrade

size_t **requested_num**
The number of devices to be upgraded

uint8_t ***requested_addr**
This address is used to buffer the result of the request during the upgrade process

Macros

MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT
mupgrade error code definition

< _cplusplus Uninitialized firmware configuration

MDF_ERR_MUPGRADE_FIRMWARE_PARTITION
Partition table error

MDF_ERR_MUPGRADE_FIRMWARE_INVALID
Non-project generated firmware

MDF_ERR_MUPGRADE_FIRMWARE_INCOMPLETE
The firmware received by the device is incomplete

MDF_ERR_MUPGRADE_FIRMWARE_DOWNLOAD
Firmware write flash error

MDF_ERR_MUPGRADE_FIRMWARE_FINISH
The firmware has been written to completion

MDF_ERR_MUPGRADE_DEVICE_NO_EXIST
The device that needs to be upgraded does not exist

MDF_ERR_MUPGRADE_SEND_PACKET_LOSS
Request device upgrade status failed

MDF_ERR_MUPGRADE_NOT_INIT
Upgrade configuration is not initialized

MDF_ERR_MUPGRADE_STOP
Upgrade configuration is not initialized

MDF_EVENT_MUPGRADE_STARTED
enumerated list OF MDF event id

The device starts to upgrade

MDF_EVENT_MUPGRADE_STATUS
Proactively report progress

MDF_EVENT_MUPGRADE_FINISH
The upgrade is complete and the new firmware will run after the reboot

MDF_EVENT_MUPGRADE_STOPED
Stop upgrading

MDF_EVENT_MUPGRADE_FIRMWARE_DOWNLOAD
Start firmware write flash

MDF_EVENT_MUPGRADE_SEND_FINISH
Send the firmware to other devices to complete

MUPGRADE_PACKET_MAX_SIZE
Firmware subcontract upgrade.

Maximum length of a single packet transmitted

MUPGRADE_PACKET_MAX_NUM
The maximum number of packets

MUPGRADE_GET_BITS(data, bits)
Bit operations to get and modify a bit in an array.

MUPGRADE_SET_BITS(data, bits)

MUPGRADE_TYPE_DATA
Type of packet.

MUPGRADE_TYPE_STATUS

2.6 Mwifi API

`Mwifi` (Wi-Fi Mesh) is the encapsulation of ESP-WIFI-MESH APIs, and it adds to ESP-WIFI-MESH the retransmission filter, data compression, fragmented transmission, and P2P multicast features.

2.6.1 Features

1. **Retransmission filter**: As ESP-WIFI-MESH won't perform data flow control when transmitting data downstream, there will be redundant fragments in case of unstable network or wireless interference. For this, Mwifi adds a 16-bit ID to each fragment, and the redundant fragments with the same ID will be discarded.
2. **Fragmented transmission**: When the data packet exceeds the limit of the maximum packet size, Mwifi splits it into fragments before they are transmitted to the target device for reassembly.
3. **Data compression**: When the packet of data is in Json and other similar formats, this feature can help reduce the packet size and therefore increase the packet transmitting speed.
4. **P2P multicast**: As the multicasting in ESP-WIFI-MESH may cause packet loss, Mwifi uses a P2P (peer-to-peer) multicasting method to ensure a much more reliable delivery of data packets.

2.6.2 Writing Applications

Mwifi supports using ESP-WIFI-MESH under self-organized mode, where only the root node that serves as a gateway of Mesh network can require the LwIP stack to transmit/receive data to/from an external IP network.

Mwifi also supports the way to fix the root node. In this case, the nodes in the entire ESP-WIFI-MESH network cannot communicate with the external IP network through wifi. If you need to communicate with the external IP network, you need to use other methods.

If you want to call Wi-Fi API to connect/disconnect/scan, please pause Mwifi first. Self organized networking must be disabled before the application calls any Wi-Fi driver APIs.

Initialization

Wi-Fi Initialization

Prior to the use of Mwifi, Wi-Fi must be initialized. `esp_wifi_set_ps` (indicates whether Mesh network enters into sleep mode) and `esp_mesh_set_6m_rate` (indicates the minimum packet transmittig speed) must be set before `esp_wifi_start`.

The following code snippet demonstrates how to initialize Wi-Fi.

```
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();

MDF_ERROR_ASSERT(nvs_flash_init());

tcpip_adapter_init();
MDF_ERROR_ASSERT(esp_event_loop_init(NULL, NULL));
MDF_ERROR_ASSERT(esp_wifi_init(&cfg));
MDF_ERROR_ASSERT(esp_wifi_set_storage(WIFI_STORAGE_RAM));
MDF_ERROR_ASSERT(esp_wifi_set_mode(WIFI_MODE_STA));
MDF_ERROR_ASSERT(esp_wifi_set_ps(WIFI_PS_NONE));
MDF_ERROR_ASSERT(esp_mesh_set_6m_rate(false));
MDF_ERROR_ASSERT(esp_wifi_start());
```

Mwifi Initialization

In general, you don't need to modify Mwifi and just use its default configuration. But if you want to modify it, you may use `make menuconfig`. See the detailed configuration below:

1. ROOT

Parameters	Description
vote_percentage	Vote percentage threshold above which the node becoms a root
vote_max_coun	Max multiple voting each device can have for the self-healing of a MESH network
backoff_rssi	RSSI threshold below which connections to the root node are not allowed
scan_min_count	The minimum number of times a device should scan the beacon frames from other devices before it becomes a root node
root_conflicts_enable	Allow more than one root in one network
root_healing_ms	Time lag between the moment a root node is disconnected from the network and the moment the devices start electing another root node

2. **Capacity**

Parameters	Description
capacity_num	Network capacity, defining max number of devices allowed in the MESH network
max_layer	Max number of allowed layers
max_connection	Max number of MESH softAP connections

3. **Stability**

Parameters	Description
assoc_expire_ms	Period of time after which a MESH softAP breaks its association with inactive leaf nodes
beacon_interval_ms	Mesh softAP beacon interval
passive_scan_ms	Mesh station passive scan duration
monitor_duration_ms	Period (ms) for monitoring the parent's RSSI. If the signal stays weak throughout the period, the node will find another parent offering more stable connection
cnx_rssi	RSSI threshold above which the connection with a parent is considered strong
select_rssi	RSSI threshold for parent selection. Its value should be greater than SWITCH_RSSI
switch_rssi	RSSI threshold below which a node selects a parent with better RSSI
attempt_count	Parent selection fail times, if the scan times reach this value, device will disconnect with associated children and join self-healing
monitor_ie_count	Allowed number of changes a parent node can introduce into its information element (IE), before the leaf nodes must update their own IEs accordingly

4. **Transmission**

Parameters	Description
xon_qsize	Number of MESH buffer queues
retransmit_enable	Enable retransmission on mesh stack
data_drop_enable	If a root is changed, enable the new root to drop the previous packet

The following code snippet demonstrates how to initialize Mwifi.

```
/*  Mwifi initialization */
wifi_init_config_t cfg = MWIFI_INIT_CONFIG_DEFAULT();
MDF_ERROR_ASSERT(mwifi_init(&cfg));
```

Configuration

The modified Mwifi configuration will only be effective after Mwifi is rebooted.

You may configure Mwifi by using the struct mwifi_config_t. See the configuration parameters below:

Parameters	Description
router_ssid	SSID of the router
router_password	Router password
router_bssid	BSSID is equal to the router's MAC address. This field must be configured if more than one router shares the same SSID. You can avoid using BSSIDs by setting up a unique SSID for each router. This field must also be configured if the router is hidden
mesh_id	Mesh network ID. Nodes sharing the same MESH ID can communicate with one another
mesh_password	Password for secure communication between devices in a MESH network
mesh_type	Only MESH_IDLE, MESH_ROOT, and MESH_NODE device types are supported. MESH_ROOT and MESH_NODE are only used for routerless solutions
channel	Mesh network and the router will be on the same channel
channel_switch_disable	If this value is not set, when "attempt" (mwifi_init_config_t) times is reached, device will change to a full channel scan for a network that could join.
router_switch_disable	If the BSSID is specified and this value is not also set, when the router of this specified BSSID fails to be found after "attempt" (mwifi_init_config_t) times, the whole network is allowed to switch to another router with the same SSID.

The following code snippet demonstrates how to configure Mwifi.

```
mwifi_config_t config = {
    .router_ssid     = CONFIG_ROUTER_SSID,
    .router_password = CONFIG_ROUTER_PASSWORD,
    .mesh_id         = CONFIG_MESH_ID,
    .mesh_password   = CONFIG_MESH_PASSWORD,
    .mesh_type       = CONFIG_MESH_TYPE,
    .channel         = CONFIG_ROUTER_CHANNEL,
};

MDF_ERROR_ASSERT(mwifi_set_config(&config));
```

Start Mwifi

After starting Mwifi, the application should check for Mwifi events to determine when it has connected to the network.

1. MDF_EVENT_MWIFI_ROOT_GOT_IP: the root node gets the IP address and can communicate with the external network;
2. MDF_EVENT_MWIFI_PARENT_CONNECTED: connects with the parent node for data communication between devices.

```
MDF_ERROR_ASSERT(mdf_event_loop_init(event_loop_cb));
MDF_ERROR_ASSERT(mwifi_start());
```

2.6.3 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/mwifi, which includes:

- Connect to the external network: This can be achieved by the root node via TCP.
- Routerless: It involves a fixed root node which communicates with the external devices via UART.
- MQTT example: Implement each node to communicate with the MQTT broker by the root node via MQTT.
- Channel switching: When the router channel changes, Mesh switches its channel accordingly.(In development)

- Low power consumption: The leaf nodes stop functioning as softAP and enter into sleep mode.(In development)

2.6.4 API Reference

Header File

- mwifi/include/mwifi.h

Functions

esp_err_t **esp_wifi_vnd_mesh_get**(mesh_assoc_t *mesh_assoc, mesh_chain_layer_t *mesh_chain)
Get mesh networking IE.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mesh_assoc`: pointer mesh networking IE.
- `mesh_chain`: pointer mesh chain layer infomation.

mdf_err_t **mwifi_init**(**const** *mwifi_init_config_t* *config)
Mwifi initialization.

Attention 1. This API should be called after WiFi is started.

Attention 2. This API must be called before all other Mwifi API can be called.

Attention 3. Always use WIFI_INIT_CONFIG_DEFAULT macro to init the config to default values, this can guarantee all the fields got correct value when more fields are added into wifi_init_config_t in future release. If you want to set your owner initial values, overwrite the default values which are set by WIFI_INIT_CONFIG_DEFAULT.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT
- MDF_ERR_MWIFI_INITED

Parameters

- `config`: Mwifi initialization configuration.

mdf_err_t **mwifi_deinit**(void)
Deinit mwifi Free all resource allocated in mwifi_init and stop Mwifi task.

Attention This API should be called if you want to remove Mwifi from the system.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

mdf_err_t **mwifi_set_init_config**(**const** *mwifi_init_config_t* **init_config*)
Set Mwifi configuration.

Attention This API shall be called between esp_mesh_init() and esp_mesh_start().

Return

- MDF_OK
- MDF_ERR_MWIFI_ARGUMENT

Parameters

- `init_config`: Use MWIFI_INIT_CONFIG_DEFAULT() to initialize the default values.

mdf_err_t **mwifi_get_init_config**(*mwifi_init_config_t* **init_config*)
Get Mwifi init configuration.

Return [description]

Parameters

- `init_config`: pointer to Mwifi init configuration.

mdf_err_t **mwifi_set_config**(**const** *mwifi_config_t* **config*)
Set the configuration of the AP.

Attention This API shall be called between esp_mesh_init() and esp_mesh_start().

Return

- MDF_OK
- MDF_ERR_MWIFI_ARGUMENT

Parameters

- `config`: pointer to AP configuration

mdf_err_t **mwifi_get_config**(*mwifi_config_t* **config*)
Get the configuration of the AP.

Return

- MDF_OK
- MDF_ERR_MWIFI_ARGUMENT

Parameters

- `config`: pointer to AP configuration

mdf_err_t **mwifi_start**(void)
Start Mwifi according to current configuration.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

mdf_err_t **mwifi_stop**(void)
Stop mwifi.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

mdf_err_t **mwifi_restart**()
restart mwifi

Attention This API should be called when the Mwifi configuration is modified.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_INIT

bool **mwifi_is_started**(void)
whether wifi_mesh is running

Return

- true
- flase

bool **mwifi_is_connected**(void)
Whether the node is already connected to the parent node.

Return

- true
- flase

void **mwifi_print_config**()
Print mesh configuration information.

mdf_err_t **mwifi_write**(**const** uint8_t **dest_addrs*, **const** *mwifi_data_type_t* **data_type*, **const** void **data*, size_t *size*, bool *block*)
Send a packet to any node in the mesh network.

Attention 1. If data encryption is enabled, you must ensure that the task that calls this function is greater than 4KB.

1. When sending data to the root node, if the destination address is NULL, the root node receives it using mwifi_root_read(). If the destination address is the mac address of the root node or MWIFI_ADDR_ROOT, the root node receives it using mwifi_read()

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START

Parameters

- `dest_addrs`: The address of the final destination of the packet If the packet is to the root and "dest_addr" parameter is NULL
- `data_type`: The type of the data If the default configuration is used, this parameter is NULL
- `data`: Pointer to a sending wifi mesh packet

- `size`: The length of the data
- `block`: Whether to block waiting for data transmission results

mdf_err_t **__mwifi_read**(uint8_t *src_addr, *mwifi_data_type_t* *data_type, void *data, size_t *size, TickType_t wait_ticks, uint8_t type)

Receive a packet targeted to self over the mesh network.

Attention Judging the allocation of buffers by the type of the parameter 'data' The memory of `data` is externally allocated, and the memory is larger than the total length expected to be received. The memory of `data` is allocated internally by `mwifi_read`, which needs to be released after the call.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `src_addr`: The address of the original source of the packet
- `data_type`: The type of the data
- `data`: Pointer to the received mesh packet To apply for buffer space externally, set the type of the data parameter to be (char *) or (uint8_t *) To apply for buffer space internally, set the type of the data parameter to be (char **) or (uint8_t **)
- `size`: A non-zero pointer to the variable holding the length of out_value. In case out_value is not zero, will be set to the actual length of the value written.
- `wait_ticks`: Wait time if a packet isn't immediately available
- `type`: Buffer space when reading data

mdf_err_t **mwifi_root_write**(**const** uint8_t *dest_addrs, size_t dest_addrs_num, **const** *mwifi_data_type_t* *data_type, **const** void *data, size_t size, bool block)

The root sends a packet to the device in the mesh.

Attention 1. If data encryption is enabled, you must ensure that the task that calls this function is greater than 8KB

1. This API is only used at the root node
2. Can send data to multiple devices at the same time
3. Packets are only transmitted down

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START

- ESP_ERR_MESH_DISCONNECTED
- ESP_ERR_MESH_OPT_UNKNOWN
- ESP_ERR_MESH_EXCEED_MTU
- ESP_ERR_MESH_NO_MEMORY
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_QUEUE_FULL
- ESP_ERR_MESH_NO_ROUTE_FOUND
- ESP_ERR_MESH_DISCARD

Parameters

- `dest_addrs`: The address of the final destination of the packet
- `dest_addrs_num`: Number of destination addresses
- `data_type`: The type of the data
- `data`: Pointer to a sending wifi mesh packet
- `size`: The length of the data
- `block`: Whether to block waiting for data transmission results

mdf_err_t **__mwifi_root_read**(uint8_t **src_addr*, *mwifi_data_type_t* **data_type*, void **data*, size_t **size*, TickType_t *wait_ticks*, uint8_t *type*)

receive a packet targeted to external IP network root uses this API to receive packets destined to external IP network root forwards the received packets to the final destination via socket. This API is only used at the root node

Attention 1. This API is only used at the root node

1. Judging the allocation of buffers by the type of the parameter 'data' The memory of `data` is externally allocated, and the memory is larger than the total length expected to be received. The memory of `data` is allocated internally by `mwifi_read`, which needs to be released after the call.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `src_addr`: the address of the original source of the packet
- `data_type`: the type of the data
- `data`: Pointer to the received mesh packet To apply for buffer space externally, set the type of the data parameter to be (char *) or (uint8_t *) To apply for buffer space internally, set the type of the data parameter to be (char **) or (uint8_t **)
- `size`: The length of the data

- `wait_ticks`: wait time if a packet isn't immediately available(0:no wait, portMAX_DELAY:wait forever)
- `type`: Buffer space when reading data

mdf_err_t **mwifi_post_root_status**(bool *status*)
Post the toDS state to the mesh stack, Usually used to notify the child node, whether the root is successfully connected to the server.

Attention This API is only for the root.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `status`: this state represents whether the root is able to access external IP network

bool **mwifi_get_root_status**()
Get the toDS state from the mesh stack, Whether the root is successfully connected to the server.

Return

- true
- flase

int8_t **mwifi_get_parent_rssi**()
Get the RSSI of the parent node.

Return RSSI of the parent_cplusplus **MWIFI_H**

Structures

struct mwifi_init_config_t
Mwifi initialization configuration.

Public Members

uint8_t **vote_percentage**
Mesh root configuration.

Vote percentage threshold above which the node becoms a root

uint8_t **vote_max_count**
Max multiple voting each device can have for the self-healing of a MESH network

int8_t **backoff_rssi**
RSSI threshold below which connections to the root node are not allowed

uint8_t **scan_min_count**
Minimum scan times before being a root

bool **root_conflicts_enable**
Allow more than one root in one network

uint16_t **root_healing_ms**
Time lag between the moment a root node is disconnected from the network and the moment the devices start electing another root node

uint16_t **capacity_num**
Mesh network capacity configuration.

Network capacity, defining max number of devices allowed in the MESH network

uint8_t **max_layer_deprecated**
Deprecated, shouldn't use it

uint8_t **max_connection**
Max number of MESH softAP connections

uint16_t **assoc_expire_ms**
Mesh network stability configuration.

Period of time after which a MESH softAP breaks its association with inactive leaf nodes

uint16_t **beacon_interval_ms**
Mesh softAP beacon interval

uint16_t **passive_scan_ms**
Mesh station passive scan duration

uint16_t **monitor_duration_ms**
Period (ms) for monitoring the parent's RSSI. If the signal stays weak throughout the period, the node will find another parent offering more stable connection

int8_t **cnx_rssi**
RSSI threshold above which the connection with a parent is considered strong

int8_t **select_rssi**
RSSI threshold for parent selection. Its value should be greater than switch_rssi

int8_t **switch_rssi**
RSSI threshold below which a node selects a parent with better RSSI

uint8_t **attempt_count**
Parent selection fail times, if the scan times reach this value, device will disconnect with associated children and join self-healing

uint8_t **monitor_ie_count**
Allowed number of changes a parent node can introduce into its information element (IE), before the leaf nodes must update their own IEs accordingly

uint8_t **xon_qsize**
Mesh network data transmission configuration.

Number of MESH buffer queues

bool **retransmit_enable**
Enable a source node to retransmit data to the node from which it failed to receive ACK

bool **data_drop_enable**
If a root is changed, enable the new root to drop the previous packet

uint16_t **max_layer**
Max number of allowed layers

uint8_t **topology**
Topology of mesh network, can be tree or chain

struct mwifi_config_t
Mwifi AP configuration.

Public Members

char **router_ssid**[32]
Information about the router connected to the root. This item does not need to be set in the routerless scheme.

SSID of the router

char **router_password**[64]
Password of router

uint8_t **router_bssid**[6]
BSSID, if this value is specified, users should also specify "router_switch_disable"

uint8_t **mesh_id**[6]
Information about the mesh internal connected.

Mesh network ID. Nodes sharing the same MESH ID can communicate with one another

char **mesh_password**[64]
Password for secure communication between devices in a MESH network

mwifi_node_type_t **mesh_type**
Only MWIFI_MESH_IDLE, MWIFI_MESH_ROOT, MWIFI_MESH_NODE and MWIFI_MESH_LEAF device types are supported. Routerless solutions must be configured as MWIFI_MESH_ROOT or MWIFI_MESH_NODE MWIFI_MESH_IDLE: The node type is selected by MESH according to the network conditions. MWIFI_MESH_ROOT: The Device is the root node MWIFI_MESH_NODE: The device is a non-root node and contains intermediate nodes and leaf nodes. MWIFI_MESH_LEAF: The device is a leaf node, closes the softap, and is used in a low-power solutions.

uint8_t **channel**
Channel, mesh and router will be on the same channel

uint8_t **channel_switch_disable**
If this value is not set, when "attempt" (*mwifi_init_config_t*) times is reached, device will change to a full channel scan for a network that could join.

uint8_t **router_switch_disable**
If the BSSID is specified and this value is not also set, when the router of this specified BSSID fails to be found after "attempt" (*mwifi_init_config_t*) times, the whole network is allowed to switch to another router with the same SSID. The new router might also be on a different channel. There is a risk that if the password is different between the new switched router and the previous one, the mesh network could be established but the root will never connect to the new switched router.

struct mwifi_data_type_t
Mwifi packet type.

Public Members

bool **compression**
Enable data compression. If the data sent is json format or string, use data compression to increase the data transmission rate.

bool **upgrade**
Upgrade packet flag

uint8_t **communicate**
Mesh data communication method, There are three types: MWIFI_COMMUNICATE_UNICAST, MWIFI_COMMUNICATE_MULTICAST, MWIFI_COMMUNICATE_BROADCAST

bool **group**
Send a package as a group

uint8_t **reserved**
reserved

uint8_t **protocol**
Type of transmitted application protocol

uint32_t **custom**
Type of transmitted application data

Macros

MWIFI_PAYLOAD_LEN
< _cplusplus Max payload size(in bytes)

MWIFI_ADDR_LEN
Length of MAC address

MWIFI_ADDR_NONE

MWIFI_ADDR_ROOT

MWIFI_ADDR_ANY
All node in the mesh network

MWIFI_ADDR_BROADCAST
Other node except the root

MWIFI_ADDR_IS_EMPTY(addr)

MWIFI_ADDR_IS_ANY(addr)

MWIFI_ADDR_IS_BROADCAST(addr)

MDF_ERR_MWIFI_NOT_INIT
Mwifi error code definition.

Mwifi isn't initialized

MDF_ERR_MWIFI_INITED
Mwifi has been initialized

MDF_ERR_MWIFI_NOT_START
Mwifi isn't started

MDF_ERR_MWIFI_ARGUMENT
Illegal argument

MDF_ERR_MWIFI_EXCEED_PAYLOAD
Packet size exceeds payload

MDF_ERR_MWIFI_TIMEOUT
Timeout

MDF_ERR_MWIFI_DISCONNECTED
Disconnected with parent on station interface

MDF_ERR_MWIFI_NO_CONFIG
Router not configured

MDF_ERR_MWIFI_NO_FOUND
Routes or devices not found

MDF_ERR_MWIFI_NO_ROOT
Routes or devices not found

MDF_EVENT_MWIFI_STARTED
enumerated list of Mwifi event id

Mwifi is started

MDF_EVENT_MWIFI_STOPPED
Mwifi is stopped

MDF_EVENT_MWIFI_CHANNEL_SWITCH
Channel switch

MDF_EVENT_MWIFI_CHILD_CONNECTED
A child is connected on softAP interface

MDF_EVENT_MWIFI_CHILD_DISCONNECTED
A child is disconnected on softAP interface

MDF_EVENT_MWIFI_ROUTING_TABLE_ADD
Routing table is changed by adding newly joined children

MDF_EVENT_MWIFI_ROUTING_TABLE_REMOVE
Routing table is changed by removing leave children

MDF_EVENT_MWIFI_PARENT_CONNECTED
Parent is connected on station interface

MDF_EVENT_MWIFI_PARENT_DISCONNECTED
Parent is disconnected on station interface

MDF_EVENT_MWIFI_NO_PARENT_FOUND
No parent found

MDF_EVENT_MWIFI_LAYER_CHANGE
Layer changes over the Mwifi network

MDF_EVENT_MWIFI_TODS_STATE
State represents if root is able to access external IP network

MDF_EVENT_MWIFI_VOTE_STARTED
The process of voting a new root is started either by children or by root

MDF_EVENT_MWIFI_VOTE_STOPPED
The process of voting a new root is stopped

MDF_EVENT_MWIFI_ROOT_ADDRESS
The root address is obtained. It is posted by Mwifi stack automatically.

MDF_EVENT_MWIFI_ROOT_SWITCH_REQ
Root switch request sent from a new voted root candidate

MDF_EVENT_MWIFI_ROOT_SWITCH_ACK
Root switch acknowledgment responds the above request sent from current root

MDF_EVENT_MWIFI_ROOT_ASKED_YIELD
Root is asked yield by a more powerful existing root.

MDF_EVENT_MWIFI_SCAN_DONE
if self-organized networking is disabled

MDF_EVENT_MWIFI_NETWORK_STATE
network state, such as whether current mesh network has a root.

MDF_EVENT_MWIFI_STOP_RECONNECTION
the root stops reconnecting to the router and non-root devices stop reconnecting to their parents.

MDF_EVENT_MWIFI_FIND_NETWORK
when the channel field in mesh configuration is set to zero, mesh stack will perform a full channel scan to find a mesh network that can join, and return the channel value after finding it.

MDF_EVENT_MWIFI_ROUTER_SWITCH
if users specify BSSID of the router in mesh configuration, when the root connects to another router with the same SSID, this event will be posted and the new router information is attached.

MDF_EVENT_MWIFI_CHANNEL_NO_FOUND
The router's channel is not set.

MDF_EVENT_MWIFI_ROOT_GOT_IP

MDF_EVENT_MWIFI_ROOT_LOST_IP

MDF_EVENT_MWIFI_EXCEPTION
Some abnormal situations happen, eg. disconnected too many times

CONFIG_MWIFI_ROOT_CONFLICTS_ENABLE
CONFIG_MWIFI_ROOT_CONFLICTS_ENABLE

CONFIG_MWIFI_RETRANSMIT_ENABLE
CONFIG_MWIFI_RETRANSMIT_ENABLE

MWIFI_INIT_CONFIG_DEFAULT()

MWIFI_RSSI_THRESHOUD_DEFAULT()

mwifi_read(src_addr, data_type, data, size, wait_ticks)

mwifi_root_read(src_addr, data_type, data, size, wait_ticks)

Type Definitions

typedef uint8_t **mwifi_node_type_t**

Enumerations

enum node_type_t
Device type.

Values:

MWIFI_MESH_IDLE
hasn't joined the mesh network yet

MWIFI_MESH_ROOT
the only sink of the mesh network. Has the ability to access external IP network

MWIFI_MESH_NODE
intermediate device. Has the ability to forward packets over the mesh network

MWIFI_MESH_LEAF
has no forwarding ability

MWIFI_MESH_STA
connect to router with a standlone Wi-Fi station mode, no network expansion capability

enum mwifi_communication_method
Mesh network internal data transmission method.

Values:

MWIFI_COMMUNICATE_UNICAST
Send data by unicast.

MWIFI_COMMUNICATE_MULTICAST
Send data by multicast.

MWIFI_COMMUNICATE_BROADCAST
Send data by broadcast.

enum mwifi_data_memory_t
Buffer space when reading data.

Values:

MWIFI_DATA_MEMORY_MALLOC_INTERNAL = 1
Buffer space is requested by internal when reading data

MWIFI_DATA_MEMORY_MALLOC_EXTERNAL = 2
Buffer space is requested by external when reading data

2.7 Mdebug API

`Mdebug` (Mesh Network debug) is an important debugging solution used in ESP-MDF. It is designed to efficiently obtain ESP-MDF device logs through wireless espnow protocol, TCP protocol, serial port, etc., so that it can be read more conveniently and quickly. The device log information can then be analyzed according to the extracted logs.

2.7.1 Application Examples

For ESP-MDF examples, please refer to the directory function_demo/Mdebug, which includes:

- The log information of the child node is transmitted to the root node through the *ESP-Mesh* network, and then outputted through the serial port of the root node or transmitted to the server through http;
- Output the desired log information through the console.

1. UART enable

The serial port is enabled and the log information will be printed out via `vprintf`. Enable the following program:

```
if (config->log_uart_enable) { /**< Set log uart enable */
    mdebug_log_init();
} else {
    mdebug_log_deinit();
}
```

2. Flash enable

Write flash enable to store log information in flash. Enable the following program:

```
if (config->log_flash_enable) { /**< Set log flash enable */
      mdebug_flash_init();
} else {
      mdebug_flash_deinit();
}
```

2.1 Log information access

1. The log is initialized, creating two text flash memory spaces, using the main functions as `esp_vfs_spiffs_register`, `esp_spiffs_info`, `sprintf`, `fopen`;

```
/**< Create spiffs file and number of files */
esp_vfs_spiffs_conf_t conf = {
    .base_path       = "/spiffs",
    .partition_label = NULL,
    .max_files       = MDEBUG_FLASH_FILE_MAX_NUM,
    .format_if_mount_failed = true
};
```

2. Write the log data, and write down the address of the write pointer. In order to address the next log write flash, use the main function as `fwrite`. The following procedure:

```
/**< Record the address of each write pointer */
fseek(g_log_info[g_log_index].fp, g_log_info[g_log_index].
↪size, SEEK_SET);
ret = fwrite(data, 1, size, g_log_info[g_log_index].fp);
```

3. Read the log data, and write down the address of the read pointer. For the next time to read the log from the flash, address the address, use the main function as `fread`. The following procedure:

```
/**< Record the pointer address for each read */
flash_log_info_t *log_info = g_log_info + ((g_log_index + 1 +␣
↪i) % MDEBUG_FLASH_FILE_MAX_NUM);
fseek(log_info->fp, log_info->offset, SEEK_SET);

ret = fread(data, 1, MIN(*size - read_size, log_info->size -␣
↪log_info->offset), log_info->fp);
```

4. The data is erased. When the data is full, the data pointer will be cleared, and the main function `rewind` will be restarted from the beginning or the end of the file header address. The following procedure:

```
/**< Erase file address pointer */
for (size_t i = 0; i < MDEBUG_FLASH_FILE_MAX_NUM; i++) {
    g_log_index          = 0;
    g_log_info[i].size   = 0;
    g_log_info[i].offset = 0;
    rewind(g_log_info[i].fp);
    mdf_info_save(MDEBUG_FLASH_STORE_KEY, g_log_info,␣
↪sizeof(flash_log_info_t) * MDEBUG_FLASH_FILE_MAX_NUM);
}
```

Note:

1. The header of the log data is an added timestamp. It is only used as an experiment, and there is no real-time calibration. Users can modify it according to their own needs. The following procedure:

```
/**< Get the current timestamp */
time(&now);
localtime_r(&now, &timeinfo);
strftime(strtime_buf, 32, "\n[%Y-%m-%d %H:%M:%S] ", &timeinfo);
```

2. Extract and select valid string data information. Because the log information in the MDF has unnecessary data at the beginning and the end, it needs to be removed. The following procedure removes unwanted data from the head and tail:

```
/**< Remove the header and tail that appear in the string in the log */
if (log_data->data[0] == 0x1b) {
    data = log_data->data + 7;
    size = log_data->size - 12;
}
```

3. ESPNOW enable

Espnow can be enabled to send the log data of the child node to the root node through espnow, so that the log information of the child node is read from the root node. Enable the following program:

```
/**< config espnow enable */
if (config->log_espnow_enable) {
    mdebug_espnow_init();
} else {
    mdebug_espnow_deinit();
}
```

Write the log to espnow. The procedure is as follows:

```
/**< write log information by espnow */
if (!g_log_config->log_espnow_enable && !MDEBUG_ADDR_IS_EMPTY(g_log_config->dest_
↪addr)) {
    log_data->type |= MDEBUG_LOG_TYPE_ESPNOW;
}

if (log_data->type & MDEBUG_LOG_TYPE_ESPNOW) {
    mdebug_espnow_write(g_log_config->dest_addr, log_data->data,
                        log_data->size, MDEBUG_ESPNOW_LOG, pdMS_TO_TICKS(MDEBUG_LOG_
↪TIMEOUT_MS));
}
```

2.7.2 Mdebug Console

Header File

- mdebug/include/mdebug_console.h

Functions

mdf_err_t **mdebug_console_init**(void)
Initialize console module.

< _cplusplus

- Initialize the console
- Register help commands
- Initialize filesystem
- Create console handle task

Attention Baudrate should not greater than 2030400 if console is enable

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdebug_console_deinit**(void)
De-initialize console module Call this once when done using console module functions.

Return

- MDF_OK
- MDF_FAIL

void **mdebug_cmd_register_common**(void)
Register frequently used system commands.

- version: Get version of chip and SDK
- heap: Get the current size of free heap memory
- restart: Software reset of the chip
- reset: Clear device configuration information
- log: Set log level for given tag
- coredump: Get core dump information_cplusplus **MDF_CONSOLE_DEBUG_H**

2.7.3 Mdebug Espnow

Header File

- mdebug/include/mdebug_espnow.h

Functions

mdf_err_t **mdebug_espnow_init**(void)
Initialize the wireless debug receiver.

- register espnow log redirect function

- Create mdebug espnow send and mdebug log send task
- Create log send queue

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdebug_espnow_deinit**(void)
De-initialize the wireless debug receiver.

Return

- MDF_OK
- MDF_FAIL

mdf_err_t **mdebug_espnow_write**(**const** uint8_t **dest_addr*, **const** void **data*, size_t *size*, *mdebug_espnow_t* *type*, TickType_t *wait_ticks*)
Send debug data with ESP-NOW.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `dest_addr`: Destination address
- `data`: Point to send data buffer
- `size`: send data len
- `type`: Type of data
- `wait_ticks`: wait time if a packet isn't immediately available

mdf_err_t **mdebug_espnow_read**(uint8_t **src_addr*, void **data*, size_t **size*, *mdebug_espnow_t* **type*, TickType_t *wait_ticks*)
receive debug data with ESP-NOW

Return

- ESP_OK
- ESP_FAIL_cplusplus **MDF_ESPNOW_DEBUG_H**

Parameters

- `src_addr`: Destination address
- `data`: Point to send data buffer
- `size`: send data len
- `type`: Type of data
- `wait_ticks`: wait time if a packet isn't immediately available

Structures

struct mdebug_coredump_packet_t
Core dump data structure.

Public Members

uint8_t **type**
Type of packet

int16_t **size**
Size of data

int16_t **seq**
Sequence

uint8_t **data**[230]
data

Enumerations

enum mdebug_espnow_t
Type of data sent during wireless debugging.

< _cplusplus

Values:

MDEBUG_ESPNOW_COREDUMP = 1
Core dump information

MDEBUG_ESPNOW_CONSOLE
Remotely call local terminal commands

MDEBUG_ESPNOW_LOG
Log information

enum [anonymous]
Type of core dump data.

Values:

MDEBUG_COREDUMP_BEGIN = 1
Start transferring core dump data

MDEBUG_COREDUMP_DATA
Send core dump data

MDEBUG_COREDUMP_END
End core dump data transfer

2.7.4 Mdebug Flash

Header File

- mdebug/include/mdebug_flash.h

Functions

mdf_err_t **mdebug_flash_init**()
Init mdebug_flash Create Several files under the spiffs folder,open the file.Open the file for the storage of the next step data.paramters MDEBUG_FLASH_FILE_MAX_NUM if files sizes change.

< _cplusplus

Return

- MDF-OK

mdf_err_t **mdebug_flash_deinit**()
Deinit medbug_flash If you open the file, close the file accordingly.

Return

- MDF-OK

mdf_err_t **mdebug_flash_write**(**const** char *_data_, size_t _size_)
Write memory data in flash.

Note Don't include timestamp in interface input data

Return

- MDF_OK

Parameters

- `data`: Data from the flash's spiffs files in the log
- `size`: Size from the flash's spiffs files in the log

mdf_err_t **mdebug_flash_read**(char *_data_, size_t *_size_)
Read memory data in flash.

Return

- MDF_OK
- read_size

Parameters

- `data`: Data from the flash's spiffs files in the log
- `size`: Size from the flash's spiffs files in the log

mdf_err_t **mdebug_flash_erase**()
Erase when the data and pointers is full.

Return

- MDF_OK

size_t **mdebug_flash_size**()
Create files size,For the data to be stored in the file for subsequent calls.paramters MDEBUG_FLASH_FILE_MAX_NUM if files sizes change.

Return

- size_cplusplus **MDF_DEBUG_FLASH_H**

2.7.5 Mdebug Log

Header File

- mdebug/include/mdebug_log.h

Functions

mdf_err_t **mdebug_log_get_config**(*mdebug_log_config_t *config*)
Get the configuration of the log during wireless debugging.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `config`: The configuration of the log

mdf_err_t **mdebug_log_set_config**(**const** *mdebug_log_config_t *config*)
Set the configuration of the log during wireless debugging.

Return

- MDF_OK
- MDF_FAIL

Parameters

- `config`: The configuration of the log

mdf_err_t **mdebug_log_init**(void)
Init log mdebug.

- Set log mdebug configuration
- Create log mdebug task

Return

- MDF_OK

mdf_err_t **mdebug_log_deinit**(void)
De-initialize log mdebug Call this once when done using log mdebug functions.

Return

- MDF_OK_cplusplus **MDF_DEBUG_LOG_H**

Structures

struct mdebug_log_config_t
Log sending configuration.

< _cplusplus

Public Members

bool **log_uart_enable**
Serial port enable

bool **log_flash_enable**
Write the log to flash enable

bool **log_espnow_enable**
enable log transferred via ESP-NOW

uint8_t **dest_addr**[6]
Turn off log sending if all is zero

struct mdebug_log_queue_t
Type of log storage queue.

Public Members

uint16_t **size**
The length of the log data

mdebug_log_type_t **type**
Ways to send logs

char **data**[0]
Log data

Enumerations

enum mdebug_log_type_t
Set the send type.

Values:

MDEBUG_LOG_TYPE_ESPNOW = 1 << 1

MDEBUG_LOG_TYPE_FLASH = 1 << 2

2.7.6 Mdebug Partially Referenced Header File

Header File

- mdebug/include/mdebug.h

Macros

MDEBUG_PRINTF(fmt, ...)
Configuration mdebug print enable, whether the output information according to the client needs to know. please assign CONFIG_MDEBUG_PRINTF_ENABLE a value.

< _cplusplus

Note CONFIG_MDEBUG_PRINTF_ENABLE = 1 enable CONFIG_MDEBUG_PRINTF_ENABLE = 0 disable

MDEBUG_ADDR_IS_EMPTY(addr)

MDF_EVENT_MDEBUG_FLASH_FULL
_cplusplus **MDF_DEBUG_H**

2.8 Third Party API

The third-party items:

- Driver: drivers for different devices, such as frequently used buttons and LEDs
- Miniz: lossless, high performance data compression library

2.8.1 Driver

We can regard IoT solution project as a platform that contains different device drivers and features.

2.8.2 Miniz

Miniz is a lossless, high performance data compression library in a single source file that implements the zlib (RFC 1950) and Deflate (RFC 1951) compressed data format specification standards. It supports the most commonly used functions exported by the zlib library, but is a completely independent implementation so zlib's licensing requirements do not apply. Miniz also contains simple to use functions for writing .PNG format image files and reading/writing/appending .ZIP format archives. Miniz's compression speed has been tuned to be comparable to zlib's, and it also has a specialized real-time compressor function designed to compare well against fastlz/minilzo.

2.9 Configuration Options

2.9.1 Introduction

ESP-MDF uses Kconfig system to provide a compile-time configuration mechanism. Kconfig is based around options of several types: integer, string, boolean. Kconfig files specify dependencies between options, default values of the options, the way the options are grouped together, etc.

Applications developers can use `make menuconfig` build target to edit components' configuration. This configuration is saved inside `sdkconfig` file in the project root directory. Based on `sdkconfig`, application build targets will generate `sdkconfig.h` file in the build directory, and will make sdkconfig options available to component makefiles.

2.9.2 Using sdkconfig.defaults

When updating ESP-MDF version, it is not uncommon to find that new Kconfig options are introduced. When this happens, application build targets will offer an interactive prompt to select values for the new options. New values are then written into `sdkconfig` file. To supress interactive prompts, applications can either define `BATCH_BUILD` environment variable, which will cause all prompts to be suppressed. This is the same effect as that of `V` or `VERBOSE` variables. Alternatively, `defconfig` build target can be used to update configuration for all new variables to the default values.

In some cases, such as when `sdkconfig` file is under revision control, the fact that `sdkconfig` file gets changed by the build system may be inconvenient. The build system offers a way to avoid this, in the form of `sdkconfig.defaults` file. This file is never touched by the build system, and must be created manually. It can contain all the options which matter for the given application. The format is the same as that of the `sdkconfig` file. Once `sdkconfig.defaults` is created, `sdkconfig` can be deleted and added to the ignore list of the revision control system (e.g. `.gitignore` file for git). Project build targets will automatically create `sdkconfig` file, populated with the settings from `sdkconfig.defaults` file, and the rest of the settings will be set to their default values. Note that when `make defconfig` is used, settings in sdkconfig will be overriden by the ones in `sdkconfig.defaults`.

2.9.3 Configuration Options Reference

Subsequent sections contain the list of available ESP-MDF options, automatically generated from Kconfig files. Note that depending on the options selected, some options listed here may not be visible by default in the interface of menuconfig.

By convention, all option names are upper case with underscores. When Kconfig generates sdkconfig and sdkconfig.h files, option names are prefixed with `CONFIG_`. So if an option `ENABLE_FOO` is defined in a Kconfig file and selected in menuconfig, then sdkconfig and sdkconfig.h files will have `CONFIG_ENABLE_FOO` defined. In this reference, option names are also prefixed with `CONFIG_`, same as in the source code.

2.9.4 Customisations

when the Kconfig tool generates Makefiles (the `auto.conf` file) its behaviour has been customised. In normal Kconfig, a variable which is set to "no" is undefined. In MDF's version of Kconfig, this variable is defined in the Makefile but has an empty value.

(Note that `ifdef` and `ifndef` can still be used in Makefiles, because they test if a variable is defined *and has a non-empty value*.)

When generating header files for C & C++, the behaviour is not customised - so `#ifdef` can be used to test if a boolean config item is set or not.

2.10 Error Codes Reference

This section lists various error code constants defined in ESP-MDF.

For general information about error codes in ESP-MDF, see *Error Handling*.

MDF_FAIL (-1): Generic mdf_err_t code indicating failure

MDF_OK (0): mdf_err_t value indicating success (no error)

MDF_ERR_NO_MEM **(0x100001)**: Out of memory

MDF_ERR_INVALID_ARG **(0x100002)**: Invalid argument

MDF_ERR_INVALID_STATE **(0x100003)**: Invalid state

MDF_ERR_INVALID_SIZE **(0x100004)**: Invalid size

MDF_ERR_NOT_FOUND **(0x100005)**: Requested resource not found

MDF_ERR_NOT_SUPPORTED **(0x100006)**: Operation or feature not supported

MDF_ERR_TIMEOUT **(0x100007)**: Operation timed out

MDF_ERR_INVALID_RESPONSE **(0x100008)**: Received response was invalid

MDF_ERR_INVALID_CRC **(0x100009)**: CRC or checksum was invalid

MDF_ERR_INVALID_VERSION **(0x10000a)**: Version was invalid

MDF_ERR_INVALID_MAC **(0x10000b)**: MAC address was invalid

MDF_ERR_NOT_INIT **(0x10000c)**: MAC address was invalid

MDF_ERR_BUF **(0x10000d)**: The buffer is too small

MDF_ERR_MWIFI_BASE **(0x200000)**: Starting number of MWIFI error codes

MDF_ERR_MWIFI_NOT_INIT **(0x200001)**: Mwifi isn't initialized

MDF_ERR_MWIFI_INITED **(0x200002)**: Mwifi has been initialized

MDF_ERR_MWIFI_NOT_START **(0x200003)**: Mwifi isn't started

MDF_ERR_MWIFI_ARGUMENT **(0x200004)**: Illegal argument

MDF_ERR_MWIFI_EXCEED_PAYLOAD **(0x200005)**: Packet size exceeds payload

MDF_ERR_MWIFI_TIMEOUT **(0x200006)**: Timeout

MDF_ERR_MWIFI_DISCONNECTED **(0x200007)**: Disconnected with parent on station interface

MDF_ERR_MWIFI_NO_CONFIG **(0x200008)**: Router not configured

MDF_ERR_MWIFI_NO_FOUND **(0x200009)**: Routes or devices not found

MDF_ERR_MWIFI_NO_ROOT **(0x20000a)**: Routes or devices not found

MDF_ERR_MESPNOW_BASE **(0x300000)**: Starting number of MESPNOW error codes

MDF_ERR_MCONFIG_BASE **(0x400000)**: Starting number of MCONFIG error codes

MDF_ERR_MUPGRADE_BASE **(0x500000)**: Starting number of MUPGRADE error codes

MDF_ERR_MUPGRADE_FIRMWARE_NOT_INIT **(0x500001)**: Uninitialized firmware configuration

MDF_ERR_MUPGRADE_FIRMWARE_PARTITION **(0x500002)**: Partition table error

MDF_ERR_MUPGRADE_FIRMWARE_INVALID **(0x500003)**: Non-project generated firmware

MDF_ERR_MUPGRADE_FIRMWARE_INCOMPLETE **(0x500004)**: The firmware received by the device is incomplete

MDF_ERR_MUPGRADE_FIRMWARE_DOWNLOAD **(0x500005)**: Firmware write flash error

MDF_ERR_MUPGRADE_FIRMWARE_FINISH **(0x500006)**: The firmware has been written to completion

MDF_ERR_MUPGRADE_DEVICE_NO_EXIST **(0x500007)**: The device that needs to be upgraded does not exist

MDF_ERR_MUPGRADE_SEND_PACKET_LOSS **(0x500008)**: Request device upgrade status failed

MDF_ERR_MUPGRADE_NOT_INIT **(0x500009)**: Upgrade configuration is not initialized

MDF_ERR_MUPGRADE_STOP **(0x50000a)**: Upgrade configuration is not initialized

MDF_ERR_MDEBUG_BASE **(0x600000)**: Starting number of MDEBUG error codes

MDF_ERR_MLINK_BASE **(0x700000)**: Starting number of MLINK error codes

MDF_ERR_CUSTOM_BASE **(0x800000)**: Starting number of COUSTOM error codes

CHAPTER 3

ESP32 Hardware Reference

CHAPTER 4

API Guides

The ESP32 is a powerful chip well positioned as a MCU of the audio projects. This section is intended to provide guidance on process of designing an audio project with the ESP32 inside.

4.1 ESP-WIFI-MESH Basic Information and FAQ

This document provides reference links to some basic information on ESP-WIFI-MESH, followed by the FAQ section.

4.1.1 Reference links

ESP-WIFI-MESH is specifically designed for ESP32 chips. These chips use the modified version of dual-core based FreeRTOS and have an official development framework, which is called ESP-IDF.

1. **About the ESP-WIFI-MESH protocol:**
    - ESP-WIFI-MESH
2. **About ESP32 Chips:**
    - ESP32 Technical Reference Manual
    - ESP32 Datasheet
3. **About ESP-IDF:**
    - ESP-IDF Programming Guide
4. **About FreeRTOS:**
    - FreeRTOS

4.1.2 Frequently Asked Questions

Root Node

1. What is a root node? How is it established?

   The root node is the top node. It gets connected to a server or a router and serves as the only interface between the ESP-WIFI-MESH network in which it is included and an external IP network.

   There are two methods to establish a root node:

   - **Automatic root node election via the physical layer protocol**. This method implies that a mobile phone or a server transmits the router information to devices. They use this information as a reference to elect a root node. For details, please refer to ESP-WIFI-MESH General Process of Building a Network.
   - **Manual root node selection on the application layer**. You may designate a specific device or a certain type of devices as a fixed root node. Other devices nearby will automatically connect to the designated root node and build a network without any assistance from the router or the mobile phone. Please note that if more than one device is designated as a fixed root node, it will lead to creation of multiple networks.

2. Will the root node failure lead to the network crash?

   - **In case of automatic root node election**: When devices in the network detect that a root node is broken, they will elect a new one. For details, please refer to Root Node Failure.
   - **In case of manual root node selection**: Network will be down until a new fixed root node is designated.

Network Configuration

1. Is a router necessary for network configuration?

   As soon as a root node is established, ESP-WIFI-MESH network configuration process becomes independent of routers.

2. How much time does the configuration process take?

   It depends on, but not limited to, the following factors:

   - **Number of devices**: The fewer the number of devices to be configured, the shorter the configuration time.
   - **Arrangement of devices**: The more even the distribution of devices, the shorter the configuration time.
   - **Wi-Fi signal strength of devices**: The stronger the Wi-Fi signal from devices, the shorter the configuration time.

3. Is one router allowed to have only one ESP-WIFI-MESH network?

   The number depends on the following scenarios:

   - **The devices within the router range can communicate either directly or indirectly**
     - In case of different MESH IDs, different MESH networks will be built.
     - **In case of an identical MESH ID:**
       * **If a root node conflict is allowed**: Devices experiencing unstable connection with the existing root node can elect their own root node which will co-exist with the old one. In this case, the network will be split into multiple MESH networks sharing the same ID, but having different root nodes.

        * **If a root node conflict is not allowed**: Devices experiencing unstable connection with the existing root node initiate the election of a new root node, making the old one a non-root node. In this case, there will always be only one MESH network with the same ID and with one root node.

    - The devices within the router range cannot communicate either directly or indirectly. In this case, not being able to detect any existing networks within their range, each device will build its own network.

4. Is an ESP-WIFI-MESH network allowed to have only one root node?

   Yes. An ESP-WIFI-MESH network must have only one root node.

5. If two MESH networks have the same MESH ID, to which of the two networks will a new device connect?

   If the new device is not assigned a parent node, it will automatically connect to the node with the strongest signal.

ESP-WIFI-MESH Performance

1. Does the surrounding environment affect ESP-WIFI-MESH performance?

   ESP-WIFI-MESH is a solution based on the Wi-Fi communication protocol, so its performance can be affected in the same manner as Wi-Fi's performance. In order to overcome any interferences and boost signal strength, please use a less-crowded channel and a high-performance antenna.

2. How far apart can two devices be from each other to maintain a stable connection on an ESP-WIFI-MESH network?

   The communication distance depends on the Wi-Fi signal strength and the bandwidth demands. The higher the signal strength and the less the bandwidth demands, the greater the communication distance can be.

Miscellaneous

1. What is ESP-MDF? How is it related to ESP-WIFI-MESH? How is it different from ESP-WIFI-MESH?

   ESP-MDF (Espressif's Mesh Development Framework) is a development framework built on ESP-WIFI-MESH - a communication protocol module for MESH networks in ESP-IDF. In addition to its basic functionality of ESP-WIFI-MESH network configuration and data transmission, ESP-MDF provides the modules and application examples related to network configuration, Over-the-air (OTA) feature, Cloud connect, local control, device driver, etc. All these features provide convenience for developers using the ESP-IDF or ESP-MDF development frameworks.

2. Since an ESP32 chip has various MAC addresses, which one should be used for ESP-WIFI-MESH communication?

   Each ESP32 chip has MAC addresses for Station (STA), access point (AP), Bluetooth low energy (BLE) and local area network (LAN).

   Their address values are incremented by one, i.e. LAN Mac = BLE Mac + 1 = AP Mac + 2 = STA Mac + 3.

   For example: - MAC for STA: *xx:xx:xx:xx:xx:00* - MAC for AP: *xx:xx:xx:xx:xx:01* - MAC for BLE: *xx:xx:xx:xx:xx:02* - MAC for LAN: *xx:xx:xx:xx:xx:03*

   The device's STA address is used For ESP-WIFI-MESH communication.

3. Can an ESP-WIFI-MESH network function without a router?

Once an ESP-WIFI-MESH network is formed, it can operate on its own, meaning that the communication within the network is independent of any routers.

4.1.3 Feedback

1. **You can provide your feedback on the following platforms:**
    - Espressif ESP-MDF Forum
    - Github ESP-MDF Issues
2. **When providing your feedback, please list the following information:**
    - **Hardware:** Type of your development board
    - **Error description**: Steps and conditions to reproduce the issue, as well as the issue occurrence probabilities
    - **ESP-MDF Version**: Use `git commit` to acquire the version details of ESP-MDF installed on your computer
    - **Logs**: Complete log files for your devices and the `.elf` files from the `build` folder

4.2 Error Handling

4.2.1 Overview

This document provides the introduction to the ESP-MDF related error debugging methods and to some common error handling patterns.

4.2.2 Error Debugging

Error Codes

In most cases, the functions specific to ESP-MDF use the type *`mdf_err_t`* to return error codes. This is a signed integer type. Success (no error) is indicated with the code `MDF_OK`, which is defined as zero.

Various ESP-MDF header files define possible error codes. Usually, these definitions have the prefix `MDF_ERR_`. Common error codes for generic failures (out of memory, timeout, invalid argument, etc.) are defined in the file `mdf_err.h`. Various components in ESP-MDF can define additional error codes for specific situations. The macro definition that starts with `MDF_ERROR_` can be used to check the return value and return the file name and the line number in case of errors. After that, the value can be converted to an error code name with the function `esp_err_to_name()` to help you easier understand which error has happened.

For the complete list of error codes, please refer to *Error Codes*.

Common Configuration

The pattern given below is recommended for use during debugging and not in the production process. Please use the following configuration to identify the issue and then refer to: examples/sdkconfig.defaults:

```
#
# FreeRTOS
#
CONFIG_FREERTOS_UNICORE=y               # Debug portMUX Recursion
CONFIG_TIMER_TASK_STACK_DEPTH=3072      # FreeRTOS timer task stack size

#
# ESP32-specific
#
CONFIG_TASK_WDT_PANIC=y                 # Invoke panic handler on Task Watchdog timeout
CONFIG_TASK_WDT_TIMEOUT_S=10            # Task Watchdog timeout period (seconds)
CONFIG_ESP32_ENABLE_COREDUMP=y          # Core dump destination
CONFIG_TIMER_TASK_STACK_SIZE=4096       # High-resolution timer task stack size
CONFIG_ESP32_ENABLE_COREDUMP_TO_FLASH=y # Select flash to store core dump
```

Fatal Errors

In case of CPU exceptions or other fatal errors, the panic handler may print CPU registers and the backtrace. A backtrace line contains PC:SP pairs, where PC stands for the program counter and SP is for the stack pointer. If a fatal error occurs inside an ISR (Interrupt Service Routine), the backtrace can include PC:SP pairs from both the ISR and the task that was interrupted.

For detailed instructions on diagnostics of unrecoverable errors, please refer to Fatal Errors.

Use IDF Monitor

If `IDF Monitor` is used, Program Counter values are converted into code locations, i.e., function name, file name, and line number:

```
Guru Meditation Error: Core  0 panic'ed (StoreProhibited). Exception was unhandled.
Core 0 register dump:
PC      : 0x400d33ed  PS      : 0x00060f30  A0      : 0x800d1055  A1      : 0x3ffba950
0x400d33ed: app_main at ~/project/esp-mdf-master/examples/get_started/main/get_
↪started.c:200

A2      : 0x00000000  A3      : 0x00000000  A4      : 0x00000000  A5      : 0x3ffbaaf4
A6      : 0x00000001  A7      : 0x00000000  A8      : 0x800d2306  A9      : 0x3ffbaa10
A10     : 0x3ffb0834  A11     : 0x3ffb3730  A12     : 0x40082784  A13     : 0x06ff1ff8
0x40082784: _calloc_r at ~/project/esp-mdf-master/esp-idf/components/newlib/syscalls.
↪c:51

A14     : 0x3ffafff4  A15     : 0x00060023  SAR     : 0x00000014  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0xffffffff

Backtrace: 0x400d33ed:0x3ffba950 0x400d1052:0x3ffbaa60
0x400d33ed: app_main at ~/project/esp-mdf-master/examples/get_started/main/get_
↪started.c:200

0x400d1052: main_task at ~/project/esp-mdf-master/esp-idf/components/esp32/cpu_start.
↪c:476
```

To find the location where a fatal error has occurred, look at the lines which follow the `Backtrace:` line. The first line after the `Backtrace:` line indicates the fatal error location, and the subsequent lines show the call stack.

Do Not Use IDF Monitor

If IDF Monitor is not used, only the backtrace can be acquired. You need to use the command `xtensa-esp32-elf-addr2line` to convert it to a code location:

```
Guru Meditation Error: Core  0 panic'ed (StoreProhibited). Exception was unhandled.
Core 0 register dump:
PC      : 0x400d33ed  PS      : 0x00060f30  A0      : 0x800d1055  A1      : 0x3ffba950
A2      : 0x00000000  A3      : 0x00000000  A4      : 0x00000000  A5      : 0x3ffbaaf4
A6      : 0x00000001  A7      : 0x00000000  A8      : 0x800d2306  A9      : 0x3ffbaa10
A10     : 0x3ffb0834  A11     : 0x3ffb3730  A12     : 0x40082784  A13     : 0x06ff1ff8
A14     : 0x3ffafff4  A15     : 0x00060023  SAR     : 0x00000014  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0xffffffff

Backtrace: 0x400d33ed:0x3ffba950 0x400d1052:0x3ffbaa60
```

Open Terminal, navigate to your project directory, and execute the following command:

```
xtensa-esp32-elf-addr2line -pfia -e build/*.elf Backtrace: 0x400d33ed:0x3ffba950␣
↪0x400d1052:0x3ffbaa60
```

The output would look as follows:

```
0x00000bac: ?? ??:0
0x400d33ed: app_main at ~/project/esp-mdf-master/examples/get_started/main/get_
↪started.c:200
0x400d1052: main_task at ~/project/esp-mdf-master/esp-idf/components/esp32/cpu_start.
↪c:476
```

Heap Memory Debugging

ESP-IDF integrates tools for requesting heap information, detecting heap corruption, and tracing memory leaks. For details, please refer to Heap Memory Debugging.

If you use the APIs from `mdf_mem.h`, you can also utilize these debugging tools. The function *`mdf_mem_print_record()`* can print all the unreleased memory and help quickly identify memory leak issues:

```
I (1448) [mdf_mem, 95]: Memory record, num: 4
I (1448) [mdf_mem, 100]: <mwifi : 181> ptr: 0x3ffc5f2c, size: 28
I (1458) [mdf_mem, 100]: <mwifi : 401> ptr: 0x3ffc8fd4, size: 174
I (1468) [mdf_mem, 100]: <get_started : 96> ptr: 0x3ffd3cd8, size: 1456
I (1468) [mdf_mem, 100]: <get_started : 66> ptr: 0x3ffd5400, size: 1456
```

Note:

1. Configuration: use `make menuconfig` to enable `CONFIG_MDF_MEM_DEBUG`.
2. Only the memory allocated and released with `MDF_*ALLOC` and `MDF_FREE` is logged.

Task Schedule

The function *`mdf_mem_print_heap()`* can be used to acquire the running status, priority, and remaining stack space of all the tasks:

```
Task Name        Status  Prio    HWM     Task
main             R       1       1800    3
IDLE             R       0       1232    4
node_write_task  B       6       2572    16
node_read_task   B       6       2484    17
Tmr Svc          B       1       1648    5
tiT              B       18      1576    7
MEVT             B       20      2080    10
eventTask        B       20      2032    8
MTXBLK           B       7       2068    11
MTX              B       10      1856    12
MTXON            B       11      2012    13
MRX              B       13      2600    14
MNWK             B       15      2700    15
mdf_event_loop   B       10      2552    6
esp_timer        B       22      3492    1
wifi             B       23      1476    9
ipc0             B       24      636     2

Current task, Name: main, HWM: 1800
Free heap, current: 170884, minimum: 169876
```

Note:

1. The function *mdf_mem_print_heap()* can be called to suspend all the tasks, but it may take a while. For this reason, this function is recommended for use during debugging only.
2. Configuration: use `make menuconfig` to enable `CONFIG_FREERTOS_USE_TRACE_FACILITY` and `CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS`.
3. Status: `R` indicates the status *ready*, `B` indicates the status *blocked*.
4. Remaining space: HWM (High Water Mark) must be set to no less than 512 bytes to avoid stack overflow.

4.2.3 Common Errors

Compilation Errors

1. `MDF_PATH` **is not set**:

esp-mdf cannot be found if `MDF_PATH` environment variable is not set:

```
Makefile:8: /project.mk: No such file or directory
make: *** No rule to make target '/project.mk'.  Stop.
```

- **Solution:** Input the command given to configure:

  ```
  $ export MDF_PATH=~/project/esp-mdf
  ```

  Input the following command to verify:

  ```
  $ echo $MDF_PATH
  ~/project/esp-mdf
  ```

2. **The cloned project is not complete**

The option `--recursive` was missing at the time of getting the project with `git clone`. For this reason, the submodules of esp-mdf were not cloned:

```
~/project/esp-mdf/project.mk:9: ~/project/esp-mdf/esp-idf/make/project.mk: No such␣
↪file or directory
make: *** No rule to make target '~/project/esp-mdf/esp-idf/make/project.mk'.  Stop.
```

- **Solution:** Run the command given below to re-get the submodules

  ```
  `shell cd $MDF_PAHT git submodule update --init `
  ```

Flashing Errors

1. **Serial port permission is limited**

In Linux, text telephone devices (TTYs) belong to the dialout group to which regular users do not have access:

```
serial.serialutil.SerialException: [Errno 13] could not open port /dev/ttyUSB0:␣
↪[Errno 13] Permission denied: '/dev/ttyUSB0'
```

- **Solution:**

  1. Modify the permission directly:

     ```
     sudo chmod 0666 /dev/ttyUSB0
     ```

  2. Add the users with limited access to the dialout group, so that they may gain access to the TTYs and other similar devices:

     ```
     sudo gpasswd --add <user> dialout
     ```

2. `make flash` **Errors**

Incompatibility between python and pyserial:

```
AttributeError: 'Serial' object has no attribute 'dtr'
AttributeError: 'module' object has no attribute 'serial_for_url'
```

- **Solution:** Run the command given below. If the issue persists, you can go to esptool issues and search for any related issues:

  ```
  sudo apt-get update
  sudo apt-get upgrade
  sudo pip install esptool
  sudo pip install --ignore-installed pyserial
  ```

ESP-WIFI-MESH Errors

1. **The device cannot connect to the router**

The router name and password have been configured correctly, but the device still cannot connect to the router. In this case, the log would look as follows:

```
I (2917) mesh: [SCAN][ch:1]AP:1, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (2917) mesh: [FAIL][1]root:0, fail:1, normal:0, <pre>backoff:0
```

(continues on next page)

(continued from previous page)

```
I (3227) mesh: [SCAN][ch:1]AP:1, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (3227) mesh: [FAIL][2]root:0, fail:2, normal:0, <pre>backoff:0

I (3527) mesh: [SCAN][ch:1]AP:2, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (3527) mesh: [FAIL][3]root:0, fail:3, normal:0, <pre>backoff:0

I (3837) mesh: [SCAN][ch:1]AP:2, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (3837) mesh: [FAIL][4]root:0, fail:4, normal:0, <pre>backoff:0

I (4137) mesh: [SCAN][ch:1]AP:2, otherID:0, MAP:0, idle:0, candidate:0, root:0,␣
↪topMAP:0[c:0,i:0]<>
I (4137) mesh: [FAIL][5]root:0, fail:5, normal:0, <pre>backoff:0
```

- **Possible Reasons:**
    1. The ESP-WIFI-MESH channel is not configured: For a quick network configuration, ESP-WIFI-MESH scans only a fixed channel, and this channel must be configured.
    2. Connect to a hidden router: The router's BSSID must be configured when ESP-WIFI-MESH connects to the hidden router.
    3. The router's channel is usually not fixed, so it switches channels in accordance with the changes in the network condition.
- **Solution:**
    1. Fixate the router's channel, and set the channel and the router's BSSID;
    2. Allow the device to automatically get the router information with `mwifi_scan()`. Although, it increases the network configuration time.

4.3 Mconfig

[]

Mconfig (Mesh Network Configuration) is a network configuration solution for ESP-WIFI-MESH, which sends network configuration information to ESP-WIFI-MESH devices in a convenient and efficient manner.

Mconfig uses ESP-Mesh App for network configuration. The App employs RSA algorithms for key exchange, 128-AES for data encryption, and CRC for verification and monitoring of transmission errors.

4.3.1 Terminology

Terminology	Description
Device	Any device that is either connected, or can be connected to an ESP-WIFI-MESH network.
Whitelist	List of devices approved for verifying newly-added devices during the chained way of network configuration. A whitelist can be formed by scanning devices' QR codes as well as by approving devices found via Bluetooth scanning. A whitelist includes each device's MAC address and an MD5 (Message-Digest Algorithm 5) hash of its public key.
Configuration Information	The configuration information related to Router and ESP-WIFI-MESH.
AES	Advanced Encryption Standard, also known as Rijndael in the field of cryptography, is a block encryption standard adopted by the U.S. government and is now used worldwide.
RSA	An asymmetric encryption algorithm that is widely used in the areas of public-key encryption and e-commerce.
CRC	Cyclic Redundancy Check is a hash function that generates a short verification code of fixed length for any network data packet or a file. It is mainly used for detection of any possible errors during data transmission or after data has been saved.

4.3.2 Process

The detailed description of the process given in the figure can be found below.

1. ***Mconfig-BluFi*: App implements network configuration for a single device through Bluetooth.**

    a. App scans the Bluetooth packet of each device and generates a whitelist, which, for example, includes Devices *A*, *B*, *C*, and *D*. Since the signal of Device *A* is the strongest, App establishes connection and sends to this device the configuration information as well as the whitelist.

    b. Device *A* uses the configuration information to connect to the access point (AP), Router in this case.

    c. Device *A* checks if the configuration information is correct, according to the status returned by AP.

    d. **Device *A* returns the network configuration status to App and does the following:**

    - In case the configuration information is correct: Device *A* prompts other devices to request the network configuration information through a Wi-Fi beacon frame.
    - In case the configuration information is not correct: Device *A* returns possible failure reasons to App and goes back to its status at Step *a*.

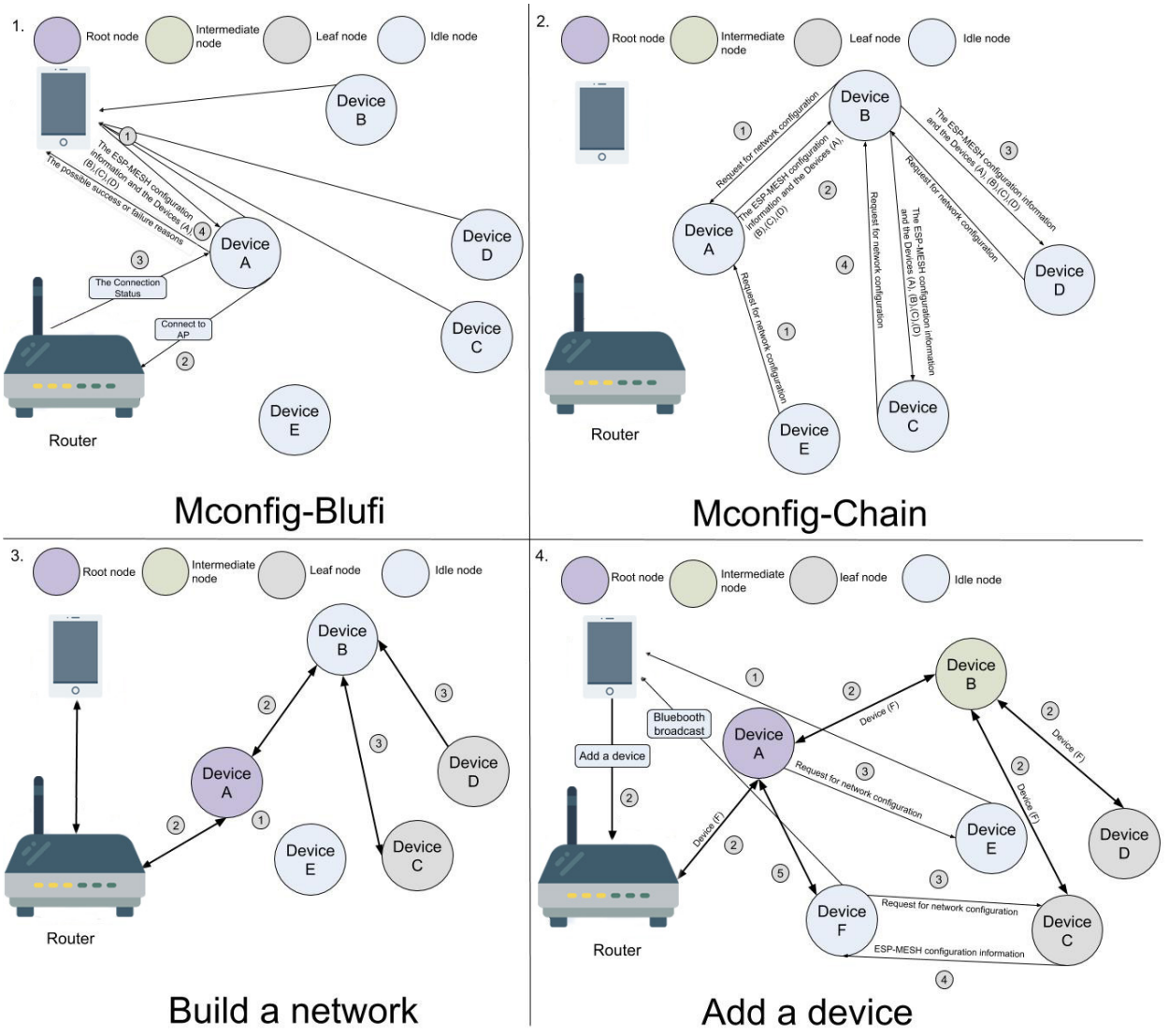
2. ***Mconfig-Chain*: The master, a single configured device, transfers the network configuration information to its slaves throu**

    a. Device *A*, as the master, sends notifications to its slaves - Devices *B* and *E*. After receiving the notifications, both devices return requests for network configuration information.

    b. **Device *A* checks the requesting devices against the whitelist:**

    - If Device *B* is on the list, it will receive the network configuration information from Device *A*.
    - If Device *E* is not whitelisted, its request will be ignored.

    c. As soon as Device *B* obtains the network configuration information, it becomes a master for Devices *C* and *D* and starts the same process.


1.
Root node
Intermediate node
Leaf node
Idle node
Device B
Device A
Device D
Device C
Device E
Router
The ESP-MESH configuration information and the Devices (A),(B),(C),(D)
The possible success or failure reasons
The Connection Status
Connect to AP
Mconfig-Blufi
2.
Root node
Intermediate node
Leaf node
Idle node
Device B
Device A
Device D
Device C
Device E
Router
Request for network configuration
The ESP-MESH configuration information and the Devices (A),(B),(C),(D)
Mconfig-Chain
3.
Root node
Intermediate node
Leaf node
Idle node
Device B
Device A
Device D
Device C
Device E
Router
Build a network
4.
Root node
Intermediate node
leaf node
Idle node
Device B
Device A
Device D
Device C
Device E
Device F
Router
Bluebooth broadcast
Add a device
Device (F)
Request for network configuration
ESP-MESH configuration information
Add a device

3. **Building a network**

    a. Root node selection: Devices *A*, *B*, *C* and *D* elect Device *A* as a root node, because of its optimal relay position for passing signal between each node and Router.

    b. Second layer formation: Once the elected root node is connected to Router, the idle node, Device *B*, within the coverage of the root node connects to Device *A*, thereby forming the second layer of the network;

    c. Formation of remaining layers: Devices *C* and *D* connect to Device *B*, forming the third layer, and so on. In this case Device *B* functions as an intermediate parent node for its leaf nodes - Devices *C* and *D*.

4. **Add a new device**

    a. App listens for incoming Bluetooth advertising packets from added devices. At some point a prompt pops up, asking the user whether Devices *E* and *F* should be added. If Device *F* is chosen to be added, App sends the command to the root node to add Device *F*.

    b. Device *A*, the root node, receives the command and forwards it to other devices in the ESP-WIFI-MESH network. When Devices *B*, *C*, and *D* receive this command to add Device *F*, Mconfig-Chain gets activated.

    c. Device *F* sends a request for network configuration to Device *C*, the one with the strongest signal.

    d. Device *C* sends the network configuration information to Device *F* through Wi-Fi.

    e. Device *F* connects to the ESP-WIFI-MESH network as soon as it receives the network configuration information.

4.3.3 Security

- **Data security**
    - **RSA**: Generation of random public keys.
    - **AES**: Encryption of network configuration information to ensure the security of data transmission.
- **Device security**
    - **Mconfig-BluFi**: Screening devices via App, based on their signal intensity. It helps avoid potential attacks from malicious devices that use weak signal to disguise themselves.
    - **Mconfig-Chain**: To avoid potential attacks from disguised devices, this solution sets a time window for network configuration, during which only network configuration requests are accepted.
- **ID security (optional)**
    - Downloads to each device a public/private RSA key pair that is randomly generated by a downloading tool, then uploads the public key to a cloud for future verification.
    - Encrypts the flash partition that stores the public/private RSA key pair.

4.3.4 Notice

If you want to customize network configuration, please be sure to:

- **Verify password**: The nodes in ESP-WIFI-MESH, except for the root node, do not check the router information. They only check if the configuration within their ESP-WIFI-MESH network is correct. For this reason, if you accidentally provide a wrong router password to a non-root node, it will not be able to connect to the router after

becoming a root node, even if their configuration within the network is correct. To avoid this potential problem, please verify the router password for non-root nodes.

4.3.5 Mconfig-BluFi

Mconfig-BluFi is a network configuration protocol, that expands the features of BluFi - a Bluetooth network configuration protocol defined by Espressif. The additional features include advertizing packet definitions, RSA encryption, and ID authentication.

Mconfig-BluFi is generally used by hardware, such as mobile phones, other devices supporting Bluetooth connection, and routers.

The network configuration process consists of four phases:

- Device discovery
- Key exchange
- Data communication
- Verification of network configuration

Note: Prior to using Mconfig-BluFi, the Bluetooth protocol stack must be enabled. Mconfig-BluFi also demands more resources, so please consider the following:

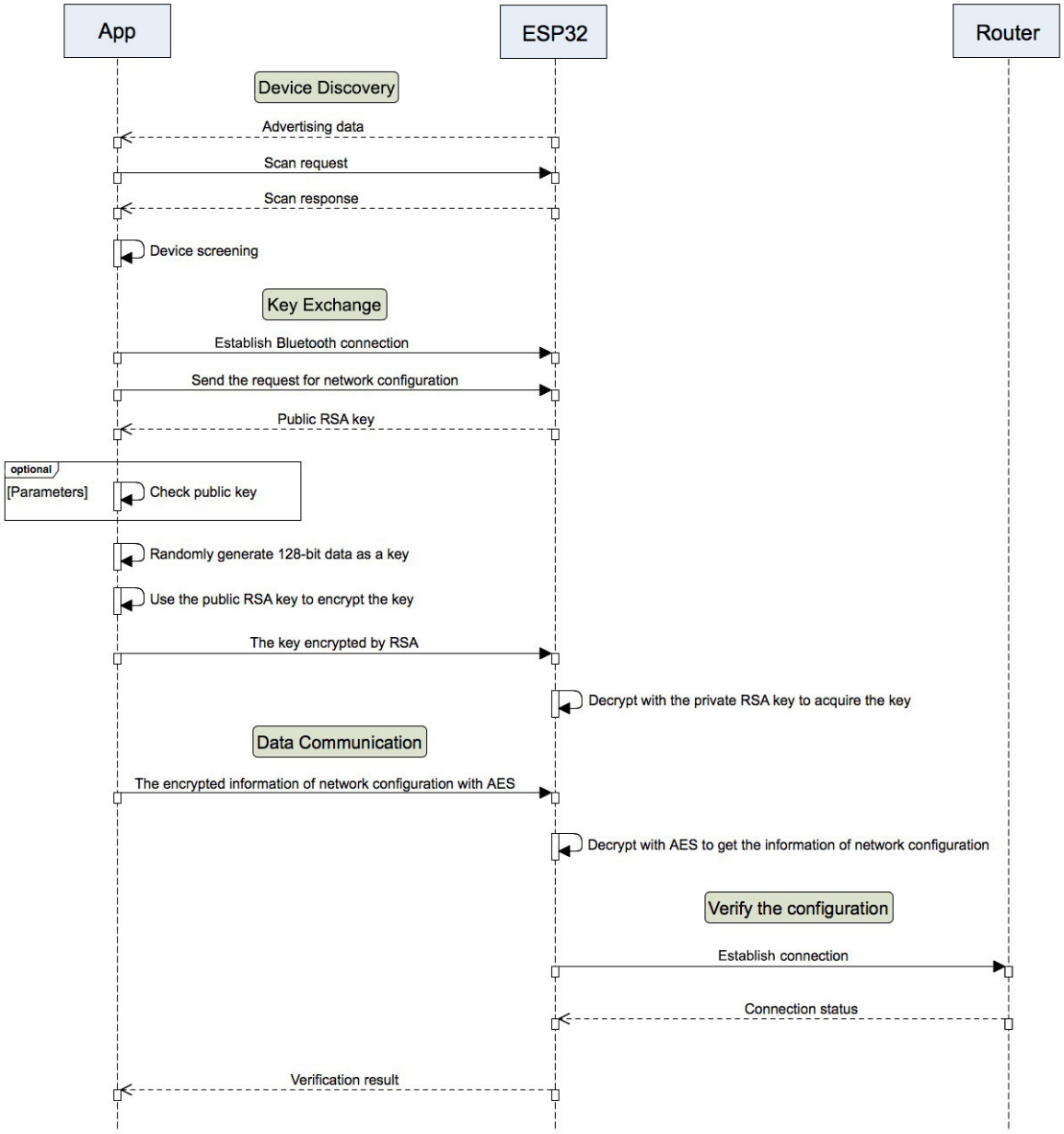
1. **Firmware size increase**: As the firmware size will increase about 500 KB, it is recommended to modify the flash partition table and ensure that the size of the partition for firmware exceeds 1 MB.
2. **Memory usage increase**: Extra 30 KB of the memory will be used. If you need to free this memory, please be aware that Bluetooth will only function as usual after reboot.

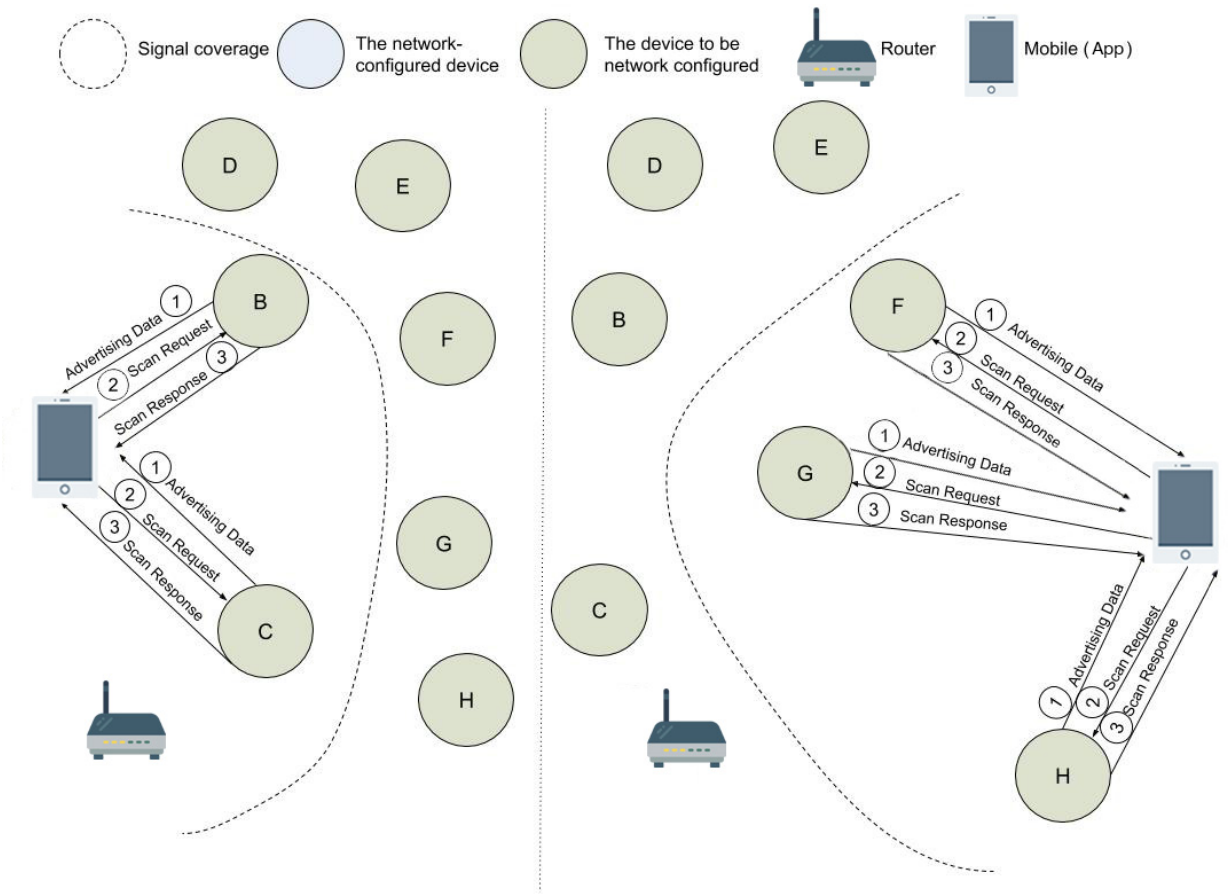
Device discovery

Devices periodically send Bluetooth advertising packets through BLE, while App listens for the packets and screens the devices according to their signal strength. Then App generates a whitelist to avoid adding wrong devices to its network later. The process is shown below.

There are two types of Bluetooth advertising packets:

- **Advertising Data: Storing the customized data of a specific product**
  - Maximum length is 31 bytes.
  - The data format must meet the requirements of Bluetooth Specification.
- **Scan Response: Storing the information of network configuration**
  - Device name uses 10 bytes.
  - Manufacturer information uses 14 bytes. See the details in the table below.


App
ESP32
Router
Device Discovery
Advertising data
Scan request
Scan response
Device screening
Key Exchange
Establish Bluetooth connection
Send the request for network configuration
Public RSA key
optional
[Parameters]
Check public key
Randomly generate 128-bit data as a key
Use the public RSA key to encrypt the key
The key encrypted by RSA
Decrypt with the private RSA key to acquire the key
Data Communication
The encrypted information of network configuration with AES
Decrypt with AES to get the information of network configuration
Verify the configuration
Establish connection
Connection status
Verification result


Signal coverage
The network-configured device
The device to be network configured
Router
Mobile (App)
D
E
B
F
G
C
H
1 Advertising Data
2 Scan Request
3 Scan Response

Field	Length	Description
company id	2 bytes	The only ID assigned to SIG member companies by Bluetooth SIG
OUI	2 bytes	The Mconfig Blufi ID that is used to filter broadcast packets. It takes the form of 0x4d, 0x44, 0x46, i.e. "MDF"
version	2 bits	The current version
whitelist	1 bit	Flag to enable whitelist filter
security	1 bit	Flag to verify the validity of whitelisted deviced
reserved	4 bits	Reserved for future extension
sta mac	6 bytes	MAC address of the device (station)
tid	2 bytes	Device type

Key Exchange

1. App connects and sends the request for network configuration through BLE to Device *A*, which exhibits the strongest signal.
2. Device *A* receives the request and returns its public RSA key to App.
3. App verifies the validity of the public RSA key.
4. App randomly generates a 128-bit key and encrypts it with the public RSA key. Later this encrypted key is sent to Device *A*.
5. Device *A* decrypts the received data with its private RSA key to obtain the 128-bit key, which is then encrypted with AES to secure data transmission between App and Device *A*.

Data Communication

App places the network configuration information and the whitelist into a data packet, and then transfers it as BluFi custom data.

The data packet is encoded in a TLV (Type-length-value or Tag-length-value) format, which is detailed below.

Type	Definition	Length, bytes	Description
	Router Configuration		
1	BLUFI_DATA_ROUTER_SSID	32	SSID of the router
2	BLUFI_DATA_ROUTER_PASSWORD	64	Router password
3	BLUFI_DATA_ROUTER_BSSID	6	BSSID is equal to the ro
4	BLUFI_DATA_MESH_ID	6	Mesh network ID. Node
5	BLUFI_DATA_MESH_PASSWORD	64	Password for secure cor
6	BLUFI_DATA_MESH_TYPE	1	Only MESH_IDLE, ME
	Mesh Network Configuration		
16	BLUFI_DATA_VOTE_PERCENTAGE	1	Vote percentage thresho
17	BLUFI_DATA_VOTE_MAX_COUNT	1	Max multiple voting eac
18	BLUFI_DATA_BACKOFF_RSSI	1	RSSI threshold below w
19	BLUFI_DATA_SCAN_MIN_COUNT	1	The minimum number o
20	BLUFI_DATA_SCAN_FAIL_COUNT	1	Max fails (60 by default
21	BLUFI_DATA_MONITOR_IE_COUNT	1	Allowed number of cha
22	BLUFI_DATA_ROOT_HEALING_MS	2	Time lag between the m

Type	Definition	Length, bytes	Description
23	BLUFI_DATA_ROOT_CONFLICTS_ENABLE	1	Allow more than one ro
24	BLUFI_DATA_FIX_ROOT_ENABLE	1	Enable a device to be se
25	BLUFI_DATA_CAPACITY_NUM	2	Network capacity, defin
26	BLUFI_DATA_MAX_LAYER	1	Max number of allowed
27	BLUFI_DATA_MAX_CONNECTION	1	Max number of MESH
28	BLUFI_DATA_ASSOC_EXPIRE_MS	2	Period of time after whi
29	BLUFI_DATA_BEACON_INTERVAL_MS	2	Mesh softAP beacon int
30	BLUFI_DATA_PASSIVE_SCAN_MS	2	Mesh station passive sc
31	BLUFI_DATA_MONITOR_DURATION_MS	2	Period (ms) for monitor
32	BLUFI_DATA_CNX_RSSI	1	RSSI threshold above w
33	BLUFI_DATA_SELECT_RSSI	1	RSSI threshold for pare
34	BLUFI_DATA_SWITCH_RSSI	1	RSSI threshold below w
35	BLUFI_DATA_XON_QSIZE	1	Number of MESH buffe
36	BLUFI_DATA_RETRANSMIT_ENABL	1	Enable a source node to
37	BLUFI_DATA_DROP_ENABLE	1	If a root is changed, ena
Whitelist Configuration			
64	BLUFI_DATA_WHITELIST	6 * N	Device address
		32 * N	Verify the validity of th

Verification of Network Configuration

When a device receives the network configuration information from AP, it connects to AP to verify if the information is correct. Then a device returns the connection status as well as the verification result to App, in the following format:

Type	Definition	Description
0	ESP_BLUFI_STA_CONN_SUCCESS	Connecting to router successful
1	ESP_BLUFI_STA_CONN_FAIL	Connecting to router failed
16	BLUFI_STA_PASSWORD_ERR	Password configuration error
17	BLUFI_STA_AP_FOUND_ERR	Router is not found
18	BLUFI_STA_TOOMANY_ERR	Router reached max number of connections
19	BLUFI_STA_CONFIG_ERR	Parameter configuration error

4.3.6 Mconfig-Chain

Mconfig-Chain is a network configuration protocol for devices based on ESP-NOW, a connectionless Wi-Fi communication protocol defined by Espressif.

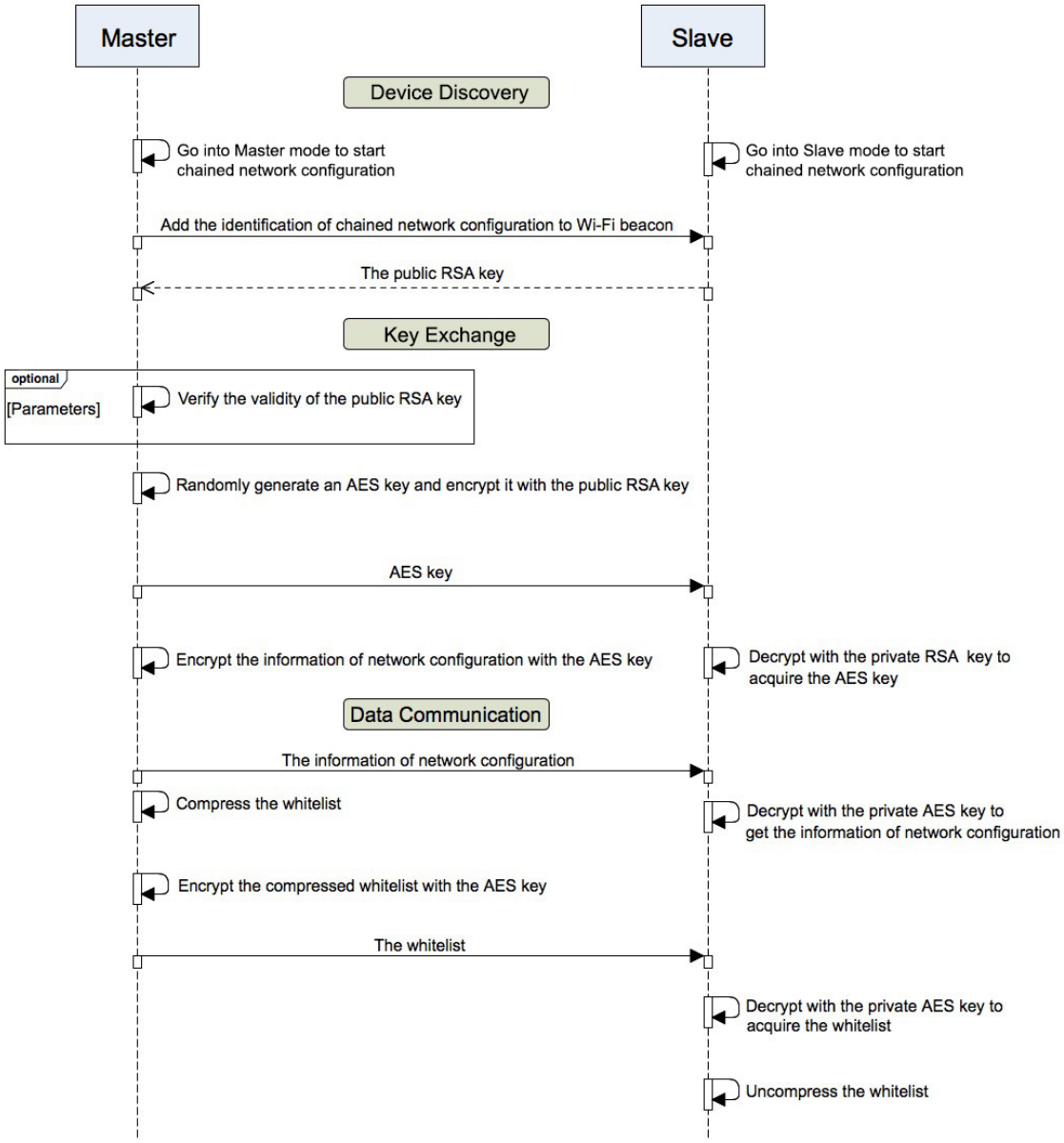
Currently, there are three methods to configure a Wi-Fi network: BLE, sniffer, and softAP, all of which are designed for network configuration of a single device. For this reason, these methods are not applicable for an ESP-WIFI-MESH network, which usually involves network configuration of multiple devices.

Mconfig-Chain is specifically designed for ESP-WIFI-MESH network configuration. It features a chained, transferable configuration process, which means that each device connected to the network can implement network configuration for other devices. Mconfig-Chain turns configuration process of a wide-range network into a simple and efficient process.

Mconfig-Chain splits devices into two types:

- **Master**: a device that initiates a connection
- **Slave**: a device that accepts a connection request

The network configuration process consists of three phases: Device Discovery, Key Exchange, and Data Communication.



Device Discovery

1. **A master adds an identification of chained network configuration to Vendor IE of Wi-Fi beacon frames and awaits the req**

    - An examples of Vendor IE identification is shown below:

Type	Data
Element ID	0xDD
Length	0X04
OUI	0X18, 0XFE, 0X34
Type	0X0F

- A master sets a window period, during which only the request from a slave can be accepted.
- The identification of chained network configuration is sent through Wi-Fi beacon frames. So if a device has STA mode only, then it cannot become a master.

2. **A slave enables a Wi-Fi sniffer in order to find the identification of chained network configuration. A sniffer keeps switchi**

    - Slaves need to switch channels when they sniffer Wi-Fi adverting packets, but the ESP-WIFI-MESH function of network self-forming does not allow channel switching. Whenever a slave switches to another channel, the function automatically switches back to its original channel. Therefore, before using slave sniffer, the function of network self-forming in ESP-WIFI-MESH should be disabled.

Key Exchange

1. A master receives the request for network configuration from a slave and checks if a slave is in the whitelist. In order to verify the device's ID, the ID authentication needs to be enabled. It requires implementing MD5 algorithms for the received public RSA key, and checking its validity against the whitelist.
2. A master removes Vendor IE identification of chained network configuration from the Wi-Fi beacon frame.
3. A master randomly generates 128-bit data as the key to communicate with a slave, encrypts it with the received public RSA key, and then sends the encrypted key to the slave through ESP-NOW.
4. The slave receives the encrypted key from the master and decrypts it using the private RSA key to acquire the communication key with the master.

Data Communication

1. The master encrypts the network configuration information as well as the whitelist using the AES key and sends it to the slave through ESP-NOW.
2. The slave uses its AES key to decrypt the received data and completes network configuration. Then it stop functioning as a slave and becomes a master.

Note: As ESP-NOW implements data encryption on the data link layer. For this, the communicating devices must use an identical key, which should be written in flash or directly downloaded into firmware.

4.4 Mlink

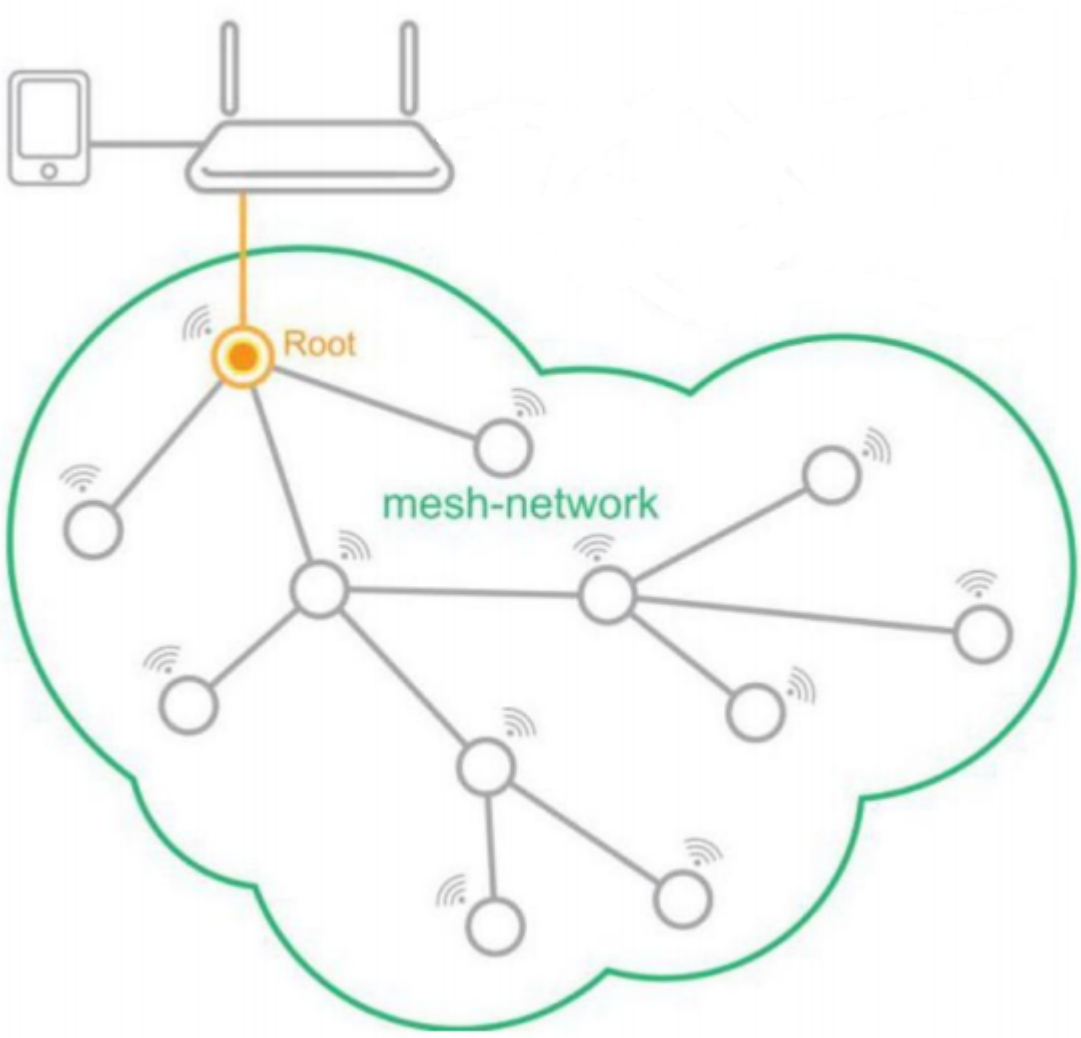
[]

4.4.1 1. Preparation

The prerequisite for the ESP-Mesh app to communicate with the ESP-MDF devices is that the device has been successfully networked, and the mobile phone and mesh network are on the same LAN.

The ESP-Mesh app is hereafter referred to as "app" in this document.

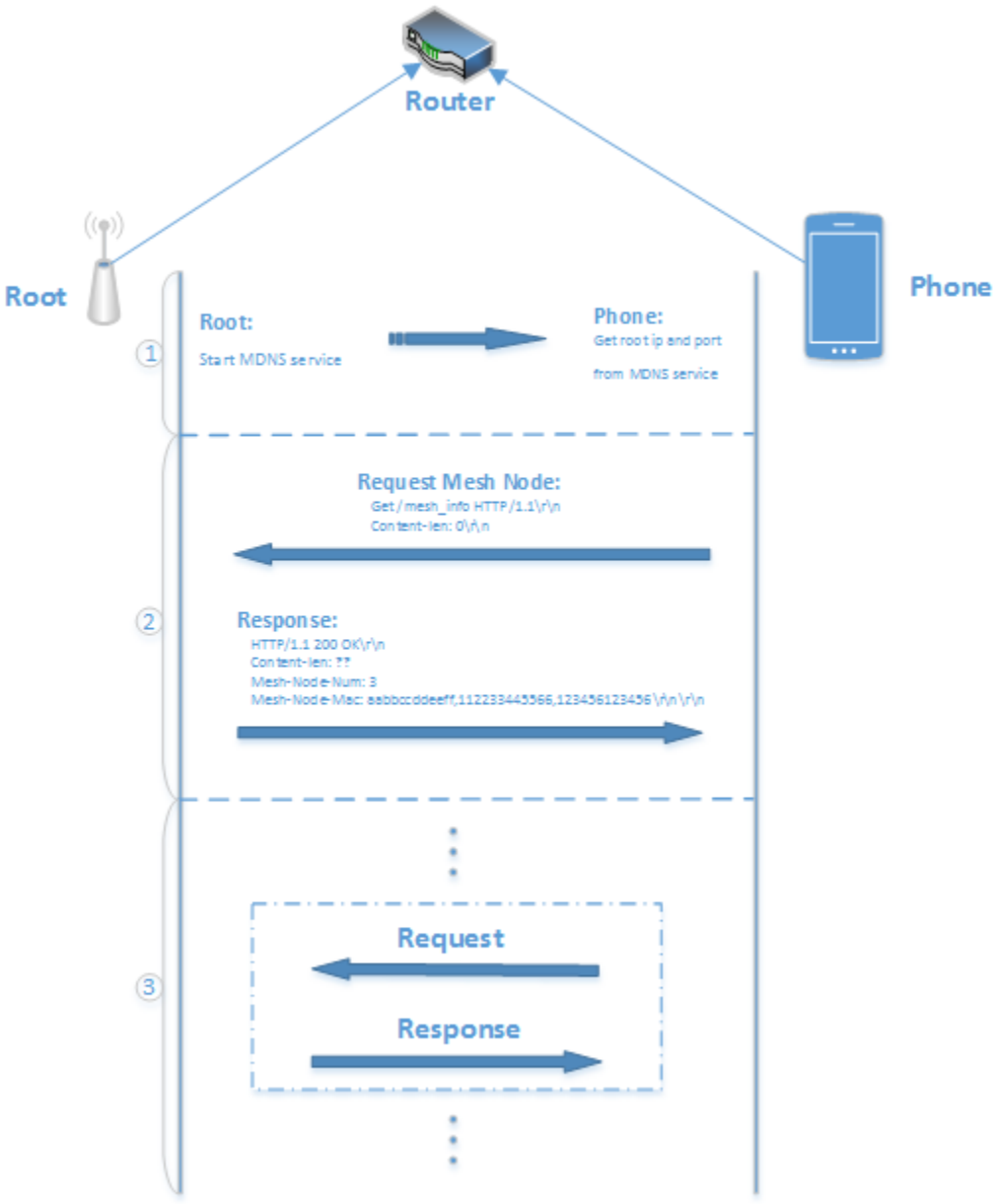
The ESP-MDF devices automatically enter the networking stage once configured. If all the networked devices are shown in the ESP-Mesh app, it means that the networking is successful.



4.4.2 2. Communication Process

The root node is the only interface of the mesh network that communicates with external network. To control the devices in the mesh network, the app needs to find the root node; then it obtains the MAC address of all devices in the mesh network from the root node; finally, the root node can communicate with any device on the mesh network. The whole process is divided into three steps:

1. The app acquires the IP address, Port number and MAC address of the root node.
2. The app inquires a list of the network's devices from the mesh root node.
3. The app and the mesh devices communicate with each other.


Router
Root
Phone
Root:
Start MDNS service
Phone:
Get root ip and port
from MDNS service
Request Mesh Node:
Get /mesh_info HTTP/1.1\r\n
Content-len: 0\r\n
Response:
HTTP/1.1 200 OK\r\n
Content-len: ??
Mesh-Node-Num: 3
Mesh-Node-Mac: aabbccddeeff,112233445566,123456123456\r\n\r\n
Request
Response
1
2
3

4.4.3 3. Communication Protocol

This chapter illustrates the communication protocols involved in the three steps mentioned above. The app gets the list of networked mesh devices, and communicates with devices in the mesh network using the standard HTTP or HTTPS communication protocol. In addition, the communication protocol also includes:

1. Device status notification: This is to allow the users to check the real-time status of the device through the app. That is, when the device status changes, the device will send UDP broadcast packets notifying the app of the changes, and then the app will inquire about the updated status of the device.
2. Local connectivity control: connectivity control between devices on the LAN.

3.1. App Gets the IP Address, Port Number and MAC Address of the Root Node

At this stage, the root node enables the mDNS service and UDP broadcasting function needed for device discovery. During device discovery, the app obtains the IP address, port number, and MAC address of the root node.

1. mDNS Service for Device Discovery

The app will then acquire the root node's IP address through mDNS service, as well as the corresponding port number and MAC address from the `port` and `txt` fields of the service info.

mDNS Service Info:

```
hostname: "esp32_mesh"
instance_name: "mesh"
service_type: "_mesh-https"
proto: "_tcp"
port: 80
txt:
    key: "mac"
    value: "112233445566"
```

2. Receiving Broadcast UDP Packets

When scanning for devices, the app broadcasts UDP packets and obtains the information about the root node from its reply.

Request:

```
"Are You Espressif IOT Smart Device?"
```

Response:

```
"ESP32 Mesh 112233445566 http 80"
```

Note:

- `112233445566` is the MAC address of the root node
- `80` is the http service port
- In addition, the app obtains the IP address of the root node through the UDP packets replied by the root node.

3.2. App Acquires the Device List

Request::

```
GET /mesh_info HTTP/1.1
Host: 192.168.1.1:80
```

Response::

```
HTTP/1.1 200 OK
Content-Length: ??
Mesh-Node-Mac: aabbccddeeff,112233445566,18fe34a1090c
Host: 192.168.1.1:80
```

Note:

- `/mesh_info` is the app command for abtaining the list of devices, which can be implemented via http URL field;
- `Mesh-Node-Mac` is the list of the node' Station MAC addresses, separated by commas;
- `Host` is a required field of the HTTP/1.1 protocol, indicating the console's IP address and port number.

3.3. App and ESP-MDF Device Communication Format

1. App Requests Format

Request::

```
POST /device_request HTTP/1.1
Content-Length: ??
Content-Type: application/json
Root-Response::??
Mesh-Node-Mac: aabbccddeeff,112233445566
Host: 192.168.1.1:80

**content_json**
```

1. `/device_request` is the app command for controlling devices, which, apart from other things, can set and get the device status, via an http request through the URL field.
2. `Content-Length` is the length of the http message body.
3. `Content-Type` is the data type of the http message body, in the format of `application/json`.
4. `Root-Response` decides whether only replies from the root node are needed. If only the replies from the root node are required, the command will not be forwarded to the mesh devices. Value `1` means replies from the root node are required; `0` means no reply from the root node is required.

Note:

- `Host` is a required field in the HTTP/1.1 protocol, indicating the app's IP address and port number.
- `**content_json**` is the http message body, corresponding to the `Request` in `3.4. App's Control of ESP-MDF Devices`.

---

2. Device Replies

Response:

```
HTTP/1.1 200 OK
Content-Length: ??
Content-Type: application/json
Mesh-Node-Mac: 30aea4062ca0
Mesh-Parent-Mac: aabbccddeeff
Host: 192.168.1.1:80
\r\n
**content_json**
```

1. `Content-Length` is the length of the http message body.
2. `Content-Type` is the data type of the http message body, in the `application/json` format.
3. `Mesh-Node-Mac` is the MAC address of the device.
4. `Mesh-Parent-Mac` is the MAC address of the device's parent node.
5. `Host` is a required field in the HTTP/1.1 protocol, indicating the app's IP address and port.
6. `**content_json**` is the http message body that corresponding to the `Request` in `2.4. App's Control of ESP-MDF Devices`.

3.4. App's Control of ESP-MDF Devices

1. Acquire device information: get_device_info

Request:

```
{
    "request": "get_device_info"
}
```

- `request` is field defining the operation on the device, followed by specific commands of operation.

Response:

```
{
    "tid": "1",
    "name": "light_064414",
    "version": "v0.8.5.1-Jan 17 2018",
    "characteristics": [
        {
            "cid": 0,
            "name": "on",
            "format": "int",
            "perms": 7,
            "value": 1,
            "min": 0,
            "max": 1,
            "step": 1
        },
        {
            "cid": 1,
            "name": "hue",
            "format": "int",
            "perms": 7,
            "value": 0,
            "min": 0,
            "max": 360,
```

(continues on next page)

(continued from previous page)


```
            "step": 1
        },
        {
            "cid": 2,
            "name": "saturation",
            "format": "int",
            "perms": 7,
            "value": 0,
            "min": 0,
            "max": 100,
            "step": 1
        },
        {
            "cid": 3,
            "name": "value",
            "format": "int",
            "perms": 7,
            "value": 100,
            "min": 0,
            "max": 100,
            "step": 1
        },
        {
            "cid": 4,
            "name": "color_temperature",
            "format": "int",
            "perms": 7,
            "value": 0,
            "min": 0,
            "max": 100,
            "step": 1
        },
        {
            "cid": 5,
            "name": "brightness",
            "format": "int",
            "perms": 7,
            "value": 100,
            "min": 0,
            "max": 100,
            "step": 1
        }
    ],
    "status_code": 0
}
```


- `tid` is the type ID of the device, which is used to distinguish different types of devices from each other, such as lights, sockets, and air conditioners.
- `name` is the device name.
- `version` is the device firmware version.
- **`characteristics` is the device characteristics, in json format.**
  - `cid` is the characteristic ID of the deviceindicating charactertistics such as brightness, hue, switches, etc.
  - `name` is the name of the device characteristics.

  - `format` is the data format. Four data types `int`, `double`, `string`, `json` are supported.
  - `value` is the value of the device characteristics.
  - `min` is the minimum value or the minimum length of the data string of `charactertics`
  - `max` is the maximum value or the maximum length of the data string of `charactertics`
  - **`step` is the minimum variation of the characteristics value**
    - When `format` is `int` or `double`, `min`,‘‘max‘‘ and `step` represent the minimum value, maximum value, and the minimum variation of the charateristics.
    - When `format` is `string` or `json`, `min` and `max` indicate the minimum and maximum lengths of the string supported respectively, without the keyword `step`.

- **`perms` stands for permission, parsed in binary integers, with the first bit representing a read permission, the second bit re**
  - If the parameter has no read permission, the corresponding value can not be accessed.
  - If the parameter has no write permission, the corresponding value can not be modified.
  - If the parameter has no execution permission, the corresponding value can not be set.
- `status_code` is the reply to the request commands; `0` indicates normal, and `-1` indicates error.

2. Acquire device status: get_status

Request:

```
{
    "request": "get_status",
    "cids": [
        0,
        1,
        2
    ]
}
```

- `cids` is the field of device characteristics, followed by the CID list of the request.

Response:

```
{
    "characteristics": [
        {
            "cid": 0,
            "value": 0
        },
        {
            "cid": 1,
            "value": 0
        },
        {
            "cid": 2,
            "value": 100
        }
    ],
    "status_code": 0
}
```

- `status_code` is the reply to the request command, `0` indicates normal, `-1` indicates that the request contains illegal parameters, such as lack of corresponding CID for a device or a value with no read permission in the `cids` list.

3. Configure the device status: set_status

Request:

```
{
    "request": "set_status",
    "characteristics": [
        {
            "cid": 0,
            "value": 0
        },
        {
            "cid": 1,
            "value": 0
        },
        {
            "cid": 2,
            "value": 100
        }
    ]
}
```

Response:

```
{
    "status_code": 0
}
```

- `status_code` is the reply value to the request command, `0` indicates normal, `-1` indicates that the request contains illegal parameters, such as lack of corresponding CID for a device or a value with no read permission in the `cids` list.

4. Enters the networking mode: config_network

Request:

```
{
    "request": "config_network"
}
```

Response:

```
{
    "status_code": 0
}
```

- `status_code` is the reply value to the request command, `0` indicates normal, `-1` indicates error.

5. Reboots the device: reboot

Request:

```
  {
      "request": "reboot",
      "delay": 50
  }
```

(continues on next page)

(continued from previous page)

```
``delay`` is the delay for executing the command. This field is not required. The␣
↪default delay is ``2s``.
```

Response:

```
{
    "status_code": 0
}
```

- `status_code` is the reply to the request command, `0` indicates normal, `-1` indicates error.

6. Reset the device: reset

Request:

```
{
    "request": "reset",
    "delay": 50
}
```

- `delay` is the delay for executing the command. This field is not required. The default delay is `2s`.

Response:

```
{
    "status_code": 0
}
```

- `status_code` is the reply value to the request command; `0` indicates normal, and `-1` indicates error.

3.5. Device Status Notification

When the status of the ESP-MDF device (on/off), network connection (connected or disconnected), and route table change, the root node will send broadcast UDP packets to notify the app to obtain the latest status of the device.

UDP Broadcast:

mac=112233445566 flag=1234 type=***

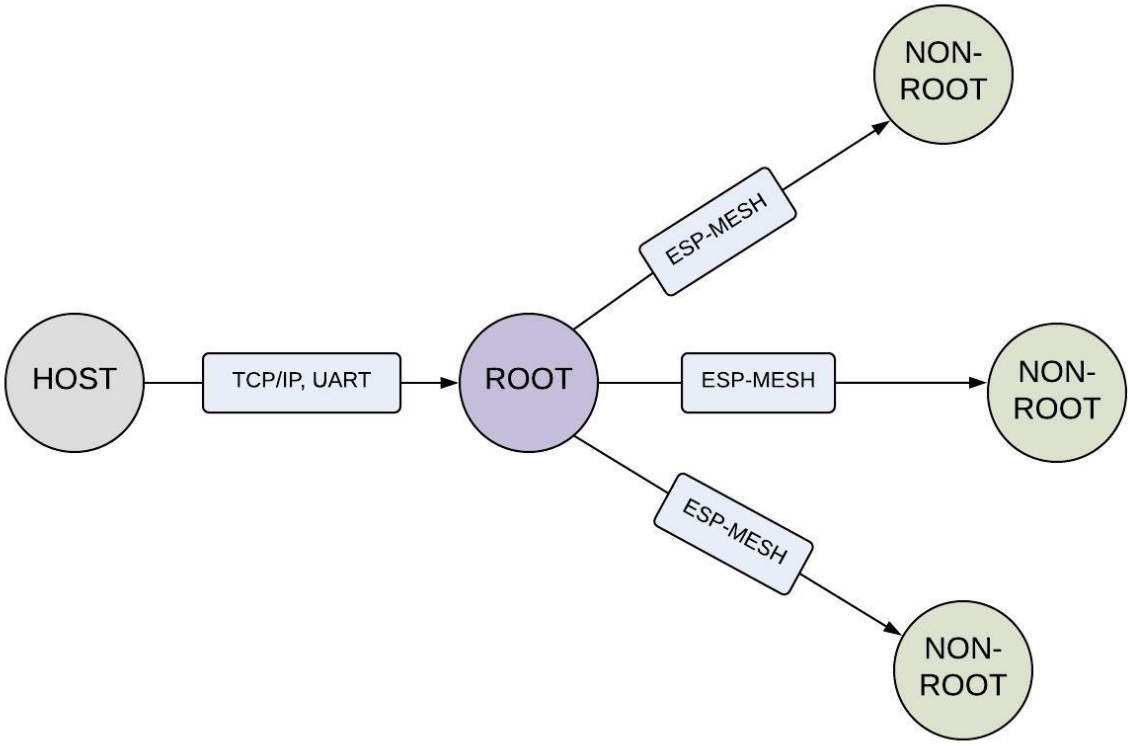
- `mac` is the MAC address of the device whose status has changed;
- `flag` is a random integer value used to distinguish among notifications at different times;
- `type` is the type of change, including:
- `status` indicates that the device status has changed;
- `https` indicates that the information of the device connection in the network has changed, and the updated information is required through https communication protocol;
- `http` indicates that the information of the device connection in the network has changed, and the updated information is required through http communication protocol;
- `sniffer` indicates that a new networked device has been sniffered.

4.5 Mupgrade

[]

Mupgrade, or MESH Upgrade, is a solution for simultaneous over-the-air (OTA) upgrading of multiple ESP-WIFI-MESH devices on the same wireless network by efficient routing of data flows.

Mupgrade downloads a firmware upgrade to a root node, which splits it into fragments and flashes multiple devices with these fragments. When each device receives all the fragments, the mass upgrade is completed.



4.5.1 Functions

- **Automatic retransmission of failed fragments**: The root node splits the firmware into fragments of a certain size and transmits them to the devices that need to be upgraded. The devices write the downloaded firmware fragments to flash and keep log of the process. If the upgrade is interrupted, the device only needs to request the remaining fragments.
- **Data compression**: Miniz is used to compress firmware fragments to reduce their size and, as a result, decrease transmission time.
- **Multicast send**: To prevent redundancy in data transmission during simultaneous upgrade of multiple devices, each device creates a copy of a received firmware fragment and sends it to the next node.
- **Firmware check**: Each firmware fragment contains Mupgrade identification and Cyclic Redundancy Check (CRC) code to avoid such issues as upgrading to wrong firmware versions, transmission errors, and incomplete firmware downloads.
- **Revert to an earlier version**: The device can be reverted to a previous version using specific approaches, such as triggering GPIO, or cutting the power supply and rebooting for multiple times.

4.5.2 Process

1. Download Firmware

a. HOST communicates with the root node through UART or Wi-Fi.

b. HOST transfers the information about firmware, such as length and identification, to the root node.

c. The root node checks if the upgrade of this firmware is supported according to the received firmware information, then erases flash partition and returns the status to HOST.

d. **HOST receives the returned status and acts accordingly:**

- Downloaded: skips Step e and directly goes to Step f.
- Length error: checks if there is a firmware sending error, or if the firmware size exceeds the partition limit.
- Error solved: continues to the next Step.

e. HOST sends the firmware to the root node that writes it directly to flash. When the root node confirms that the received firmware length matches the declared length, it will automatically check the firmware and return its downloading details to HOST.

f. HOST sends to the root node a list of the devices to be upgraded.

Note:

- The partition tables of all the devices to be upgraded must contain: ota_data, ota_0, ota_1. For more information on these partitions, please refer to Over The Air Updates (OTA).
- The firmware size must not exceed the size of its destination partition (ota_0 or ota_1).
- The root node writes the received fragment into flash. At this point, if sending of another packet to the root node as well as all other tasks are not suspended, it may cause data packet loss. So, if UART is used to send the firmware, please enable flow control, or send a firmware fragment and wait for the ACK signal from the root node before sending the next fragment.

---

2. Transfer Firmware

a. The root node receives the device list and verifies it.

b. Then the root node sends the request for upgrade status to the devices on the list.

c. The devices receive the request for upgrade status and check their flash for missing or partially downloaded fragments, erasing such fragments in the process. Then the devices return their reports to the root node.

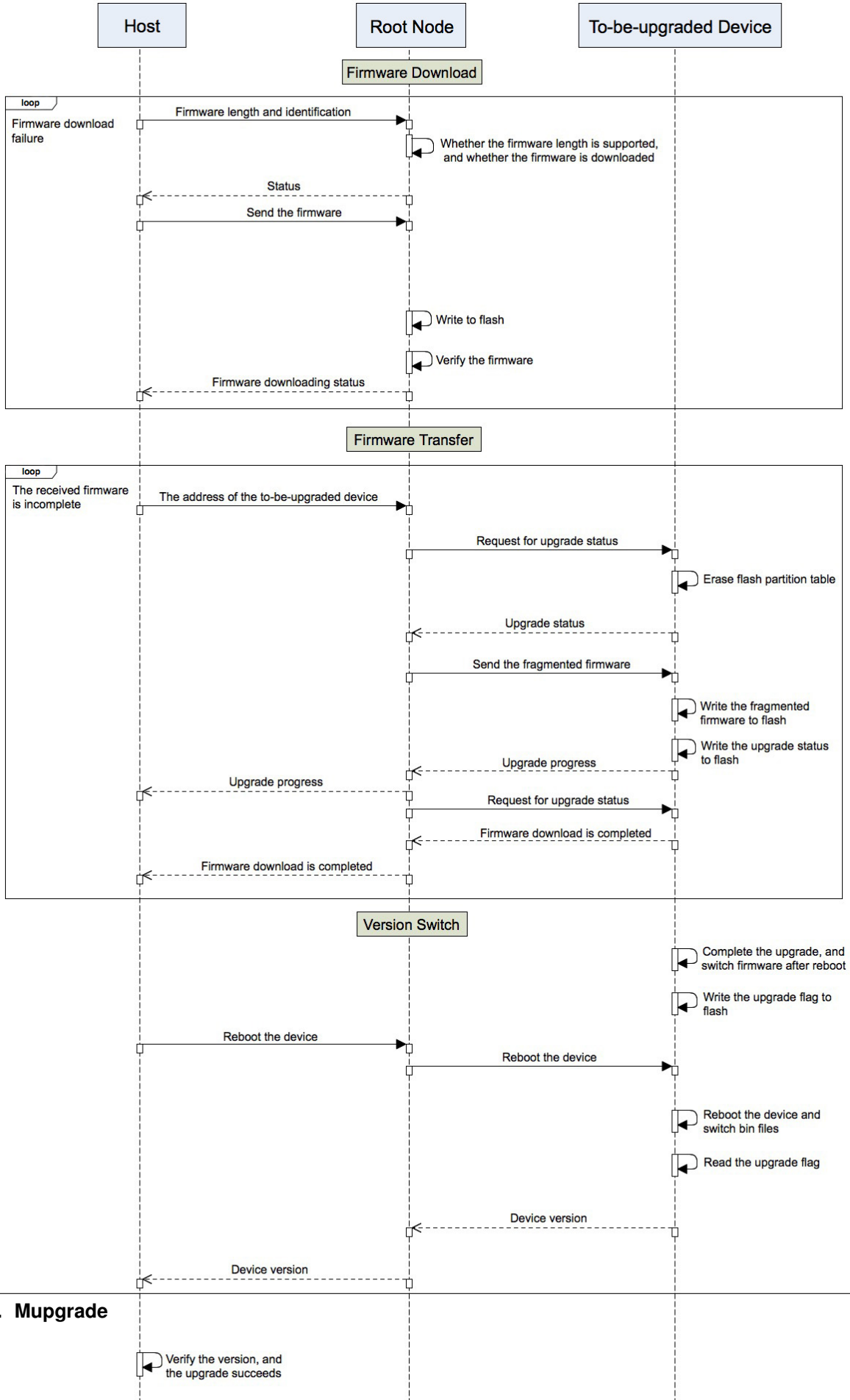
d. The root node transmits the failed firmware fragments to target devices in accordance with their reports.

e. The target devices verify the IDs of the received firmware fragments and write these fragments into flash accordingly, reporting the upgrade status each time the progress increases by 10%.

f. The root node goes back to Step b either until all the devices on the list complete downloading firmware or until the number of cycles reach a user-defined limit.

3. Switch Versions

a. As the devices to be upgraded receive the last firmware fragment, they mark the partition in the ota_data of flash to be run after reboot.


Host
Root Node
To-be-upgraded Device
Firmware Download
loop
Firmware download failure
Firmware length and identification
Whether the firmware length is supported, and whether the firmware is downloaded
Status
Send the firmware
Write to flash
Verify the firmware
Firmware downloading status
Firmware Transfer
loop
The received firmware is incomplete
The address of the to-be-upgraded device
Request for upgrade status
Erase flash partition table
Upgrade status
Send the fragmented firmware
Write the fragmented firmware to flash
Write the upgrade status to flash
Upgrade progress
Upgrade progress
Request for upgrade status
Firmware download is completed
Firmware download is completed
Version Switch
Complete the upgrade, and switch firmware after reboot
Write the upgrade flag to flash
Reboot the device
Reboot the device
Reboot the device and switch bin files
Read the upgrade flag
Device version
Device version
Verify the version, and the upgrade succeeds

b. When the root node receives the information that all the target devices are ready for upgrade, it sends a reboot command.

c. The devices receive the reboot command and report their current version when the reboot is completed.

d. HOST verifies the versions and completes the upgrade.

4.5.3 Partition Table

A partition table defines the flash layout. A single ESP32's flash can contain multiple apps, as well as many different kinds of data. To find more information, please see Partition Tables.

The default partition table in ESP-IDF provides a partition of only 1 MB for apps, which is quite a limited size for ESP-WIFI-MESH application development.

In order to help you configure the partition table, please find two types of partitions below for your reference.

1. Without *factory* partition:

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,   16k
otadata,  data, ota,      0xd000,   8k
phy_init, data, phy,      0xf000,   4k
ota_0,    app,  ota_0,    0x10000,  1920k
ota_1,    app,  ota_1,    ,         1920k
coredump, data, coredump, ,         64K
reserved, data, 0xfe,     ,         128K
```

2. With *factory* partition:

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,   16k
otadata,  data, ota,      0xd000,   8k
phy_init, data, phy,      0xf000,   4k
factory,  app,  factory,  0x10000,  1280k
ota_0,    app,  ota_0,    ,         1280k
ota_1,    app,  ota_1,    ,         1280k
coredump, data, coredump, ,         64K
reserved, data, 0xfe,     ,         128K
```

Note:

1. Before updating the partition table, please erase the entire flash.
2. App partitions (factory, ota_0, ota_1) have to be at offsets aligned to 0x10000 (64K).
3. The partition table cannot be modified wirelessly.
4. The root node uses ota_0 or ota_1 to cache the firmware. The factory partition is used to store backup firmware, without which recovering a device after a fatal error can be much harder.

4.5.4 Notice

If you want to customize the upgrade approach, please keep in mind the following:

- **Do not upgrade from device to device**: It may lead to incompatibility between different versions of devices, which will destroy the original network, create standalone nodes, and increase upgrade difficulties.

- **Do not transmit an entire firmware file**: ESP-WIFI-MESH is a multi-hop network, which means it can only guarantee a reliable transmission from node to node, and NOT end to end. If an entire firmware is attempted to be transmitted in one go, devices located a few nodes away from the root node are very likely to experience data loss, which will immediately cause upgrade failure.

4.6 Mwifi

[]

Mwifi (MESH Wi-Fi) is the most important component for ESP-MDF, for it allows the ESP-WIFI-MESH network to form and to communicate. To further enhance ESP-WIFI-MESH, we have encapsulated native APIs for Mwifi.

4.6.1 Function

- **Network configuration**: We provide a simple-to-use API to configure the root node parameters, network capacity, ESP-WIFI-MESH network stability, data transmission, routing information, and ESP-WIFI-MESH connection parameters;
- **Data transmission by segments**: Data fragmentation is implemented for application layer data, thus allowing the underlying data packet of ESP-WIFI-MESH to be sent by segments;
- **Data compression**: *miniz* is used to compress each data packet, thus reducing the size of the data packet (and speeding up the transmission);
- **Unicast transmission**: After receiving a data packet, the device will send each destination address this data packet, which will then gets to the destination address through multi-hop transmission in the ESP-WIFI-MESH network;
- **Multicast transmission**: After receiving a data packet, the device will send this packet to all the nodes of the next layer, thus reducing data transmission times, in a manner similar to IP multicast;
- **Broadcast transmission**: After receiving a data packet, the device will broadcast this packet to all the neighboring nodes, which will then continue to broadcast this packet to their neighboring nodes;
- **Device grouping**: You can specify a bunch of nodes (of the same type, or those you want to control in one go) in one group, and control all these nodes by using the only unique identifier for the group, which is the group address;
- **Duplicate packet filtering**: When receiving data packets using the API provided by Mwifi, duplicated packets will be filtered out;
- **Secured thread**: A mutex is required before sending any data packets and will be released after the sending, meaning FreeRTOS operating system is supported in this case.

Network configuration

Mwifi provides an API to configure the ESP-WIFI-MESH network for parameters including router SSID and password, MESH ID and password, node type, and MESH channel, etc.

1. Network without a router

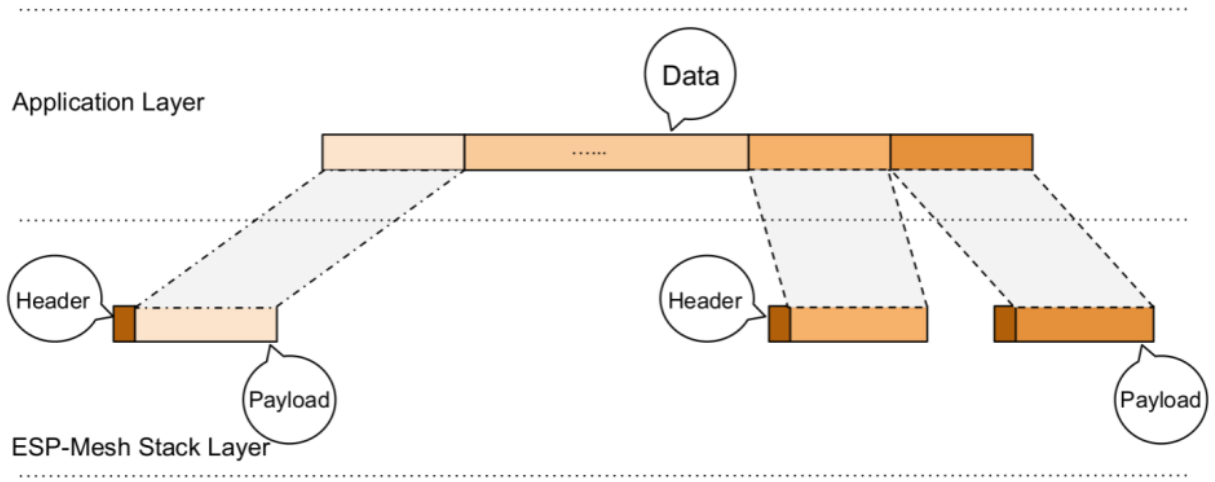
For an ESP-WIFI-MESH network without a router, you can only appoint a node to be the root node. For details, please go to :*example/function_demo/mwifi/no_router*.

2. Network with a router

For an ESP-WIFI-MESH network with a router, you can appoint a node to be the root node, or enable automatic root node election. For details, please go to :*example/function_demo/mwifi/router*.

Data transmission by segments

In some user cases, the data in the application layer may exceed the maximum length of each packet, which calls for data transmission by segments. The maximum length of a data packet in the ESP-MDF application layer is 8095 bytes, while that of a data packet in the ESP-WIFI-MESH underlying protocol stack is 1472 bytes.



Data compression

Considering the amount of data will be relatively large in the application layer, application layer data is compressed before transmission, and decompressed upon receiving. This is possible because of the *miniz* data compression library.

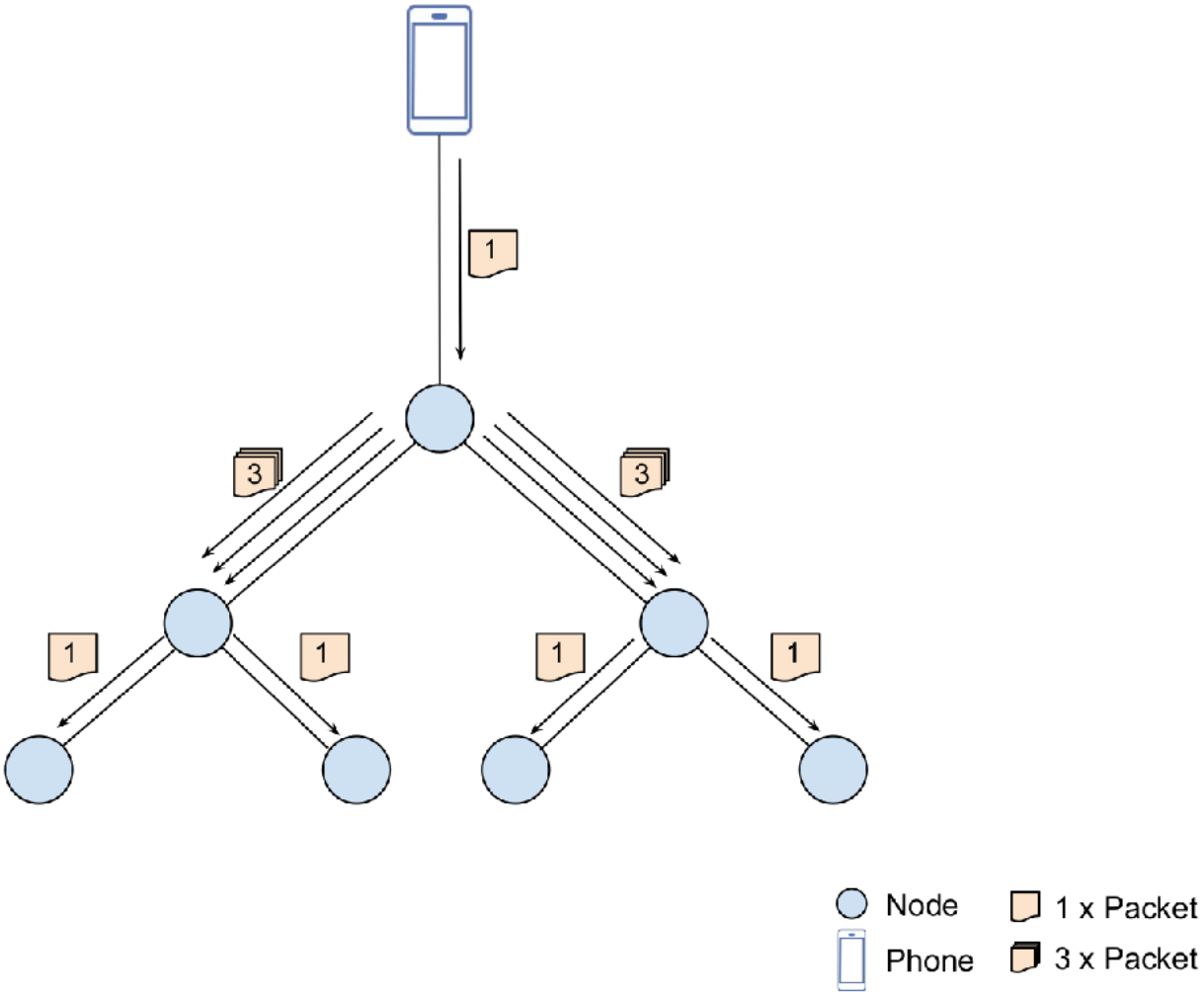
- Time: 2 ~ 4 ms for compression, 1 ~ 2 ms for decompression
- Memory requirement: The task stack should be not less than 8 KB; The dynamic memory should be larger than 7 KB for compression and 2 KB for decompression.
- Compression ratio: The compression ratio varies dramatically with the data content. The compression ratio increases when there are more repetitive content. This is applicable to some plain text formats, such as json.

Unicast transmission

As shown, in the unicast transmission mode, the data sent by the device will traverse to the destination address, through multi-hop transmission, in the MEHS network. In one-to-more communication

---

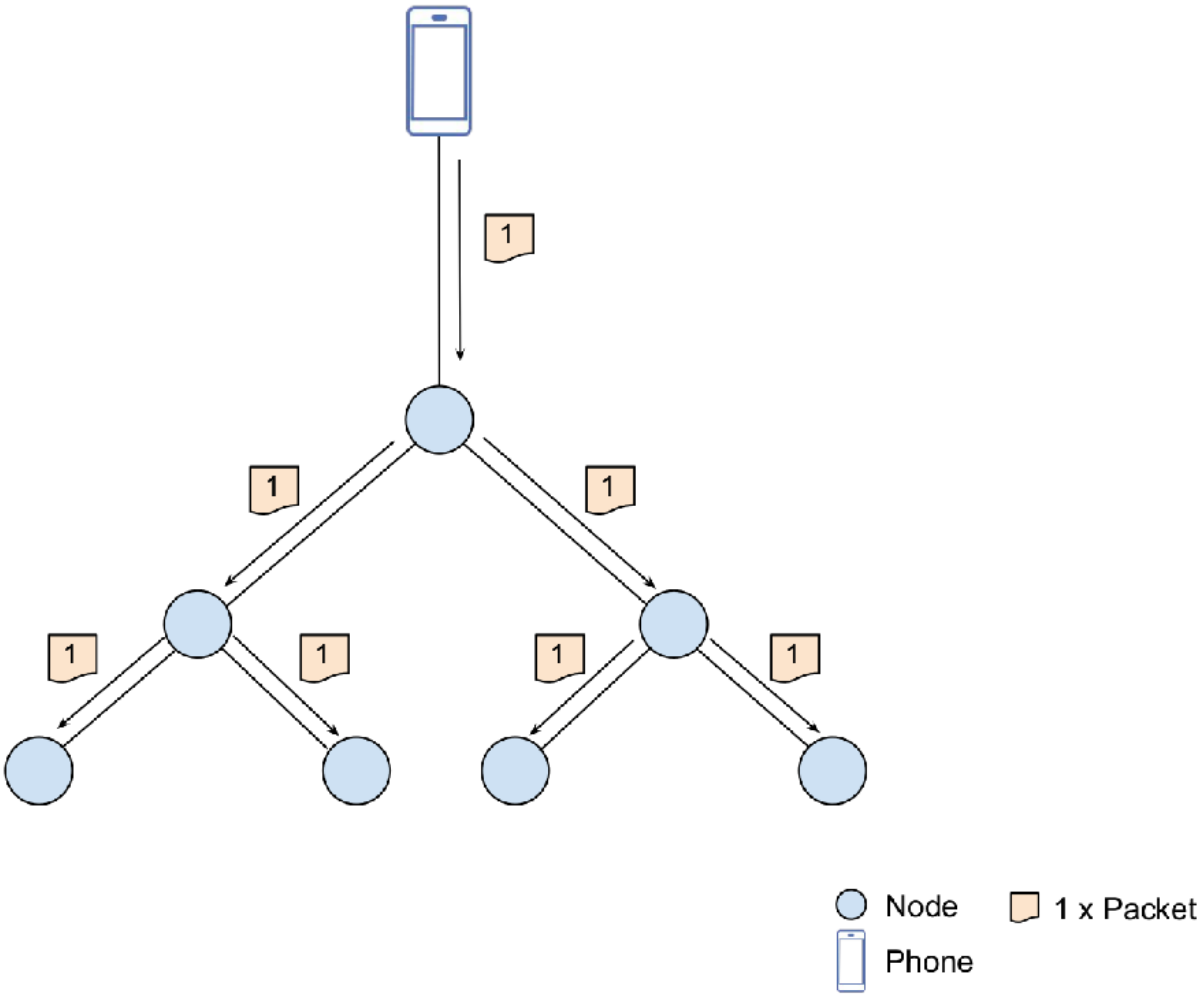
Note: Flow control is added for up-stream transmission (from a child node to a root node). Therefore, the transmission can be blocked and you need to wait for the completion of the sending or the returning of an error.


1
3
3
1
1
1
1
Node
1 x Packet
Phone
3 x Packet

Multicast transmission

In actual user cases, some applications require "one- to more-nodes" data transmission, which means a source node needs to send a packet to more than one destination nodes. These users cases can be, for example, turning off all the lightings in one house. Multicast transmission can significantly reduce the consumption of various resources in the network, compared with unicast mode.

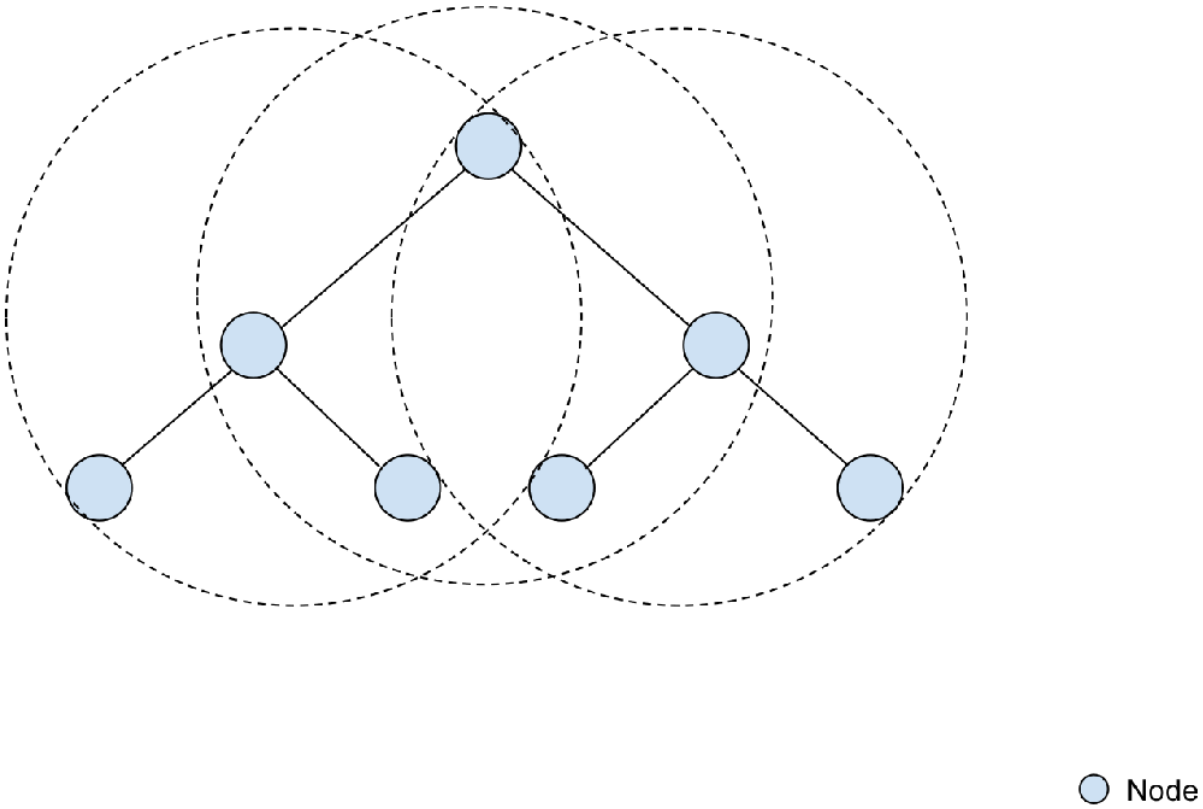
As can be seen from the figure, in the unicast mode, the source node needs to transmit the data packet to all the destination nodes, which requires n unicasts (meaning sending n copies of the same packet). However, in the multicast mode, the source node simply needs to transmit the data packet once to the root node. Then, the root node will relay this packet upon receipt to all nodes in the next layer, which will then repeat the same behavior (relaying the packet to nodes of the next layer). Eventually, all the nodes in the ESP-WIFI-MESH network are able to receive the packets from the source node.



Broadcast transmission

As introduced in the above section, in user cases that require one- to more-nodes communication, multicast transmission is preferred because it can significantly reduce the consumption of various resources in the network. However, broadcast transmission can be a good choice for cases that have a lower tolerance for time delay, a higher requirement for transmission speed, but can tolerate higher packet loss, such as audio streaming.

When using the broadcast transmission, device A will broadcast data packets to all its neighboring nodes, which will then broadcast the data packets as well, upon receipt, to their neighboring nodes. Note that, device A, which is the original source node, also receives the same broadcast data packet from device B. However, device A then finds out the received packet is the same with what it sent out earlier, thus it will not broadcast this packet.



Device grouping

In typical user cases of ESP-WIFI-MESH, there usually exist plenty of devices with similar functionality in the same ESP-WIFI-MESH network. Therefore, users may consider grouping these devices, thus controlling these devices by groups.

The root node sends data packets to all its child nodes, but only the nodes in the destination group can parse the packets.

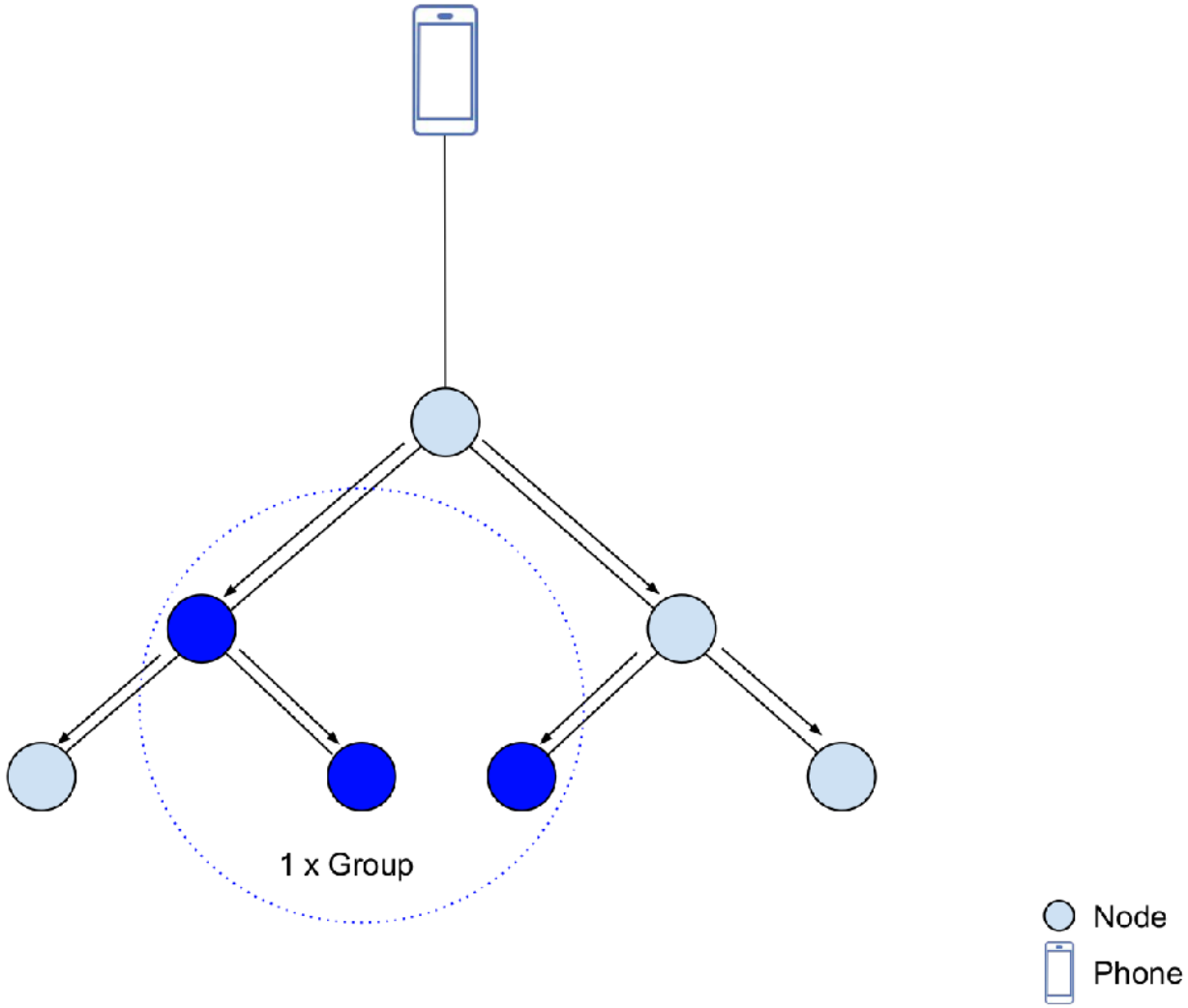
4.7 Mdebug

[]

Mdebug (Mesh Network debug) is an important debugging solution used in ESP-MDF. It is designed to efficiently obtain ESP-MDF device logs through wireless espnow protocol, TCP protocol, serial port, etc., so that it can be read more conveniently and quickly. Then the device log information can be analyzed according to the extracted logs.

The Mdebug guide is divided into the following forms:

1. *Introduction*
2. *Functions*

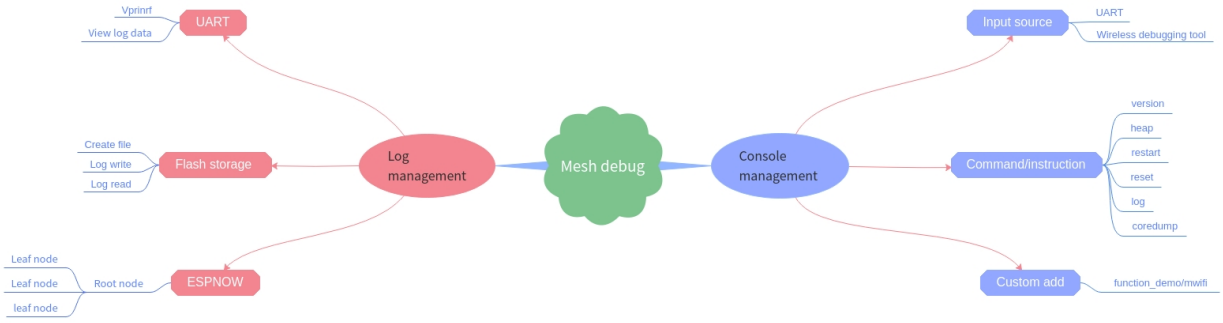

1 x Group
Node
Phone

3. *Debug Method*
4. *Command Management*
5. *Log Management*

4.7.1 Introduction

The traditional debugging method is to read all the log information by connecting the serial port. Some of the log information is unnecessary information, which wastes a lot of filtering time of the user, and needs to be connected online through the PC and the device to collect log information, resulting in the log information. A waste of time and resources.

The difference between the Mdebug debugging method and the traditional debugging method is that, while not affecting the operation of the normal device, the Mdebug debugging can filter the log through wireless debugging, log storage, and control commands, thereby improving the efficiency and convenience of the log. Save time and resources for devices to find problems and read the information they need.



Mdebug is mainly divided into command management and log management

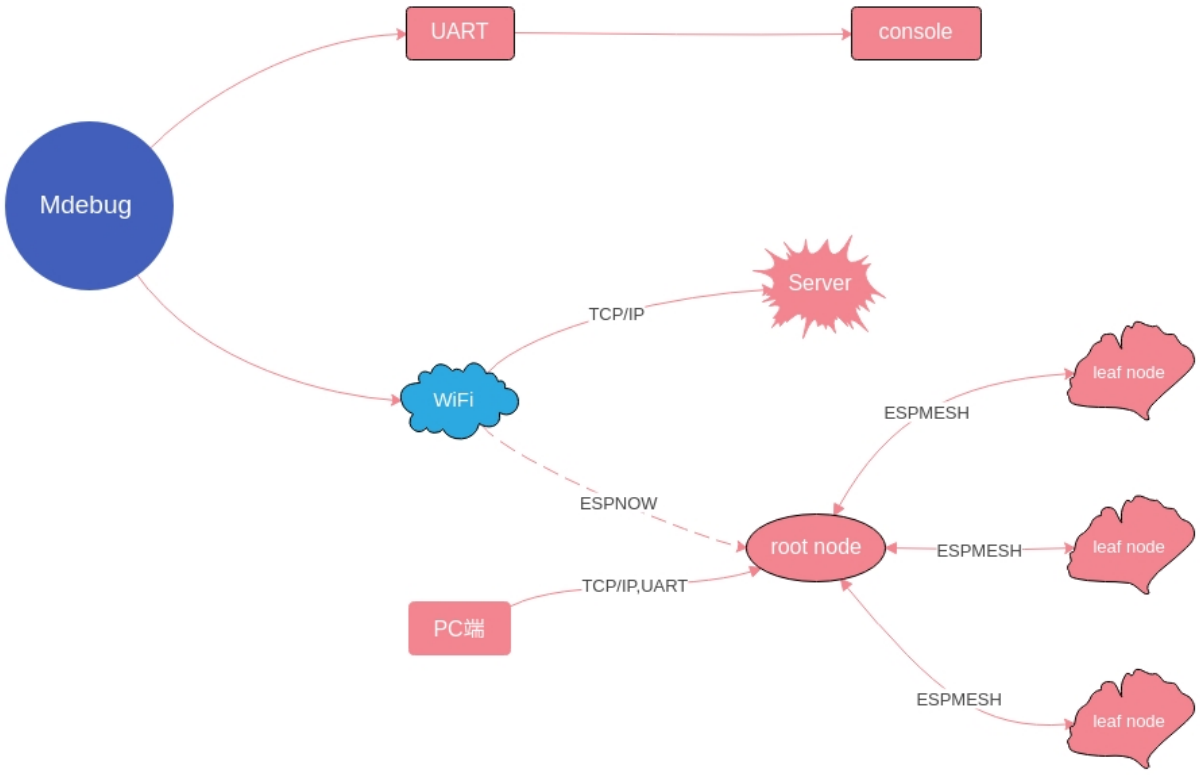
1. Command management can be roughly divided into *input source*, *command/instruction*, *user-defined add*;
2. Log management can be roughly divided into *UART*, *Flash storage*, *ESPNOW*.

4.7.2 Functions

- `Mdebug espnow`: Transfer log data from the device to other devices or servers via `ESPNOW` wireless WiFi form, thus improving the speed and convenience of debugging.
- `Mdebug flash`: The log data in the device is directly written into the flash memory. The memory space has been created in the flash memory. Power loss protection log information, and the log information can be read at any time, thereby improving the availability of data and the convenience of the customer.
- `Mdebug log`: The log data in the device is read out through the serial port monitor, and the log information such as *MDF_LOGI, MDF_LOGD, MDF_LOGW, MDF_LOGE* in *ESP_MDF* is read out.
- `Mdebug console`: The log data in the device is read through the input command line in the terminal, which improves the operability and practicability of the user.

4.7.3 Debug Method

The debugging method can be summarized as two methods.One is UART serial port debugging, and the other is wireless WiFi debugging.



1. Serial port debugging uses the serial port debugging tool on the PC port (use minicom under linux), directly obtains the output log information, and can output the required log information by controlling the command line debugging method.
2. The wireless WiFi debugging has two communication modes: TCP/IP protocol and ESPNOW protocol. One is that the log information is transmitted to the Sever end through the TCP/IP protocol, and the other is that the log information from the child node is transmitted to the root through the ESPNOW protocol. The node then extracts log information from the serial port of the root node or TCP/IP.

4.7.4 Command Management

1. Input source

- **serial** PC monitor
- **ESPNOW** Wireless debugging tool (ESP-WROVER-KIT-V2

2. Command and operation

2.1 General instruction

- **help** Print registered commands and their descriptions
- **version** Get the chip and SDK version
- **heap** Get the current available heap memory size
- **restart** Soft restart chip

- **reset** Clear all configuration information of the device

2.2 Log command

Command definition	log -or [<tag>] [<level>] [-s <addr(xx:xx:xx:xx:xx:xx)>] [-e <enable_type ('uart'or'flash'or'espnow')>] [-d <disable_type('uart'or'flash'or'espnow')>]	
Instruction	log -o	Get the log enable status
	log -r	Read the log information
	log -s	Send logs to the specified device
Parameter	tag	Use tag to filter logs
	level	Use level to filter logs
	addr	Monitoring device MAC address
	e 'uart' or 'flash' or 'espnow'	Enable serial port, flash, espnow
	d 'uart' or 'flash' or 'espnow'	Disable serial port, flash, espnow
Examples	log mdebug_cmd INFO	Set the log output level of TAG to mwifi to INFO
	log * NONE	Set all logs to not output

2.3 Coredump command

Command definition	coredump [-loe] [-q] [-s <addr (xx:xx:xx:xx:xx:xx)>]	
Instruction	coredump -l	Get the length of the coredump data on the device
	coredump -o	Read the coredump data on the device and print it to the console
	coredump -e	Erase the coredump data on the device
	coredump -s	Send coredump data from the device to the specified device
Parameter	addr	Monitoring device MAC address
	sequence	The serial number of the coredump data
Examples	coredump -s 30:ae:a4:00:4b:90	Send coredump data to 30:ae:a4:00:4b:90 device
	coredump -q 110 -s 30:ae:a4:00:4b:90	Send the coredump data starting with sequence number 110 to the 30:ae:a4:80:16:3c device

3. Custom add

Users can see how to customize the addition according to example:*function_demo/mwifi/console_test* in ESP-MDF. Add the features you want to suit your needs.

4.7.5 Log Management

Mdebug can be roughly divided into two forms according to the way the log is written:

1. The log information of the device is printed directly from the serial port by printing or the log information is stored, and then the read is called. The storage of the log information first writes the device log to the flash memory (here, a partition is allocated as *storage* in the flash memory, in order to store the device log, but the allocated memory here is limited, according to the file size set by the user. The decision) will be temporarily stored in the form of a file, and then the data will be sent to the PC or the server in the form of a packet through a serial port or wirelessly;
2. The device will send log information in the form of espnow. The child node log information is sent to the root node through the ESP-MESH network, and the log information is read from the device of the root node.

According to the log reading mode, there are three enabled states in the log, namely `uart`, `flash`, and `espnow`.

1. UART enable

The serial port is enabled and the log information will be printed out via `vprintf`.

1. **The I/O port for reading the log is UART0. The pin of the serial port is TXD0, which is GPIO1, RXD0 is GPIO3, and it is**

UART0	
TXDO	RXD0
GPIO1	GPIO3

2. Read the log information, diagnose the problem and find the problem. If the device is running normally, you can disable the serial port. If you do not close the serial port, it will occupy the memory. At the same time, the serial port prints too much information and activates. The watchdog makes the normal device run the program `Backtrace`.
3. The default state of the serial port is enabled. You can turn it off according to your needs.

2. Flash enable

Write flash enable to store log information in flash.

2.1 Save the log to the flash

The certain memory space selected in the partition table is `storage`, and the memory allocated here provides memory space for writing the log to the flash. The file name is `spiffs`,

spiffs partition:

```
# Name,   Type, SubType,  Offset,   Size,  Flags
nvs,      data, nvs,      0x9000,   16k
otadata,  data, ota,      0xd000,   8k
phy_init, data, phy,      0xf000,   4k
ota_0,    app,  ota_0,    0x10000,  1920k
```

(continues on next page)

(continued from previous page)

```
ota_1,    app,  ota_1,    ,         1920k
coredump, data, coredump, ,         64K
storage,  data, spiffs,   ,         64K
reserved, data, 0xfe,     ,         64K
```

Note:

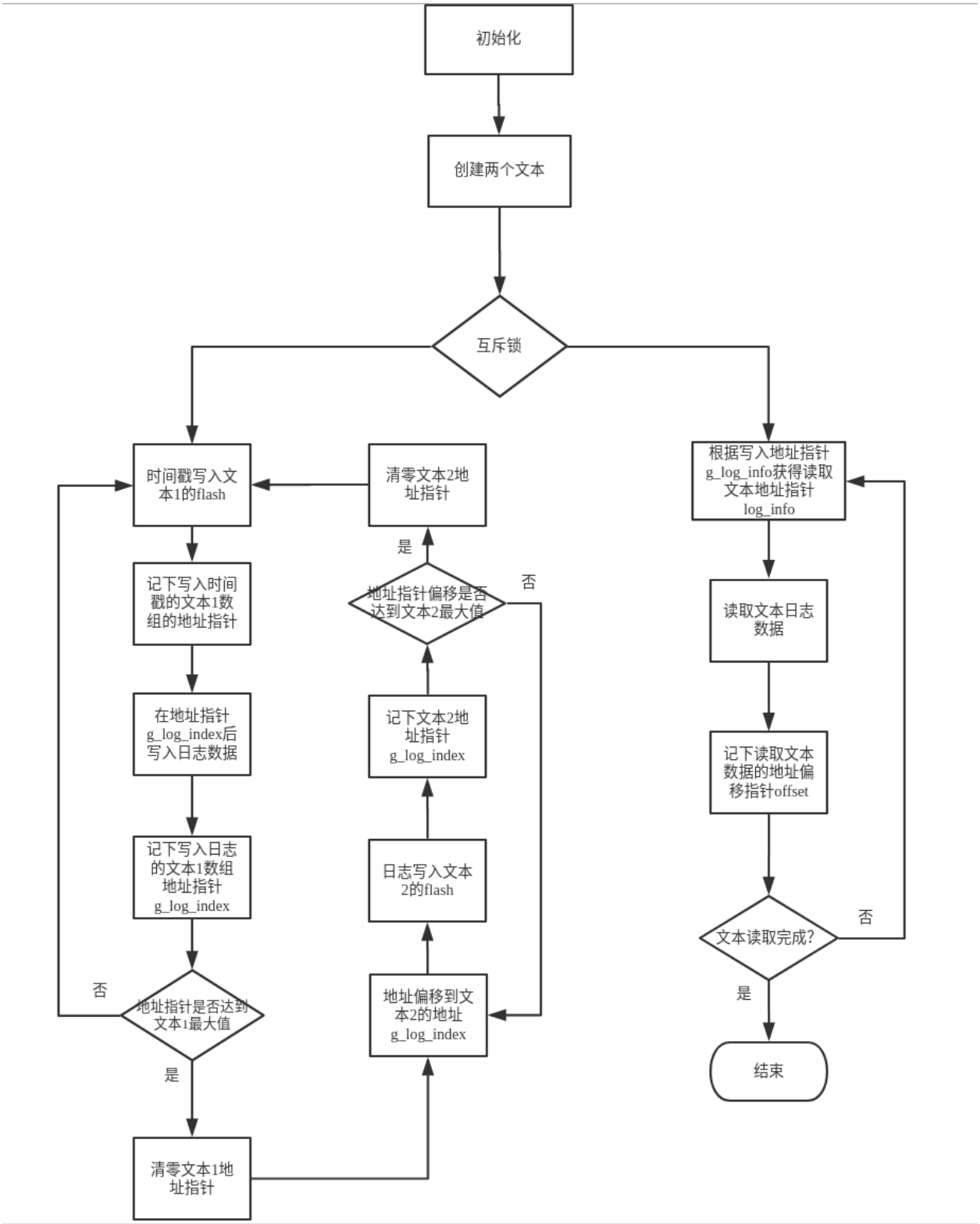
1. Before updating the partition table, you need to erase the entire flash at first;
2. The partition table cannot be modified by OTA;
3. The size of the file space can be allocated according to the user's reasonable choice.

2.2 Log information access

1. Log initialization, create two text flash memory space, and get the state of the text `stat`, determine whether to write or read, use the main function is `esp_vfs_spiffs_register`, `esp_spiffs_info`, `Sprintf`, `fopen`;
2. Add a mutex to the access. When the log information is read, the write function will be disabled. The main functions used are `xSemaphoreTake`, `xSemaphoreGive`;
3. Write to flash, first write the timestamp, and write down the address pointer *g_log_info[g_log_index]* of the text array. The main functions used are `time`, `localtime_r`, `strftime`;
4. Write the log data, and write down the address pointer `g_log_index` of the array of text 1 to write the address for the next log write flash. The main functions are `fseek`, `fwrite`;
5. Judge, if the address pointer of the text 1 array is full, clear the address pointer of text 1, the address is offset to the address pointer of text 2, and start writing log data; if it is not full, it will continue to write in text 1. Log data
6. Similarly, when the address pointer of the text 2 array is full, the address pointer of the text 2 is cleared, the address is offset to the address pointer of the text 1, and the log data is started to be written; if it is not full, it will continue in the text 2 Write log data;
7. Obtain the read text address pointer *log_info* according to the write address pointer *g_log_info*, then read the log data in the text, and also write down the address offset pointer *offset* of the read text data, for the next time from flash Read the log for address addressing, using the main functions as `fseek`, `fread`;
8. Judge that if the text is not read, the text log data will continue to be read; if the text is read, the read task will end.

Note:

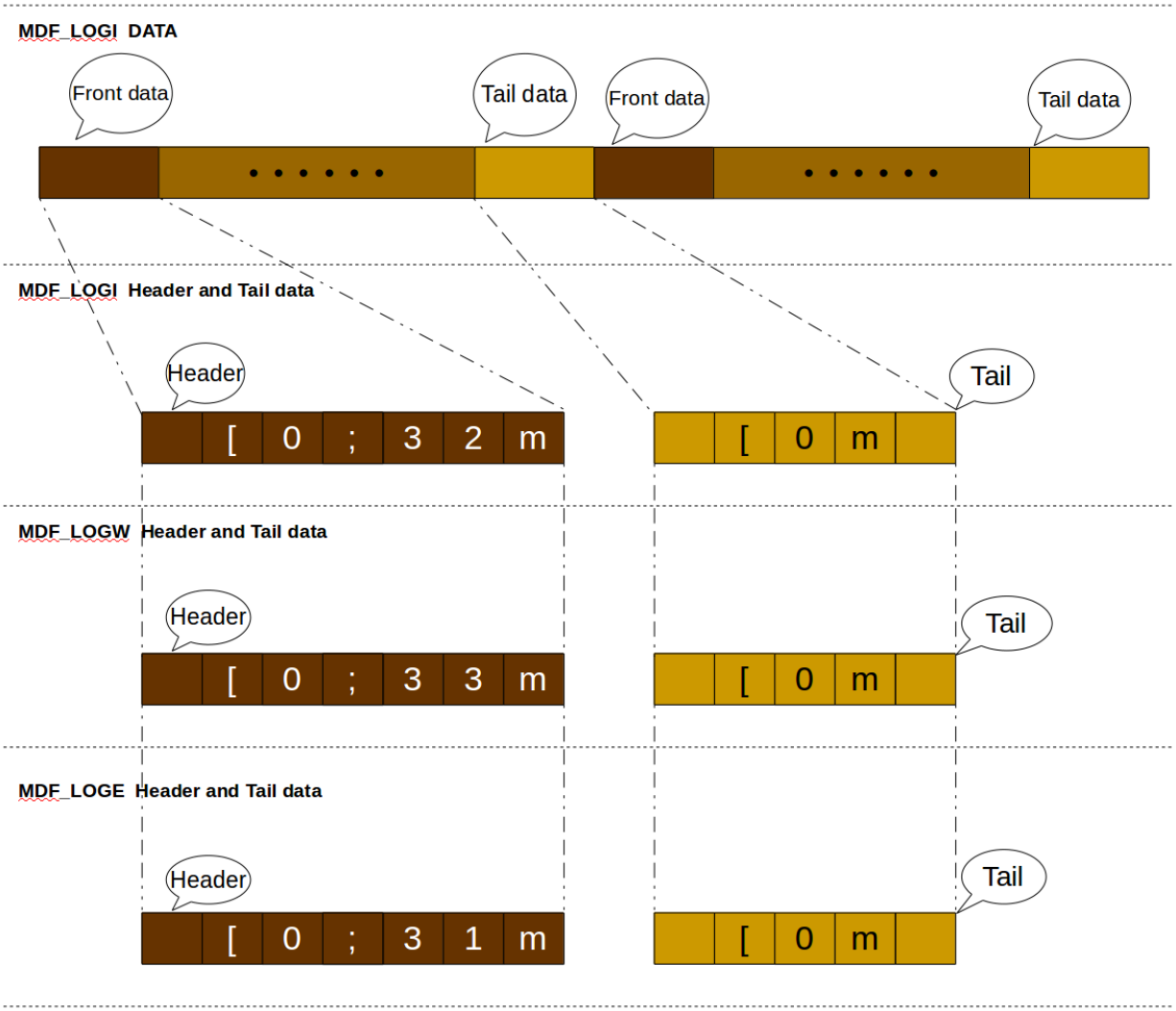
1. The header of the log data is added with a timestamp. It is only used as an experiment, and there is no real-time calibration. The user can modify it according to his own needs.
2. The file size of the log storage is CONFIG_MDEBUG_FLASH_FILE_MAX_SIZE = 16384. You can modify the storage space of the log file according to your needs.
3. `Log Redirection` The log storage information output is re-defined. This is to debug the log write flash. When the log output information has a problem, the log output information can be better debugged. The debugging function is `MDEBUG_PRINTF(fmt, ...);`


初始化
创建两个文本
互斥锁
时间戳写入文本1的flash
记下写入时间戳的文本1数组的地址指针
在地址指针g_log_index后写入日志数据
记下写入日志的文本1数组地址指针g_log_index
地址指针是否达到文本1最大值
否
是
清零文本1地址指针
清零文本2地址指针
是
地址指针偏移是否达到文本2最大值
否
记下文本2地址指针g_log_index
日志写入文本2的flash
地址偏移到文本2的地址g_log_index
根据写入地址指针g_log_info获得读取文本地址指针log_info
读取文本日志数据
记下读取文本数据的地址偏移指针offset
文本读取完成?
否
是
结束

4. Added data erasure. When the data is full, the data pointer will be cleared, and the main function `rewind` used will be restarted from the file header pointer address.

2.3 Log data format

The log data will come from *MDF_LOGI, MDF_LOGD, MDF_LOGW, MDF_LOGE*, etc. in *ESP_MDF*. This is because the IDF's log library will use the function of class vprintf to output the formatted string to the dedicated UART by default. The extracted data is shown below:



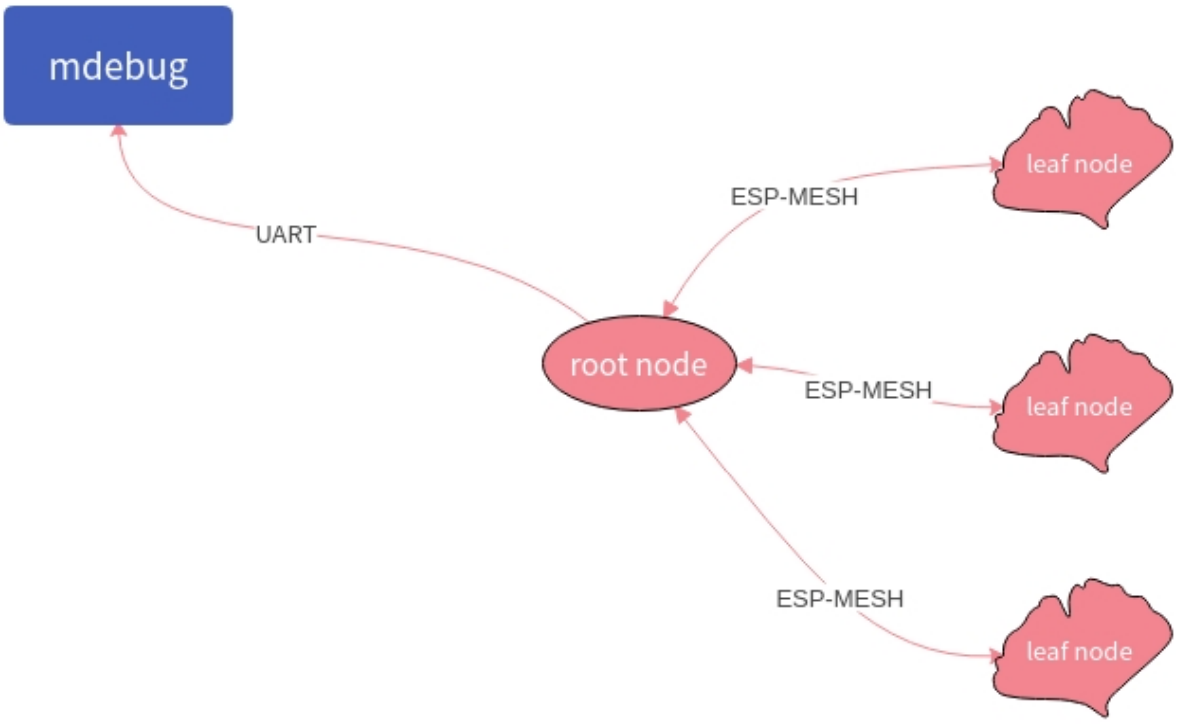
Since the log information in the MDF has unnecessary data at the beginning and the end, it is necessary to extract and select valid string data information, so it needs to be removed and filtered, and then the log data is extracted. `Front data` contains the added information such as the font color, so the header part data needs to be removed, and `Tail data` contains the data such as line breaks, which also needs to be removed. There is no such useless data for *MDF_LOGD*, so no processing is required.

3. ESPNOW enable

3.1 ESP-NOW Features

- Both sender and receiver must be on the same channel
- The receiving end may not add the MAC address of the sender in the case of non-encrypted communication (addition required for encrypted communication), but the sender must add the MAC address of the receiver.
- ESP-NOW can add up to 20 paired devices and support up to 6 devices for communication encryption
- Receive packets by registering a callback function, and check the delivery (success or failure)
- Secure data with CTR and CBC-MAC protocol (CCMP)

3.2 ESP-NOW enable process



By enabling *espnow*, the log data of the child node can be sent to the root node through espnow, so that the log information of the child node is read from the root node, and then the log information is read through the serial port. For more on *espnow* see example:*wireless_debug*.

```
MDF_ERROR_ASSERT(mdebug_console_init());
MDF_ERROR_ASSERT(mdebug_espnow_init());
mdebug_cmd_register_common();
```

For more *espnow* see example:*wireless_debug* and the official documentation `espnow`.

Note: Because ESP-NOW is the same as ESP-MESH, it sends and receives data packets through the Wi-Fi interface. Therefore, when the ESP-MESH device has a large amount of data transmission, it will generate some delay for its

control command reception or data transmission. After actual testing, the ESP-MESH device delay caused by the following configuration parameters is a negligible threshold when the network environment is good:

- 50 ESP-MESH devices (the more the number of devices, the worse the network environment)
- Add 10 ESP-MESH devices to the ESP-NOW receiving end (the more the receiving end is added, the worse the network environment)
- The transfer log level is info (the lower the log level, the worse the network environment)

CHAPTER 5

ESP-MDF Contributions Guide

[]

This guide introduces how to add code to ESP-MDF. For details, please refer to Contributions Guide.

5.1 Add Code

When you add new functions to ESP-MDF, please add the corresponding examples as well to the directory `examples`. Also, please add two variables `ARTIFACTS_NAME` and `EXAMPLE_PATH` to the file `.gitlab-ci.yml`, and test the examples in `gitlab`. Please refer to the `get_started` example below:

```
build_example_get_started:
  <<: *build_template
  variables:
    ARTIFACTS_NAME: "get_started"
    EXAMPLE_PATH: "examples/get_started"
```

5.2 Format Code

Make sure to format the code prior to your submission, with either one of the formatting tools for ESP-MDF: astyle and dos2unix. Please install such tools first. For the installation on Linux, please refer to the command below:

```
sudo apt-get install astyle
sudo apt-get install dos2unix
```

Run the code formatting command below:

```
tools/format.sh new_file.c
```

5.3 Generate API Documentation

For the code comments, please follow the Doxygen Manual. You may use `Doxygen` to automatically generate API documentation according to the following steps:

```
sudo apt-get install doxygen
export MDF_PATH=~/esp/esp-mdf
cd ~/esp/esp-mdf/docs
pip install -r requirements.txt
cd ~/esp/esp-mdf/docs/en
make html
```

CHAPTER 6

ESP-MDF Versions

The ESP-MDF GitHub repository is updated regularly, especially on the "master branch" where new development happens. There are also stable releases which are recommended for production use.

6.1 Releases

Documentation for the current stable version can always be found at this URL:

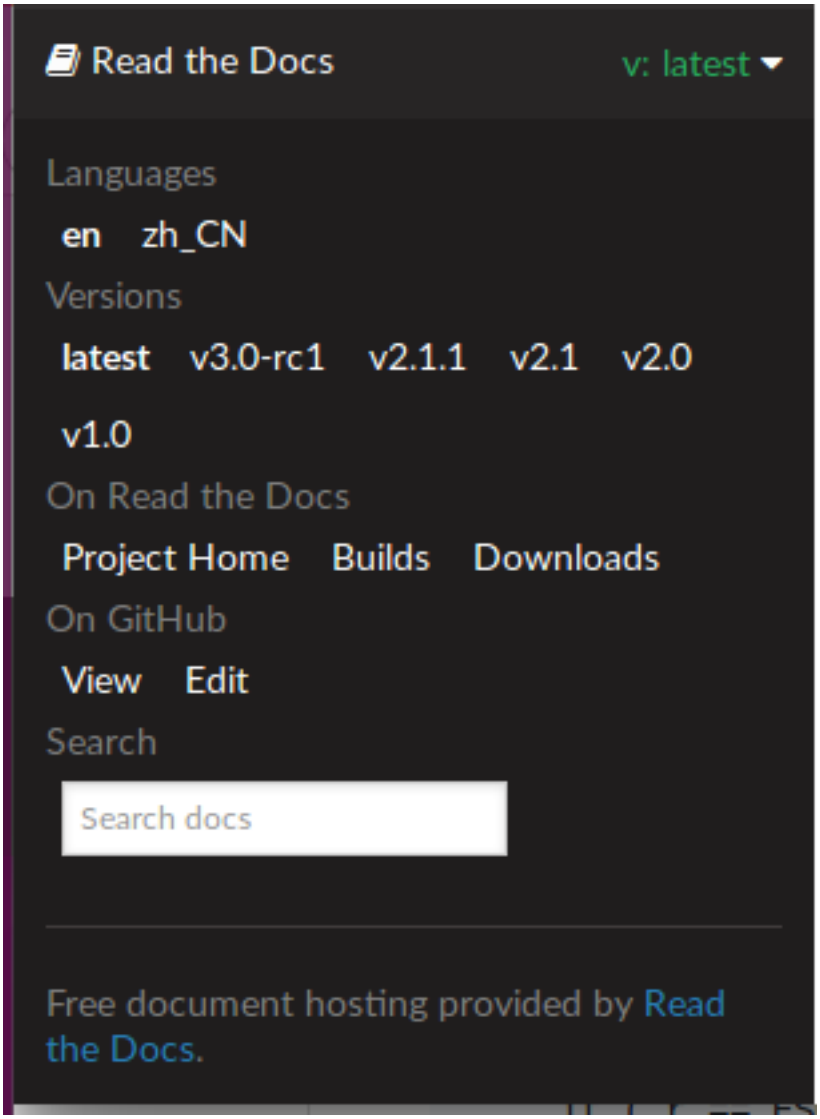
https://docs.espressif.com/projects/esp-mdf/en/stable/

Documentation for the latest version ("master branch") can always be found at this URL:

https://docs.espressif.com/projects/esp-mdf/en/latest/

The full history of releases can be found on the GitHub repository Releases page. There you can find release notes, links to each version of the documentation, and instructions for obtaining each version.

Documentation for all releases can also be found in the HTML documentation by clicking the "versions" pop up in the bottom-left corner of the page. You can use this popup to switch between versions of the documentation.



6.2 Which Version Should I Start With?

- For production purposes, use the current stable version. Stable versions have been manually tested, and are updated with "bugfix releases" which fix bugs without changing other functionality (see *Versioning Scheme* for more details).
- For prototyping, experimentation or for developing new ESP-MDF features, use the latest version (master branch in Git). The latest version in the master branch has all the latest features and has passed automated testing, but has not been completely manually tested ("bleeding edge").
- If a required feature is not yet available in a stable release, but you don't want to use the master branch, it is possible to check out a pre-release version or a release branch. It is recommended to start from a stable version and then follow the instructions for *Updating to a Pre-Release Version* or *Updating to a Release Branch*.

See *Updating ESP-MDF* if you already have a local copy of ESP-MDF and wish to update it.

6.3 Versioning Scheme

ESP-MDF uses Semantic Versioning. This means:

- Major Releases like `v3.0` add new functionality and may change functionality. This includes removing deprecated functionality.

  When updating to a new major release (for example, from `v2.1` to `v3.0`), some of your project's code may need updating and functionality will need to be re-tested. The release notes on the Releases page include lists of Breaking Changes to refer to.

- Minor Releases like `v3.1` add new functionality and fix bugs but will not change or remove documented functionality, or make incompatible changes to public APIs.

  If updating to a new minor release (for example, from `v3.0` to `v3.1`) then none of your project's code should need updating, but you should re-test your project. Pay particular attention to items mentioned in the release notes on the Releases page.

- Bugfix Releases like `v3.0.1` only fix bugs and do not add new functionality.

  If updating to a new bugfix release (for example, from `v3.0` to `v3.0.1`), you should not need to change any code in your project and should only need to re-test functionality relating directly to bugs listed in the release notes on the Releases page.

6.4 Checking The Current Version

The local ESP-MDF version can be checked using git:

```
cd $MDF_PATH
git describe --tags --dirty
```

The version is also compiled into the firmware and can be accessed (as a string) via the macro `MDF_VER`. The default ESP-MDF bootloader will print the version on boot (these versions in code will not always update, it only changes if that particular source file is recompiled).

Examples of ESP-MDF versions:

Version String	Meaning
`v0.5.0`	Master branch pre-release, in development for version 3.2. 306 commits after v3.2 development started. Commit identifier `beb3611ca`.

6.5 Git Workflow

The development (Git) workflow of the Espressif ESP-MDF team is:

- New work is always added on the master branch (latest version) first. The ESP-MDF version on `master` is always tagged with `-dev` (for "in development"), for example `v3.1-dev`.
- Changes are first added to an internal Git repository for code review and testing, but are pushed to GitHub after automated testing passes.

- When a new version (developed on `master`) becomes feature complete and "beta" quality, a new branch is made for the release, for example `release/v3.1`. A pre-release tag is also created, for example `v3.1-beta1`. You can see a full list of branches and a list of tags on GitHub. Beta pre-releases have release notes which may include a significant number of Known Issues.
- As testing of the beta version progresses, bug fixes will be added to both the `master` branch and the release branch. New features (for the next release) may start being added to `master` at the same time.
- Once testing is nearly complete a new release candidate is tagged on the release branch, for example `v3.1-rc1`. This is still a pre-release version.
- If no more significant bugs are found or reported then the final Major or Minor Version is tagged, for example `v3.1`. This version appears on the Releases page.
- As bugs are reported in released versions, the fixes will continue to be committed to the same release branch.
- Regular bugfix releases are made from the same release branch. After manual testing is complete, a bugfix release is tagged (i.e. `v3.1.1`) and appears on the Releases page.

6.6 Updating ESP-MDF

Updating ESP-MDF depends on which version(s) you wish to follow:

- *Updating to Stable Release* is recommended for production use.
- *Updating to Master Branch* is recommended for latest features, development use, and testing.
- *Updating to a Release Branch* is a compromise between these two.

Note: These guides assume you already have a local copy of ESP-MDF.

6.6.1 Updating to Stable Release

To update to new ESP-MDF releases (recommended for production use), this is the process to follow:

- Check the Releases page regularly for new releases.
- When a bugfix release for a version you are using is released (for example if using `v3.0.1` and `v3.0.2` is available), check out the new bugfix version into the existing ESP-MDF directory:

```
cd $MDF_PATH
git fetch
git checkout vX.Y.Z
git submodule update --init --recursive
```

- When major or minor updates are released, check the Release Notes on the releases page and decide if you would like to update or to stay with your existing release. Updating is via the same Git commands shown above.

Note: If you installed the stable release via zip file rather than using git, it may not be possible to change versions this way. In this case, update by downloading a new zip file and replacing the entire `MDF_PATH` directory with its contents.

6.6.2 Updating to a Pre-Release Version

It is also possible to `git checkout` a tag corresponding to a pre-release version or release candidate, the process is the same as *Updating to Stable Release*.

Pre-release tags are not always found on the Releases page. Consult the list of tags on GitHub for a full list. Caveats for using a pre-release are similar to *Updating to a Release Branch*.

6.6.3 Updating to Master Branch

Note: Using Master branch means living "on the bleeding edge" with the latest ESP-MDF code.

To use the latest version on the ESP-MDF master branch, this is the process to follow:

- Check out the master branch locally:

```
cd $MDF_PATH
git checkout master
git pull
git submodule update --init --recursive
```

- Periodically, re-run `git pull` to pull the latest version of master. Note that you may need to change your project or report bugs after updating master branch.
- To switch from `master` to a release branch or stable version, run `git checkout` as shown in the other sections.

Important: It is strongly recommended to regularly run `git pull` and then `git submodule update --init --recursive` so a local copy of `master` does not get too old. Arbitrary old master branch revisions are effectively unsupportable "snapshots" that may have undocumented bugs. For a semi-stable version, try *Updating to a Release Branch* instead.

6.6.4 Updating to a Release Branch

In stability terms, using a release branch is part-way between using `master` branch and only using stable releases. A release branch is always beta quality or better, and receives bug fixes before they appear in each stable release.

You can find a list of branches on GitHub.

For example, to follow the branch for ESP-MDF v3.1, including any bugfixes for future releases like `v3.1.1`, etc:

```
cd $MDF_PATH
git fetch
git checkout release/v3.1
git pull
git submodule --update --init --recursive
```

Each time you `git pull` this branch, ESP-MDF will be updated with fixes for this release.

Note: The is no dedicated documentation for release branches. It is recommended to use the documentation for the closest version to the branch which is currently checked out.

CHAPTER 7

Resources

- The esp32.com forum is a place to ask questions and find community resources.
- This ESP Mesh Development Framework inherits from ESP IoT Development Framework and you can learn about it in ESP-IDF documentation.
- Check the Issues section on GitHub if you find a bug or have a feature request. Please check existing Issues before opening a new one.
- If you're interested in contributing to ESP Mesh Development Framework, please check the Contributions Guide.
- Several books have been written about ESP32 and they are listed on Espressif web site.
- For additional ESP32 product related information, please refer to documentation section of Espressif site.
- Mirror of this documentation is available under: https://dl.espressif.com/doc/esp-mdf/latest/.

CHAPTER 8

Copyrights and Licenses

8.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2018 Espressif Systems. This source code is licensed under the ESPRESSIF MIT License as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses:

Please refer to the COPYRIGHT in ESP-MDF Programming Guide

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

CHAPTER 9

About

This is documentation of ESP-MDF, the framework to develop mesh applications for ESP32 chip by Espressif.

The **ESP32** is 2.4 GHz Wi-Fi and Bluetooth combo, 32 bit dual core chip running up to 240 MHz, designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It has several peripherals on board including I2S interfaces to easy integrate with dedicated mesh chips. These hardware features together with the ESP-MDF software provide a powerful platform to implement mesh applications including native wireless networking and powerful user interface.

The **ESP-MDF** provides a range of API components including **Mesh Streams**, **Codecs** and **Services** organized in **Mesh Pipeline**, all integrated with mesh hardware through **Media HAL** and with **Peripherals** onboard of **ESP32**.

The ESP-MDF also provides integration with **Aliyun** cloud services.

The **ESP-MDF** builds on well established, FreeRTOS based, Espressif IOT Development Framework ESP-IDF.

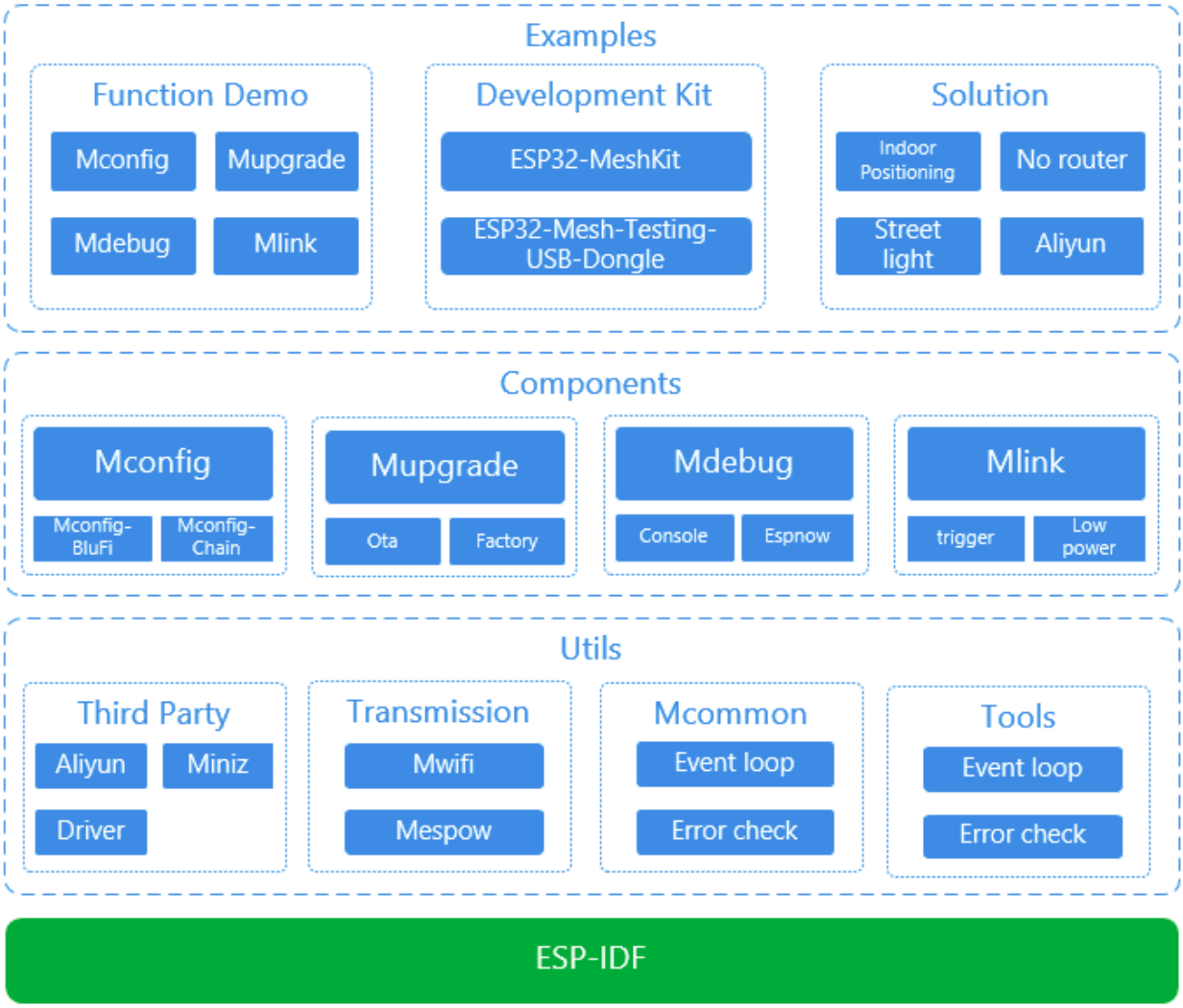


Fig. 1: Espressif Mesh Development Framework

CHAPTER 10

Switch Between Languages/

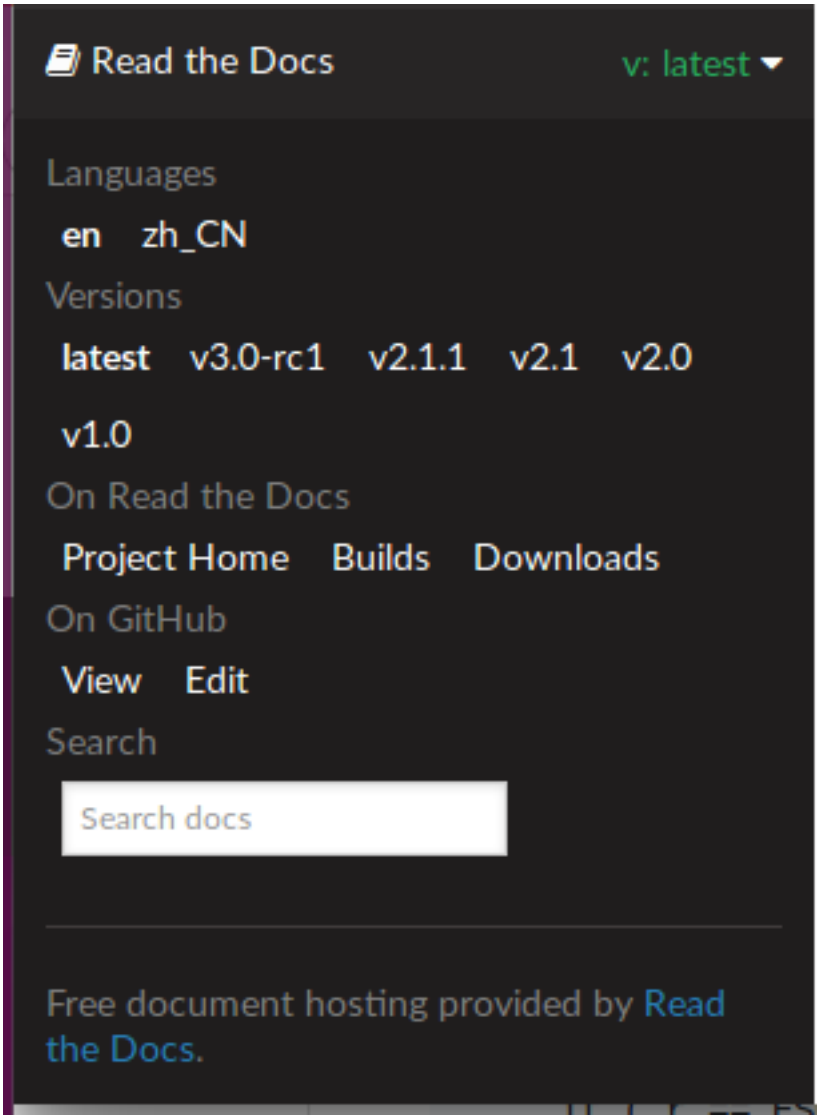
The ESP-MDF Programming Manual is now available in two languages. Please refer to the English version if there is any discrepancy.

ESP-MDF

- English/
- Chinese/

You can easily change from one language to another by the panel on the sidebar like below. Just click on the **Read the Docs** title button on the left-bottom corner if it is folded.

Read the Docs


Read the Docs
v: latest
Languages
en
zh_CN
Versions
latest
v3.0-rc1
v2.1.1
v2.1
v2.0
v1.0
On Read the Docs
Project Home
Builds
Downloads
On GitHub
View
Edit
Search
Search docs
Free document hosting provided by Read the Docs.

- genindex

Index

Symbols

C

E

L

M

N

P