]

# Overview

ESP-NOW is a kind of connectionless Wi-Fi communication protocol that is defined by Espressif. In ESP-NOW, application data is encapsulated in a vendor-specific action frame and then transmitted from one Wi-Fi device to another without connection.

CTR with CBC-MAC Protocol (CCMP) is used to protect the action frame for security. ESP-NOW is widely used in smart light, remote controlling, sensor, etc.

# Frame Format

ESP-NOW uses a vendor-specific action frame to transmit ESP-NOW data. The default ESP-NOW bit rate is 1 Mbps.

Currently, ESP-NOW supports two versions: v1.0 and v2.0. The maximum packet length supported by v2.0 devices is 1470 (`ESP_NOW_MAX_DATA_LEN_V2`) bytes, while the maximum packet length supported by v1.0 devices is 250 (`ESP_NOW_MAX_DATA_LEN`) bytes.

The v2.0 devices are capable of receiving packets from both v2.0 and v1.0 devices. In contrast, v1.0 devices can only receive packets from other v1.0 devices.

However, v1.0 devices can receive v2.0 packets if the packet length is less than or equal to 250 (`ESP_NOW_MAX_IE_DATA_LEN`).For packets exceeding this length, the v1.0 devices will either truncate the data to the first 250 (`ESP_NOW_MAX_IE_DATA_LEN`) bytes or discard the packet entirely.

For detailed behavior, please refer to the documentation corresponding to the specific IDF version.

The format of the vendor-specific action frame is as follows:

```
-------------------------------------------------------------------------------------------------
| MAC Header | Category Code | Organization Identifier | Random Values | Vendor Specific Content |   FCS   |
-------------------------------------------------------------------------------------------------
```

```
  24 bytes         1 byte              3 bytes               4 bytes             7-x bytes
4 bytes
```

- Category Code: The Category Code field is set to the value (127) indicating the vendor-specific category.
- Organization Identifier: The Organization Identifier contains a unique identifier (0x18fe34), which is the first three bytes of MAC address applied by Espressif.
- Random Value: The Random Value filed is used to prevents relay attacks.
- Vendor Specific Content: The Vendor Specific Content contains several (at least one) vendor-specific element fields. For version v2.0, x = 1512(1470 + 6*7), for version v1.0, x = 257(250 + 7).

The format of the vendor-specific element frame is as follows:

```
ESP-NOW v1.0:
-------------------------------------------------------------------------------------
| Element ID | Length | Organization Identifier | Type | Reserved | Version |    Body    |
-------------------------------------------------------------------------------------
                                                        7~4 bits | 3~0 bits
    1 byte     1 byte            3 bytes         1 byte        1 byte         0-250 bytes

ESP-NOW v2.0:
-------------------------------------------------------------------------------------
----------
| Element ID | Length | Organization Identifier | Type | Reserved | More data |Version |
Body    |
-------------------------------------------------------------------------------------
----------
                                                        7~5 bits |   1 bit   |3~0 bits
    1 byte     1 byte            3 bytes         1 byte             1 byte
0-250 bytes
```

- Element ID: The Element ID field is set to the value (221), indicating the vendor-specific element.
- Length: The length is the total length of Organization Identifier, Type, Version and Body, the maximum value is 255.
- Organization Identifier: The Organization Identifier contains a unique identifier (0x18fe34), which is the first three bytes of MAC address applied by Espressif.
- Type: The Type field is set to the value (4) indicating ESP-NOW.
- Version: The Version field is set to the version of ESP-NOW.
- Body: The Body contains the actual ESP-NOW data to be transmitted.

As ESP-NOW is connectionless, the MAC header is a little different from that of standard frames. The FromDS and ToDS bits of FrameControl field are both 0. The first address field is set to the destination address. The second address field is set to the source address. The third address field is set to broadcast address (0xff:0xff:0xff:0xff:0xff:0xff).

# Security

ESP-NOW uses the CCMP method, which is described in IEEE Std. 802.11-2012, to protect the vendor-specific action frame. The Wi-Fi device maintains a Primary Master Key (PMK) and several Local Master Keys (LMKs, each paired device has one LMK). The lengths of both PMK and LMK are 16 bytes.

- PMK is used to encrypt LMK with the AES-128 algorithm.
  Call `esp_now_set_pmk()` to set PMK. If PMK is not set, a default PMK will be used.
- LMK of the paired device is used to encrypt the vendor-specific action frame with the CCMP method. If the LMK of the paired device is not set, the vendor-specific action frame will not be encrypted.

Encrypting multicast vendor-specific action frame is not supported.

## Initialization and Deinitialization

Call `esp_now_init()` to initialize ESP-NOW and `esp_now_deinit()` to de-initialize ESP-NOW. ESP-NOW data must be transmitted after Wi-Fi is started, so it is recommended to start Wi-Fi before initializing ESP-NOW and stop Wi-Fi after de-initializing ESP-NOW.

When `esp_now_deinit()` is called, all of the information of paired devices are deleted.

## Add Paired Device

Call `esp_now_add_peer()` to add the device to the paired device list before you send data to this device. If security is enabled, the LMK must be set. A device with a broadcast MAC address must be added before sending broadcast data.

You can send ESP-NOW data via both the Station and the SoftAP interface. Make sure that the interface is enabled before sending ESP-NOW data.

The range of the channel of paired devices is from 0 to 14. If the channel is set to 0, data will be sent on the current channel. Otherwise, the channel must be set as the channel that the local device is on.

For the receiving device, calling `esp_now_add_peer()` is not required. If no paired device is added, it can only receive broadcast packets and unencrypted unicast

packets. To receive encrypted unicast packets, a paired device must be added, and the same LMK must be set.

The maximum number of paired devices is 20, and the paired encryption devices are no more than 17, the default is 7. If you want to change the number of paired encryption devices, set CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM in the Wi-Fi component configuration menu.

## Send ESP-NOW Data

Call `esp_now_send()` to send ESP-NOW data and `esp_now_register_send_cb()` to register sending callback function. It will return *ESP_NOW_SEND_SUCCESS* in sending callback function if the data is received successfully on the MAC layer. Otherwise, it will return *ESP_NOW_SEND_FAIL*. Several reasons can lead to ESP-NOW fails to send data. For example, the destination device does not exist; the channels of the devices are not the same; the action frame is lost when transmitting on the air, etc. It is not guaranteed that application layer can receive the data. If necessary, send back ack data when receiving ESP-NOW data. If receiving ack data timeouts, retransmit the ESP-NOW data. A sequence number can also be assigned to ESP-NOW data to drop the duplicate data.

If there is a lot of ESP-NOW data to send, call `esp_now_send()` to send less than or equal to the maximum packet length (v1.0 is 250 bytes, v2.0 is 1470 bytes) of data once a time. Note that too short interval between sending two ESP-NOW data may lead to disorder of sending callback function. So, it is recommended that sending the next ESP-NOW data after the sending callback function of the previous sending has returned. The sending callback function runs from a high-priority Wi-Fi task. So, do not do lengthy operations in the callback function. Instead, post the necessary data to a queue and handle it from a lower priority task.

## Receiving ESP-NOW Data

Call `esp_now_register_recv_cb()` to register receiving callback function. Call the receiving callback function when receiving ESP-NOW. The receiving callback function also runs from the Wi-Fi task. So, do not do lengthy operations in the callback function. Instead, post the necessary data to a queue and handle it from a lower priority task.

# Config ESP-NOW Rate

Call `esp_now_set_peer_rate_config()` to configure ESP-NOW rate of each peer. Make sure that the peer is added before configuring the rate. This API should be called after `esp_wifi_start()` and `esp_now_add_peer()`.

# Config ESP-NOW Power-saving Parameter

Sleep is supported only when ESP32 is configured as station.

Call `esp_now_set_wake_window()` to configure Window for ESP-NOW RX at sleep. The default value is the maximum, which allowing RX all the time.

If Power-saving is needed for ESP-NOW,
call `esp_wifi_connectionless_module_set_wake_interval()` to configure Interval as well.

Please refer to connectionless module power save to get more detail.

# Application Examples

- wifi/espnow demonstrates how to use the ESPNOW feature of ESP32's Wi-Fi, including starting Wi-Fi, initializing ESP-NOW, registering ESP-NOW sending or receiving callback function, adding ESP-NOW peer information, and sending and receiving ESP-NOW data between two devices.

# API Reference

## Header File

- components/esp_wifi/include/esp_now.h
- This header file can be included with:

```
#include "esp_now.h"
```

- This header file is a part of the API provided by the `esp_wifi` component. To declare that your component depends on `esp_wifi`, add the following to your CMakeLists.txt:

```
REQUIRES esp_wifi
```

or

```
PRIV_REQUIRES esp_wifi
```

## Functions

### esp_err_t esp_now_init(void)

Initialize ESPNOW function.

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_INTERNAL : Internal error

### esp_err_t esp_now_deinit(void)

De-initialize ESPNOW function.

**Returns:**

- ESP_OK : succeed

### esp_err_t esp_now_get_version(uint32_t *version)

Get the version of ESPNOW. Currently, ESPNOW supports two versions: v1.0 and v2.0.

```
       The v2.0 devices are capable of receiving packets from both v2.0 and v1.0
devices. In contrast, v1.0 devices can only receive packets from other v1.0
devices.
       However, v1.0 devices can receive v2.0 packets if the packet length is less
than or equal to ESP_NOW_MAX_IE_DATA_LEN.
       For packets exceeding this length, the v1.0 devices will either truncate the
data to the first ESP_NOW_MAX_IE_DATA_LEN bytes or discard the packet entirely.
       For detailed behavior, please refer to the documentation corresponding to
the specific IDF version.
```

**Parameters:**

**version** -- ESPNOW version

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_ARG : invalid argument

### esp_err_t esp_now_register_recv_cb(esp_now_recv_cb_t cb)

Register callback function of receiving ESPNOW data.

**Parameters:**

**cb** -- callback function of receiving ESPNOW data

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL : internal error

### esp_err_t esp_now_unregister_recv_cb(void)

Unregister callback function of receiving ESPNOW data.

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

---

**esp_err_t esp_now_register_send_cb(esp_now_send_cb_t cb)**

Register callback function of sending ESPNOW data.

**Parameters:**

**cb** -- callback function of sending ESPNOW data

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL : internal error

---

**esp_err_t esp_now_unregister_send_cb(void)**

Unregister callback function of sending ESPNOW data.

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

---

**esp_err_t esp_now_send(*const* uint8_t *peer_addr, *const* uint8_t *data, size_t len)**

Send ESPNOW data.

**Attention**

1. If peer_addr is not NULL, send data to the peer whose MAC address matches peer_addr

**Attention**

2. If peer_addr is NULL, send data to all of the peers that are added to the peer list

**Attention**

3. The maximum length of data must be less than ESP_NOW_MAX_DATA_LEN

**Attention**

4. The buffer pointed to by data argument does not need to be valid after esp_now_send returns

**Parameters:**

- **peer_addr** -- peer MAC address
- **data** -- data to send
- **len** -- length of data

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_INTERNAL : internal error
- ESP_ERR_ESPNOW_NO_MEM : out of memory, when this happens, you can delay a while before sending the next data
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found
- ESP_ERR_ESPNOW_IF : current Wi-Fi interface doesn't match that of peer
- ESP_ERR_ESPNOW_CHAN: current Wi-Fi channel doesn't match that of peer

---

**esp_err_t esp_now_add_peer(*const* esp_now_peer_info_t *peer)**

Add a peer to peer list.

**Parameters:**

**peer** -- peer information

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_FULL : peer list is full
- ESP_ERR_ESPNOW_NO_MEM : out of memory
- ESP_ERR_ESPNOW_EXIST : peer has existed

---

**esp_err_t esp_now_del_peer(*const* uint8_t *peer_addr)**

Delete a peer from peer list.

**Parameters:**

**peer_addr** -- peer MAC address

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

---

**esp_err_t esp_now_mod_peer(*const* esp_now_peer_info_t *peer)**

Modify a peer.

**Parameters:**

**peer** -- peer information

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_FULL : peer list is full

### esp_err_t esp_now_set_peer_rate_config(*const* uint8_t *peer_addr, esp_now_rate_config_t *config)

Set ESPNOW rate config for each peer.

**Attention**

1. This API should be called after esp_wifi_start() and esp_now_init().

**Parameters:**

- **peer_addr** -- peer MAC address
- **config** -- rate config to be configured.

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_INTERNAL : internal error

### esp_err_t esp_now_get_peer(*const* uint8_t *peer_addr, esp_now_peer_info_t *peer)

Get a peer whose MAC address matches peer_addr from peer list.

**Parameters:**

- **peer_addr** -- peer MAC address
- **peer** -- peer information

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

### esp_err_t esp_now_fetch_peer(bool from_head, esp_now_peer_info_t *peer)

Fetch a peer from peer list. Only return the peer which address is unicast, for the multicast/broadcast address, the function will ignore and try to find the next in the peer list.

**Parameters:**

- **from_head** -- fetch from head of list or not
- **peer** -- peer information

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

**bool esp_now_is_peer_exist(*const* uint8_t *peer_addr)**

Peer exists or not.

**Parameters:**

**peer_addr** -- peer MAC address

**Returns:**

- true : peer exists
- false : peer not exists

**esp_err_t esp_now_get_peer_num(esp_now_peer_num_t *num)**

Get the number of peers.

**Parameters:**

**num** -- number of peers

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

**esp_err_t esp_now_set_pmk(*const* uint8_t *pmk)**

Set the primary master key.

**Attention**

1. primary master key is used to encrypt local master key

**Parameters:**

**pmk** -- primary master key

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

**esp_err_t esp_now_set_wake_window(uint16_t window)**

Set wake window for esp_now to wake up in interval unit.

**Attention**

1. This configuration could work at connected status. When ESP_WIFI_STA_DISCONNECTED_PM_ENABLE is enabled, this configuration could work at disconnected status.

**Attention**

2. Default value is the maximum.

**Parameters:**

**window** -- Milliseconds would the chip keep waked each interval, from 0 to 65535.

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

---

**esp_err_t esp_now_set_user_oui(uint8_t *oui)**

Set the OUI (Organization Identifier) in the vendor-specific element for ESPNOW.

**Parameters:**

**oui** -- The oui should occupy 3 bytes. If the oui is NULL, then use the default value (0x18fe34).

**Returns:**

- ESP_OK : succeed

---

**esp_err_t esp_now_get_user_oui(uint8_t *oui)**

Get the OUI (Organization Identifier) in the vendor-specific element for ESPNOW.

**Parameters:**

**oui** -- user configured OUI.

**Returns:**

- ESP_OK : succeed
- ESP_ERR_ESPNOW_ARG : invalid argument

---

**esp_err_t esp_now_switch_channel_tx(esp_now_switch_channel_t *config)**

ESPNOW switch to a specific channel for a required duration, and send one ESPNOW data.

**Parameters:**

**config** -- ESPNOW switch channel relevant information

**Returns:**

- ESP_OK : succeed

- ESP_ERR_NO_MEM: failed to allocate memory
- ESP_ERR_INVALID_ARG: the <channel, sec_channel> pair is invalid
- ESP_FAIL: failed to send frame

**esp_err_t esp_now_remain_on_channel(esp_now_remain_on_channel_t *config)**

ESPNOW remain on the target channel for required duration.

**Parameters:**

**config** -- ESPNOW remain on channel relevant information

**Returns:**

- ESP_OK : succeed
- ESP_ERR_NO_MEM: failed to allocate memory
- ESP_ERR_INVALID_ARG: the <channel, sec_channel> pair is invalid
- ESP_FAIL: failed to perform roc operation

## Structures

***struct* esp_now_peer_info**

ESPNOW peer information parameters.

### Public Members

**uint8_t peer_addr[ESP_NOW_ETH_ALEN]**

ESPNOW peer MAC address that is also the MAC address of station or softap

**uint8_t lmk[ESP_NOW_KEY_LEN]**

ESPNOW peer local master key that is used to encrypt data

**uint8_t channel**

Wi-Fi channel that peer uses to send/receive ESPNOW data. If the value is 0, use the current channel which station or softap is on. Otherwise, it must be set as the channel that station or softap is on.

**wifi_interface_t ifidx**

Wi-Fi interface that peer uses to send/receive ESPNOW data

**bool encrypt**

ESPNOW data that this peer sends/receives is encrypted or not

**void *priv**

ESPNOW peer private data

***struct* esp_now_peer_num**

Number of ESPNOW peers which exist currently.

### Public Members

**int total_num**

Total number of ESPNOW peers, maximum value is ESP_NOW_MAX_TOTAL_PEER_NUM

**int encrypt_num**

Number of encrypted ESPNOW peers, maximum value is ESP_NOW_MAX_ENCRYPT_PEER_NUM

## *struct* esp_now_recv_info

ESPNOW receive packet information.

### Public Members

**uint8_t *src_addr**

Source address of ESPNOW packet

**uint8_t *des_addr**

Destination address of ESPNOW packet

**wifi_pkt_rx_ctrl_t *rx_ctrl**

Rx control info of ESPNOW packet

## *struct* esp_now_switch_channel_t

ESPNOW switch channel information.

### Public Members

**wifi_action_tx_t type**

ACTION TX operation type

**uint8_t channel**

Channel on which to perform ESPNOW TX Operation

**wifi_second_chan_t sec_channel**

Secondary channel

**uint32_t wait_time_ms**

Duration to wait for on target channel

**uint8_t op_id**

Unique Identifier for operation provided by wifi driver

**uint8_t dest_mac[6]**

Destination MAC address

**uint16_t data_len**

Length of the appended Data

**uint8_t data[0]**

Appended Data payload

## *struct* esp_now_remain_on_channel_t

ESPNOW remain on channel information.

### Public Members

**wifi_roc_t type**

ROC operation type

**uint8_t channel**

Channel on which to perform ESPNOW ROC Operation

**wifi_second_chan_t sec_channel**

Secondary channel

**uint32_t wait_time_ms**

Duration to wait for on target channel

**uint8_t op_id**

ID of this specific ROC operation provided by wifi driver

## Macros

**ESP_ERR_ESPNOW_BASE**

ESPNOW error number base.

**ESP_ERR_ESPNOW_NOT_INIT**

ESPNOW is not initialized.

**ESP_ERR_ESPNOW_ARG**

Invalid argument

**ESP_ERR_ESPNOW_NO_MEM**

Out of memory

**ESP_ERR_ESPNOW_FULL**

ESPNOW peer list is full

**ESP_ERR_ESPNOW_NOT_FOUND**

ESPNOW peer is not found

**ESP_ERR_ESPNOW_INTERNAL**

Internal error

**ESP_ERR_ESPNOW_EXIST**

ESPNOW peer has existed

**ESP_ERR_ESPNOW_IF**

Interface error

**ESP_ERR_ESPNOW_CHAN**

Channel error

**ESP_NOW_ETH_ALEN**

Length of ESPNOW peer MAC address

**ESP_NOW_KEY_LEN**

Length of ESPNOW peer local master key

**ESP_NOW_MAX_TOTAL_PEER_NUM**

Maximum number of ESPNOW total peers

**ESP_NOW_MAX_ENCRYPT_PEER_NUM**

Maximum number of ESPNOW encrypted peers

**ESP_NOW_MAX_IE_DATA_LEN**

Maximum data length in a vendor-specific element

**ESP_NOW_MAX_DATA_LEN**

Maximum length of data sent in each ESPNOW transmission for v1.0

**ESP_NOW_MAX_DATA_LEN_V2**

Maximum length of data sent in each ESPNOW transmission for v2.0

## Type Definitions

***typedef struct* esp_now_peer_info esp_now_peer_info_t**

ESPNOW peer information parameters.

***typedef struct* esp_now_peer_num esp_now_peer_num_t**

Number of ESPNOW peers which exist currently.

***typedef struct* esp_now_recv_info esp_now_recv_info_t**

ESPNOW receive packet information.

***typedef* wifi_tx_info_t esp_now_send_info_t**

ESPNOW sending packet information.

***typedef* wifi_tx_rate_config_t esp_now_rate_config_t**

ESPNOW rate config.

***typedef* void (*esp_now_recv_cb_t)(*const* esp_now_recv_info_t *esp_now_info, *const* uint8_t *data, int data_len)**

Callback function of receiving ESPNOW data.

**Attention**

esp_now_info is a local variable, it can only be used in the callback.

**Param esp_now_info:**

received ESPNOW packet information

**Param data:**

received data

**Param data_len:**

length of received data

***typedef* void (*esp_now_send_cb_t)(*const* esp_now_send_info_t *tx_info, esp_now_send_status_t status)**

Callback function of sending ESPNOW data.

**Param tx_info:**

Sending information for ESPNOW data

**Param status:**

status of sending ESPNOW data (succeed or fail). This is will be removed later, since the tx_info->tx_status also works.

## Enumerations

***enum* esp_now_send_status_t**

Status of sending ESPNOW data .

*Values:*

***enumerator* ESP_NOW_SEND_SUCCESS**

Send ESPNOW data successfully

***enumerator* ESP_NOW_SEND_FAIL**

Send ESPNOW data fail