ESP-NETIF

[中文]

The purpose of the ESP-NETIF library is twofold:

- It provides an abstraction layer for the application on top of the TCP/IP stack. This allows applications to choose between IP stacks in the future.
- The APIs it provides are thread-safe, even if the underlying TCP/IP stack APIs are not.

ESP-IDF currently implements ESP-NETIF for the lwIP TCP/IP stack only. However, the adapter itself is TCP/IP implementation-agnostic and allows different implementations.

Some ESP-NETIF API functions are intended to be called by application code, for example, to get or set interface IP addresses, and configure DHCP. Other functions are intended for internal ESP-IDF use by the network driver layer. In many cases, applications do not need to call ESP-NETIF APIs directly as they are called by the default network event handlers.

If you are only interested in using the most common network interfaces with default setting, please read ESP-NETIF User's Manual to see how you can initialize default interfaces and register event handlers.

If you would like to learn more about the library interaction with other components, please refer to the ESP-NETIF Architecture.

In case your application needs to configure the network interfaces differently, e.g. setting a static IP address or just update the configuration runtime, please read ESP-NETIF Programmer's Manual.

If you would like to develop your own network driver, implement support for a new TCP/IP stack or customize the ESP-NETIF in some other way, please refer to the ESP-NETIF Developer's Manual.

ESP-NETIF User's Manual

It is usually just enough to create a default network interface after startup and destroy it upon closing (see Initialization). It is also useful to receive a notification upon assigning a new IP address or losing it (see IP Events).

Initialization

Since the ESP-NETIF component uses system events, the typical network startup code looks like this (note that error handling is omitted for clarity, see ethernet/basic/main/ethernet_example_main.c for complete startup code):

```
// 1) Initialize the TCP/IP stack and the event loop
esp_netif_init();
esp_event_loop_create_default();

// 2) Create the network interface handle
esp_netif = esp_netif_new(&config);

// 3) Create the network interface driver (e.g., Ethernet) and it's network layer glue
// and register the ESP-NETIF event (e.g., to bring the interface up upon link-up event)
esp_netif_glue_t glue = driver_glue(driver);

// 4) Attach the driver's glue layer to the network interface handle
esp_netif_attach(esp_netif, glue);

// 5) Register user-side event handlers
esp_event_handler_register(DRIVER_EVENT, ...);  // to observe driver states, e.g., link-up
esp_event_handler_register(IP_EVENT, ...);      // to observe ESP-NETIF states, e.g., get an
IP
```

Note

These steps must be performed in the exact order shown above, as the network interface drivers use the default event loop when registering system events.

- The default event loop needs to be created **before** initializing an interface driver, as the driver typically needs to register system event handlers.
- Registering application event handlers must occur **after** calling `esp_netif_attach()`, because event handlers are called in the order they were registered. To ensure that system handlers are called first, you should register application handlers afterward.

Steps `2)`, `3)` and `4)` are quite complex for most common use-cases, so ESP-NETIF provides some pre-configured interfaces and convenience functions that create the most common network interfaces in their most common configurations.

Note

Each network interface needs to be initialized separately, so if you would like to use multiple interfaces, you would have to run steps `2)` to `5)` for every interface. Set `1)` should be performed only once.

Creating and configuring the interface and attaching the network interface driver to it (steps `2)`, `3)` and `4)`) is described in Common Network Interfaces.

Using the ESP-NETIF event handlers (step `5)`) is described in IP Events.

Common Network Interfaces

As the initialization of network interfaces could be quite complex, ESP-NETIF provides some convenient methods of creating the most common ones, such as Wi-Fi and Ethernet.

Please refer to the following examples to understand the initialization process of the default interface:

- wifi/getting_started/station demonstrates how to use the station functionality to connect ESP32 to an AP.
- wifi/getting_started/softAP demonstrates how to use the SoftAP functionality to configure ESP32 as an AP.
- ethernet/basic demonstrates how to use the Ethernet driver, attach it to *esp_netif*, and obtain an IP address that can be pinged.
- protocols/l2tap demonstrates how to use the ESP-NETIF L2 TAP interface to access the Data Link Layer for receiving and transmitting frames, implement non-IP protocols, and echo Ethernet frames with specific EthTypes.
- protocols/static_ip demonstrates how to configure Wi-Fi as a station, including setting up a static IP, netmask, gateway and DNS server.

Wi-Fi Default Initialization

The initialization code as well as registering event handlers for default interfaces, such as softAP and station, are provided in separate APIs to facilitate simple startup code for most applications:

- `esp_netif_create_default_wifi_sta()`
- `esp_netif_create_default_wifi_ap()`

Please note that these functions return the `esp_netif` handle, i.e., a pointer to a network interface object allocated and configured with default settings, which means that:

- The created object has to be destroyed if a network de-initialization is provided by an application using `esp_netif_destroy_default_wifi()`.
- These *default* interfaces must not be created multiple times unless the created handle is deleted using `esp_netif_destroy_default_wifi()`.
- When using Wi-Fi in `AP+STA` mode, both these interfaces have to be created. Please refer to the example wifi/softap_sta/main/softap_sta.c.

IP Events

In the final section of Initialization code (step `5)`), you register two sets of event handlers:

- **Network Interface Driver Events**: These events notify you about the driver's lifecycle states, such as when a Wi-Fi station joins an AP or gets disconnected. Handling these events is outside the scope of the ESP-NETIF component. It is worth noting that the same events are also used by ESP-NETIF to set the network interface to a desired state. Therefore, if your application uses the driver's events to determine specific states of the network interface, you should register these handlers **after** registering the system handlers (which typically happens when attaching the driver to the interface). This is why handler registration occurs in the final step of the Initialization code.
- **IP Events**: These events notify you about IP address changes, such as when a new address is assigned or when a valid address is lost. Specific types of these events are listed in `ip_event_t`. Each common interface has a related pair of `GOT_IP` and `LOST_IP` events.

Registering event handlers is crucial due to the asynchronous nature of networking, where changes in network state can occur unpredictably. By registering event handlers, applications can respond to these changes promptly, ensuring appropriate actions are taken in response to network events.

Note

Lost IP events are triggered by a timer configurable by CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL. The timer is started upon losing the IP address and the event will be raised after the configured interval, which is 120 s by default. The event could be disabled when setting the interval to 0.

ESP-NETIF Architecture

```
                       |          (A) USER CODE                 |
                       |                 Apps                   |
      .................| init          settings      events     |
      .                +----------------------------------------+
      .                   .                |           *
      .                   .                |           *
  --------+            +===========================+   *     +-----------------------+
          |            | new/config   get/set/apps |   *     | init                  |
          |            |                           |...*.....| Apps (DHCP, SNTP)     |
          |            |---------------------------|   *     |                       |
    init  |            |                           |****     |                       |
    start |************|  event handler            |*********|  DHCP                 |
    stop  |            |                           |         |                       |
          |            |---------------------------|         |                       |
          |            |                           |         |    NETIF              |
    +-----|            |                           |         +-----------------+     |
    | glue|---<----|---|  esp_netif_transmit       |--<------| netif_output    |     |
    |     |        |   |                           |         |                 |     |
    |     |--->----|---|  esp_netif_receive        |-->------| netif_input     |     |
    |     |        |   |                           |         + ----------------+     |
    |     |...<....|...|  esp_netif_free_rx_buffer |...<.....| packet buffer         |
    +-----|     |  |   |                           |         |                       |
          |     |  |   |                           |         |         (D)           |
    (B)   |     |  |   |          (C)              |         +-----------------------+
  --------+     |  |   +===========================+               NETWORK STACK
NETWORK         |  |           ESP-NETIF
INTERFACE       |  |
DRIVER          |  |   +---------------------------+         +------------------+
                |  |   |                           |.........| open/close       |
                |  |   |                           |         |                  |
                |  -<--|  l2tap_write              |-----<---|  write           |
                |      |                           |         |                  |
                ---->--|  esp_vfs_l2tap_eth_filter |----->---|  read            |
                       |                           |         |        (A)       |
                       |            (E)            |         +------------------+
                       +---------------------------+             USER CODE
                             ESP-NETIF L2 TAP
```

Data and Event Flow in the Diagram

- `........` Initialization line from user code to ESP-NETIF and network interface driver
- `--<--->--` Data packets going from communication media to TCP/IP stack and back
- `********` Events aggregated in ESP-NETIF propagate to the driver, user code, and network stack
- `|` User settings and runtime configuration

ESP-NETIF Interaction

A) User Code, Boilerplate

Overall application interaction with a specific IO driver for the communication media (network interface driver) and configured TCP/IP network stack is abstracted using ESP-NETIF APIs and is outlined as below:

A. Initialization code

    1. Initializes IO driver
    2. Creates a new instance of ESP-NETIF and configure it with
        - ESP-NETIF specific options (flags, behavior, name)
        - Network stack options (netif init and input functions, not publicly available)
        - IO driver specific options (transmit, free rx buffer functions, IO driver handle)
    3. Attaches the IO driver handle to the ESP-NETIF instance created in the above steps
    4. Configures event handlers
        - Use default handlers for common interfaces defined in IO drivers; or define a specific handler for customized behavior or new interfaces
        - Register handlers for app-related events (such as IP lost or acquired)

B. Interaction with network interfaces using ESP-NETIF API

    1. Gets and sets TCP/IP-related parameters (DHCP, IP, etc)
    2. Receives IP events (connect or disconnect)
    3. Controls application lifecycle (set interface up or down)

B) Network Interface Driver

Network interface driver (also called I/O Driver, or Media Driver) plays these two important roles in relation to ESP-NETIF:

1. Event handlers: Defines behavior patterns of interaction with ESP-NETIF (e.g., ethernet link-up -> turn netif on)
2. Glue IO layer: Adapts the input or output functions to use ESP-NETIF transmit, receive, and free receive buffer
    - Installs driver_transmit to the appropriate ESP-NETIF object so that outgoing packets from the network stack are passed to the IO driver
    - Calls `esp_netif_receive()` to pass incoming data to the network stack

C) ESP-NETIF

ESP-NETIF serves as an intermediary between an IO driver and a network stack, connecting the packet data path between the two. It provides a set of interfaces for attaching a driver to an ESP-NETIF object at runtime and configures a network stack during compiling.

Additionally, a set of APIs is provided to control the network interface lifecycle and its TCP/IP properties. As an overview, the ESP-NETIF public interface can be divided into six groups:

1. Initialization APIs (to create and configure ESP-NETIF instance)
2. Input or Output API (for passing data between IO driver and network stack)
3. Event or Action API
    - Used for network interface lifecycle management
    - ESP-NETIF provides building blocks for designing event handlers
4. Setters and Getters API for basic network interface properties
5. Network stack abstraction API: enabling user interaction with TCP/IP stack
    - Set interface up or down
    - DHCP server and client API
    - DNS API
    - SNTP Service
6. Driver conversion utilities API

D) Network Stack

The network stack has no public interaction with application code with regard to public interfaces and shall be fully abstracted by ESP-NETIF API.

E) ESP-NETIF L2 TAP Interface

The ESP-NETIF L2 TAP interface is a mechanism in ESP-IDF used to access Data Link Layer (L2 per OSI/ISO) for frame reception and transmission from the user application. Its typical usage in the embedded world might be the implementation of non-IP-related protocols, e.g., PTP, Wake on LAN. Note that only Ethernet (IEEE 802.3) is currently supported. Please read more about L2 TAP in L2 TAP Interface Usage.

ESP-NETIF Programmer's Manual

In some cases, it is not enough to simply initialize a network interface by default, start using it and connect to the local network. If so, please consult the programming guide: ESP-NETIF Programmers Manual.

You would typically need to use specific sets of ESP-NETIF APIs in the following use-cases:

- Configure IP, Gateway, and DNS
- Configure DHCP options
- SNTP Service
- L2 TAP Interface Usage
- IP Event: Transmit/Receive Packet
- API Reference

ESP-NETIF Developer's Manual

In some cases, user applications might need to customize ESP-NETIF, register custom drivers or even use a custom TCP/IP stack. If so, please consult the ESP-NETIF Developer's manual.