|  |
| --- |
|  |
|  | |
|  | | **This code is taken form github.**  **link:** [**https://github.com/espressif/esp-idf/blob/cf056a7/components/esp\_wifi/include/esp\_mesh.h**](https://github.com/espressif/esp-idf/blob/cf056a7/components/esp_wifi/include/esp_mesh.h)  **explanation:** [**https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp\_mesh.html#\_CPPv413esp\_mesh\_sendPK11mesh\_addr\_tPK11mesh\_data\_tiA\_K10mesh\_opt\_ti**](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_mesh.html#_CPPv413esp_mesh_sendPK11mesh_addr_tPK11mesh_data_tiA_K10mesh_opt_ti) |
|  | // Licensed under the Apache License, Version 2.0 (the "License"); | |
|  | // you may not use this file except in compliance with the License. | |
|  | // You may obtain a copy of the License at | |
|  |  | |
|  | // http://www.apache.org/licenses/LICENSE-2.0 | |
|  | // | |
|  | // Unless required by applicable law or agreed to in writing, software | |
|  | // distributed under the License is distributed on an "AS IS" BASIS, | |
|  | // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. | |
|  | // See the License for the specific language governing permissions and | |
|  | // limitations under the License. | |
|  |  | |
|  |  | |
|  |  | |
|  | \* Mesh Stack | |
|  | \* | |
|  | \* Mesh event defines almost all system events applications tasks need. | |
|  | \* Mesh event contains Wi-Fi connection states on station interface, children connection states on softAP interface and etc.. | |
|  | \* Applications need to register a mesh event callback handler by API esp\_mesh\_set\_config() firstly. | |
|  | \* This handler is to receive events posted from mesh stack and LwIP stack. | |
|  | \* Applications could add relative handler for each event. | |
|  | \* Examples: | |
|  | \* (1) Applications could use Wi-Fi station connect states to decide when to send data to its parent, to the root or to external IP network; | |
|  | \* (2) Applications could use Wi-Fi softAP states to decide when to send data to its children. | |
|  | \* | |
|  | \* In present implementation, applications are able to access mesh stack directly without having to go through LwIP stack. | |
|  | \* Applications use esp\_mesh\_send() and esp\_mesh\_recv() to send and receive messages over the mesh network. | |
|  | \* In mesh stack design, normal devices don't require LwIP stack. But since IDF hasn't supported system without initializing LwIP stack yet, | |
|  | \* applications still need to do LwIP initialization and two more things are required to be done | |
|  | \* (1) stop DHCP server on softAP interface by default | |
|  | \* (2) stop DHCP client on station interface by default. | |
|  | \* Examples: | |
|  | \* tcpip\_adapter\_init(); | |
|  | \* tcpip\_adapter\_dhcps\_stop(TCPIP\_ADAPTER\_IF\_AP)； | |
|  | \* tcpip\_adapter\_dhcpc\_stop(TCPIP\_ADAPTER\_IF\_STA)； | |
|  | \* | |
|  | \* Over the mesh network, only the root is able to access external IP network. | |
|  | \* In application mesh event handler, once a device becomes a root, start DHCP client immediately whether DHCP is chosen. | |
|  | \*/ | |
|  |  | |
|  | #ifndef \_\_ESP\_MESH\_H\_\_ | |
|  | #define \_\_ESP\_MESH\_H\_\_ | |
|  |  | |
|  | #include "esp\_err.h" | |
|  | #include "esp\_wifi.h" | |
|  | #include "esp\_wifi\_types.h" | |
|  | #include "esp\_mesh\_internal.h" | |
|  | #include "lwip/ip\_addr.h" | |
|  |  | |
|  | #ifdef \_\_cplusplus | |
|  | extern "C" { | |
|  | #endif | |
|  |  | |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
|  | \* Constants | |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |
|  | #define MESH\_ROOT\_LAYER (1) /\*\*< root layer value \*/ | |
|  | #define MESH\_MTU (1500) /\*\*< max transmit unit(in bytes) \*/ | |
|  | #define MESH\_MPS (1472) /\*\*< max payload size(in bytes) \*/ | |
|  | /\*\* | |
|  | \* @brief Mesh error code definition | |
|  | \*/ | |
|  | #define ESP\_ERR\_MESH\_WIFI\_NOT\_START (ESP\_ERR\_MESH\_BASE + 1) /\*\*< Wi-Fi isn't started \*/ | |
|  | #define ESP\_ERR\_MESH\_NOT\_INIT (ESP\_ERR\_MESH\_BASE + 2) /\*\*< mesh isn't initialized \*/ | |
|  | #define ESP\_ERR\_MESH\_NOT\_CONFIG (ESP\_ERR\_MESH\_BASE + 3) /\*\*< mesh isn't configured \*/ | |
|  | #define ESP\_ERR\_MESH\_NOT\_START (ESP\_ERR\_MESH\_BASE + 4) /\*\*< mesh isn't started \*/ | |
|  | #define ESP\_ERR\_MESH\_NOT\_SUPPORT (ESP\_ERR\_MESH\_BASE + 5) /\*\*< not supported yet \*/ | |
|  | #define ESP\_ERR\_MESH\_NOT\_ALLOWED (ESP\_ERR\_MESH\_BASE + 6) /\*\*< operation is not allowed \*/ | |
|  | #define ESP\_ERR\_MESH\_NO\_MEMORY (ESP\_ERR\_MESH\_BASE + 7) /\*\*< out of memory \*/ | |
|  | #define ESP\_ERR\_MESH\_ARGUMENT (ESP\_ERR\_MESH\_BASE + 8) /\*\*< illegal argument \*/ | |
|  | #define ESP\_ERR\_MESH\_EXCEED\_MTU (ESP\_ERR\_MESH\_BASE + 9) /\*\*< packet size exceeds MTU \*/ | |
|  | #define ESP\_ERR\_MESH\_TIMEOUT (ESP\_ERR\_MESH\_BASE + 10) /\*\*< timeout \*/ | |
|  | #define ESP\_ERR\_MESH\_DISCONNECTED (ESP\_ERR\_MESH\_BASE + 11) /\*\*< disconnected with parent on station interface \*/ | |
|  | #define ESP\_ERR\_MESH\_QUEUE\_FAIL (ESP\_ERR\_MESH\_BASE + 12) /\*\*< queue fail \*/ | |
|  | #define ESP\_ERR\_MESH\_QUEUE\_FULL (ESP\_ERR\_MESH\_BASE + 13) /\*\*< queue full \*/ | |
|  | #define ESP\_ERR\_MESH\_NO\_PARENT\_FOUND (ESP\_ERR\_MESH\_BASE + 14) /\*\*< no parent found to join the mesh network \*/ | |
|  | #define ESP\_ERR\_MESH\_NO\_ROUTE\_FOUND (ESP\_ERR\_MESH\_BASE + 15) /\*\*< no route found to forward the packet \*/ | |
|  | #define ESP\_ERR\_MESH\_OPTION\_NULL (ESP\_ERR\_MESH\_BASE + 16) /\*\*< no option found \*/ | |
|  | #define ESP\_ERR\_MESH\_OPTION\_UNKNOWN (ESP\_ERR\_MESH\_BASE + 17) /\*\*< unknown option \*/ | |
|  | #define ESP\_ERR\_MESH\_XON\_NO\_WINDOW (ESP\_ERR\_MESH\_BASE + 18) /\*\*< no window for software flow control on upstream \*/ | |
|  | #define ESP\_ERR\_MESH\_INTERFACE (ESP\_ERR\_MESH\_BASE + 19) /\*\*< low-level Wi-Fi interface error \*/ | |
|  | #define ESP\_ERR\_MESH\_DISCARD\_DUPLICATE (ESP\_ERR\_MESH\_BASE + 20) /\*\*< discard the packet due to the duplicate sequence number \*/ | |
|  | #define ESP\_ERR\_MESH\_DISCARD (ESP\_ERR\_MESH\_BASE + 21) /\*\*< discard the packet \*/ | |
|  | #define ESP\_ERR\_MESH\_VOTING (ESP\_ERR\_MESH\_BASE + 22) /\*\*< vote in progress \*/ | |
|  | #define ESP\_ERR\_MESH\_XMIT (ESP\_ERR\_MESH\_BASE + 23) /\*\*< XMIT \*/ | |
|  | #define ESP\_ERR\_MESH\_QUEUE\_READ (ESP\_ERR\_MESH\_BASE + 24) /\*\*< error in reading queue \*/ | |
|  | #define ESP\_ERR\_MESH\_PS (ESP\_ERR\_MESH\_BASE + 25) /\*\*< mesh PS is not specified as enable or disable \*/ | |
|  | #define ESP\_ERR\_MESH\_RECV\_RELEASE (ESP\_ERR\_MESH\_BASE + 26) /\*\*< release esp\_mesh\_recv\_toDS \*/ | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Flags bitmap for esp\_mesh\_send() and esp\_mesh\_recv() | |
|  | \*/ | |
|  | #define MESH\_DATA\_ENC (0x01) /\*\*< data encrypted (Unimplemented) \*/ | |
|  | #define MESH\_DATA\_P2P (0x02) /\*\*< point-to-point delivery over the mesh network \*/ | |
|  | #define MESH\_DATA\_FROMDS (0x04) /\*\*< receive from external IP network \*/ | |
|  | #define MESH\_DATA\_TODS (0x08) /\*\*< identify this packet is target to external IP network \*/ | |
|  | #define MESH\_DATA\_NONBLOCK (0x10) /\*\*< esp\_mesh\_send() non-block \*/ | |
|  | #define MESH\_DATA\_DROP (0x20) /\*\*< in the situation of the root having been changed, identify this packet can be dropped by new root \*/ | |
|  | #define MESH\_DATA\_GROUP (0x40) /\*\*< identify this packet is target to a group address \*/ | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Option definitions for esp\_mesh\_send() and esp\_mesh\_recv() | |
|  | \*/ | |
|  | #define MESH\_OPT\_SEND\_GROUP (7) /\*\*< data transmission by group; used with esp\_mesh\_send() and shall have payload \*/ | |
|  | #define MESH\_OPT\_RECV\_DS\_ADDR (8) /\*\*< return a remote IP address; used with esp\_mesh\_send() and esp\_mesh\_recv() \*/ | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Flag of mesh networking IE | |
|  | \*/ | |
|  | #define MESH\_ASSOC\_FLAG\_VOTE\_IN\_PROGRESS (0x02) /\*\*< vote in progress \*/ | |
|  | #define MESH\_ASSOC\_FLAG\_NETWORK\_FREE (0x08) /\*\*< no root in current network \*/ | |
|  | #define MESH\_ASSOC\_FLAG\_ROOTS\_FOUND (0x20) /\*\*< root conflict is found \*/ | |
|  | #define MESH\_ASSOC\_FLAG\_ROOT\_FIXED (0x40) /\*\*< fixed root \*/ | |
|  |  | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh PS (Power Save) duty cycle type | |
|  | \*/ | |
|  | #define MESH\_PS\_DEVICE\_DUTY\_REQUEST (0x01) /\*\*< requests to join a network PS without specifying a device duty cycle. After the | |
|  | device joins the network, a network duty cycle will be provided by the network \*/ | |
|  | #define MESH\_PS\_DEVICE\_DUTY\_DEMAND (0x04) /\*\*< requests to join a network PS and specifies a demanded device duty cycle \*/ | |
|  | #define MESH\_PS\_NETWORK\_DUTY\_MASTER (0x80) /\*\*< indicates the device is the NWK-DUTY-MASTER (network duty cycle master) \*/ | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh PS (Power Save) duty cycle applied rule | |
|  | \*/ | |
|  | #define MESH\_PS\_NETWORK\_DUTY\_APPLIED\_ENTIRE (0) /\*\* the specified network duty is applied to the entire network <\*/ | |
|  | #define MESH\_PS\_NETWORK\_DUTY\_APPLIED\_UPLINK (1) /\*\* the specified network duty is applied to only the up-link path <\*/ | |
|  |  | |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
|  | \* Enumerations | |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |
|  | /\*\* | |
|  | \* @brief Enumerated list of mesh event id | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_EVENT\_STARTED, /\*\*< mesh is started \*/ | |
|  | MESH\_EVENT\_STOPPED, /\*\*< mesh is stopped \*/ | |
|  | MESH\_EVENT\_CHANNEL\_SWITCH, /\*\*< channel switch \*/ | |
|  | MESH\_EVENT\_CHILD\_CONNECTED, /\*\*< a child is connected on softAP interface \*/ | |
|  | MESH\_EVENT\_CHILD\_DISCONNECTED, /\*\*< a child is disconnected on softAP interface \*/ | |
|  | MESH\_EVENT\_ROUTING\_TABLE\_ADD, /\*\*< routing table is changed by adding newly joined children \*/ | |
|  | MESH\_EVENT\_ROUTING\_TABLE\_REMOVE, /\*\*< routing table is changed by removing leave children \*/ | |
|  | MESH\_EVENT\_PARENT\_CONNECTED, /\*\*< parent is connected on station interface \*/ | |
|  | MESH\_EVENT\_PARENT\_DISCONNECTED, /\*\*< parent is disconnected on station interface \*/ | |
|  | MESH\_EVENT\_NO\_PARENT\_FOUND, /\*\*< no parent found \*/ | |
|  | MESH\_EVENT\_LAYER\_CHANGE, /\*\*< layer changes over the mesh network \*/ | |
|  | MESH\_EVENT\_TODS\_STATE, /\*\*< state represents whether the root is able to access external IP network \*/ | |
|  | MESH\_EVENT\_VOTE\_STARTED, /\*\*< the process of voting a new root is started either by children or by the root \*/ | |
|  | MESH\_EVENT\_VOTE\_STOPPED, /\*\*< the process of voting a new root is stopped \*/ | |
|  | MESH\_EVENT\_ROOT\_ADDRESS, /\*\*< the root address is obtained. It is posted by mesh stack automatically. \*/ | |
|  | MESH\_EVENT\_ROOT\_SWITCH\_REQ, /\*\*< root switch request sent from a new voted root candidate \*/ | |
|  | MESH\_EVENT\_ROOT\_SWITCH\_ACK, /\*\*< root switch acknowledgment responds the above request sent from current root \*/ | |
|  | MESH\_EVENT\_ROOT\_ASKED\_YIELD, /\*\*< the root is asked yield by a more powerful existing root. If self organized is disabled | |
|  | and this device is specified to be a root by users, users should set a new parent | |
|  | for this device. if self organized is enabled, this device will find a new parent | |
|  | by itself, users could ignore this event. \*/ | |
|  | MESH\_EVENT\_ROOT\_FIXED, /\*\*< when devices join a network, if the setting of Fixed Root for one device is different | |
|  | from that of its parent, the device will update the setting the same as its parent's. | |
|  | Fixed Root Setting of each device is variable as that setting changes of the root. \*/ | |
|  | MESH\_EVENT\_SCAN\_DONE, /\*\*< if self-organized networking is disabled, user can call esp\_wifi\_scan\_start() to trigger | |
|  | this event, and add the corresponding scan done handler in this event. \*/ | |
|  | MESH\_EVENT\_NETWORK\_STATE, /\*\*< network state, such as whether current mesh network has a root. \*/ | |
|  | MESH\_EVENT\_STOP\_RECONNECTION, /\*\*< the root stops reconnecting to the router and non-root devices stop reconnecting to their parents. \*/ | |
|  | MESH\_EVENT\_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. \*/ | |
|  | MESH\_EVENT\_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. \*/ | |
|  | MESH\_EVENT\_PS\_PARENT\_DUTY, /\*\*< parent duty \*/ | |
|  | MESH\_EVENT\_PS\_CHILD\_DUTY, /\*\*< child duty \*/ | |
|  | MESH\_EVENT\_MAX, | |
|  | } mesh\_event\_id\_t; | |
|  |  | |
|  | /\*\* @brief ESP-MESH event base declaration \*/ | |
|  | ESP\_EVENT\_DECLARE\_BASE(MESH\_EVENT); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Device type | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_IDLE, /\*\*< hasn't joined the mesh network yet \*/ | |
|  | MESH\_ROOT, /\*\*< the only sink of the mesh network. Has the ability to access external IP network \*/ | |
|  | MESH\_NODE, /\*\*< intermediate device. Has the ability to forward packets over the mesh network \*/ | |
|  | MESH\_LEAF, /\*\*< has no forwarding ability \*/ | |
|  | MESH\_STA, /\*\*< connect to router with a standlone Wi-Fi station mode, no network expansion capability \*/ | |
|  | } mesh\_type\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Protocol of transmitted application data | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_PROTO\_BIN, /\*\*< binary \*/ | |
|  | MESH\_PROTO\_HTTP, /\*\*< HTTP protocol \*/ | |
|  | MESH\_PROTO\_JSON, /\*\*< JSON format \*/ | |
|  | MESH\_PROTO\_MQTT, /\*\*< MQTT protocol \*/ | |
|  | MESH\_PROTO\_AP, /\*\*< IP network mesh communication of node's AP inteface \*/ | |
|  | MESH\_PROTO\_STA, /\*\*< IP network mesh communication of node's STA inteface \*/ | |
|  | } mesh\_proto\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief For reliable transmission, mesh stack provides three type of services | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_TOS\_P2P, /\*\*< provide P2P (point-to-point) retransmission on mesh stack by default \*/ | |
|  | MESH\_TOS\_E2E, /\*\*< provide E2E (end-to-end) retransmission on mesh stack (Unimplemented) \*/ | |
|  | MESH\_TOS\_DEF, /\*\*< no retransmission on mesh stack \*/ | |
|  | } mesh\_tos\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Vote reason | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_VOTE\_REASON\_ROOT\_INITIATED = 1, /\*\*< vote is initiated by the root \*/ | |
|  | MESH\_VOTE\_REASON\_CHILD\_INITIATED, /\*\*< vote is initiated by children \*/ | |
|  | } mesh\_vote\_reason\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh disconnect reason code | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_REASON\_CYCLIC = 100, /\*\*< cyclic is detected \*/ | |
|  | MESH\_REASON\_PARENT\_IDLE, /\*\*< parent is idle \*/ | |
|  | MESH\_REASON\_LEAF, /\*\*< the connected device is changed to a leaf \*/ | |
|  | MESH\_REASON\_DIFF\_ID, /\*\*< in different mesh ID \*/ | |
|  | MESH\_REASON\_ROOTS, /\*\*< root conflict is detected \*/ | |
|  | MESH\_REASON\_PARENT\_STOPPED, /\*\*< parent has stopped the mesh \*/ | |
|  | MESH\_REASON\_SCAN\_FAIL, /\*\*< scan fail \*/ | |
|  | MESH\_REASON\_IE\_UNKNOWN, /\*\*< unknown IE \*/ | |
|  | MESH\_REASON\_WAIVE\_ROOT, /\*\*< waive root \*/ | |
|  | MESH\_REASON\_PARENT\_WORSE, /\*\*< parent with very poor RSSI \*/ | |
|  | MESH\_REASON\_EMPTY\_PASSWORD, /\*\*< use an empty password to connect to an encrypted parent \*/ | |
|  | MESH\_REASON\_PARENT\_UNENCRYPTED, /\*\*< connect to an unencrypted parent/router \*/ | |
|  | } mesh\_disconnect\_reason\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh topology | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_TOPO\_TREE, /\*\*< tree topology \*/ | |
|  | MESH\_TOPO\_CHAIN, /\*\*< chain topology \*/ | |
|  | } esp\_mesh\_topology\_t; | |
|  |  | |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
|  | \* Structures | |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |
|  | /\*\* | |
|  | \* @brief IP address and port | |
|  | \*/ | |
|  | typedef struct { | |
|  | ip4\_addr\_t ip4; /\*\*< IP address \*/ | |
|  | uint16\_t port; /\*\*< port \*/ | |
|  | } \_\_attribute\_\_((packed)) mip\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh address | |
|  | \*/ | |
|  | typedef union { | |
|  | uint8\_t addr[6]; /\*\*< mac address \*/ | |
|  | mip\_t mip; /\*\*< mip address \*/ | |
|  | } mesh\_addr\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Channel switch information | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t channel; /\*\*< new channel \*/ | |
|  | } mesh\_event\_channel\_switch\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Parent connected information | |
|  | \*/ | |
|  | typedef struct { | |
|  | wifi\_event\_sta\_connected\_t connected; /\*\*< parent information, same as Wi-Fi event SYSTEM\_EVENT\_STA\_CONNECTED does \*/ | |
|  | uint16\_t self\_layer; /\*\*< layer \*/ | |
|  | uint8\_t duty; /\*\*< parent duty \*/ | |
|  | } mesh\_event\_connected\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief No parent found information | |
|  | \*/ | |
|  | typedef struct { | |
|  | int scan\_times; /\*\*< scan times being through \*/ | |
|  | } mesh\_event\_no\_parent\_found\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Layer change information | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint16\_t new\_layer; /\*\*< new layer \*/ | |
|  | } mesh\_event\_layer\_change\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief The reachability of the root to a DS (distribute system) | |
|  | \*/ | |
|  | typedef enum { | |
|  | MESH\_TODS\_UNREACHABLE, /\*\*< the root isn't able to access external IP network \*/ | |
|  | MESH\_TODS\_REACHABLE, /\*\*< the root is able to access external IP network \*/ | |
|  | } mesh\_event\_toDS\_state\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief vote started information | |
|  | \*/ | |
|  | typedef struct { | |
|  | int reason; /\*\*< vote reason, vote could be initiated by children or by the root itself \*/ | |
|  | int attempts; /\*\*< max vote attempts before stopped \*/ | |
|  | mesh\_addr\_t rc\_addr; /\*\*< root address specified by users via API esp\_mesh\_waive\_root() \*/ | |
|  | } mesh\_event\_vote\_started\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief find a mesh network that this device can join | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t channel; /\*\*< channel number of the new found network \*/ | |
|  | uint8\_t router\_bssid[6]; /\*\*< router BSSID \*/ | |
|  | } mesh\_event\_find\_network\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Root address | |
|  | \*/ | |
|  | typedef mesh\_addr\_t mesh\_event\_root\_address\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Parent disconnected information | |
|  | \*/ | |
|  | typedef wifi\_event\_sta\_disconnected\_t mesh\_event\_disconnected\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Child connected information | |
|  | \*/ | |
|  | typedef wifi\_event\_ap\_staconnected\_t mesh\_event\_child\_connected\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Child disconnected information | |
|  | \*/ | |
|  | typedef wifi\_event\_ap\_stadisconnected\_t mesh\_event\_child\_disconnected\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Root switch request information | |
|  | \*/ | |
|  | typedef struct { | |
|  | int reason; /\*\*< root switch reason, generally root switch is initialized by users via API esp\_mesh\_waive\_root() \*/ | |
|  | mesh\_addr\_t rc\_addr; /\*\*< the address of root switch requester \*/ | |
|  | } mesh\_event\_root\_switch\_req\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Other powerful root address | |
|  | \*/ | |
|  | typedef struct { | |
|  | int8\_t rssi; /\*\*< rssi with router \*/ | |
|  | uint16\_t capacity; /\*\*< the number of devices in current network \*/ | |
|  | uint8\_t addr[6]; /\*\*< other powerful root address \*/ | |
|  | } mesh\_event\_root\_conflict\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Routing table change | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint16\_t rt\_size\_new; /\*\*< the new value \*/ | |
|  | uint16\_t rt\_size\_change; /\*\*< the changed value \*/ | |
|  | } mesh\_event\_routing\_table\_change\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Root fixed | |
|  | \*/ | |
|  | typedef struct { | |
|  | bool is\_fixed; /\*\*< status \*/ | |
|  | } mesh\_event\_root\_fixed\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Scan done　event information | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t number; /\*\*< the number of APs scanned \*/ | |
|  | } mesh\_event\_scan\_done\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Network state information | |
|  | \*/ | |
|  | typedef struct { | |
|  | bool is\_rootless; /\*\*< whether current mesh network has a root \*/ | |
|  | } mesh\_event\_network\_state\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief New router information | |
|  | \*/ | |
|  | typedef wifi\_event\_sta\_connected\_t mesh\_event\_router\_switch\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief PS duty information | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t duty; /\*\*< parent or child duty \*/ | |
|  | mesh\_event\_child\_connected\_t child\_connected; /\*\*< child info \*/ | |
|  | } mesh\_event\_ps\_duty\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh event information | |
|  | \*/ | |
|  | typedef union { | |
|  | mesh\_event\_channel\_switch\_t channel\_switch; /\*\*< channel switch \*/ | |
|  | mesh\_event\_child\_connected\_t child\_connected; /\*\*< child connected \*/ | |
|  | mesh\_event\_child\_disconnected\_t child\_disconnected; /\*\*< child disconnected \*/ | |
|  | mesh\_event\_routing\_table\_change\_t routing\_table; /\*\*< routing table change \*/ | |
|  | mesh\_event\_connected\_t connected; /\*\*< parent connected \*/ | |
|  | mesh\_event\_disconnected\_t disconnected; /\*\*< parent disconnected \*/ | |
|  | mesh\_event\_no\_parent\_found\_t no\_parent; /\*\*< no parent found \*/ | |
|  | mesh\_event\_layer\_change\_t layer\_change; /\*\*< layer change \*/ | |
|  | mesh\_event\_toDS\_state\_t toDS\_state; /\*\*< toDS state, devices shall check this state firstly before trying to send packets to | |
|  | external IP network. This state indicates right now whether the root is capable of sending | |
|  | packets out. If not, devices had better to wait until this state changes to be | |
|  | MESH\_TODS\_REACHABLE. \*/ | |
|  | mesh\_event\_vote\_started\_t vote\_started; /\*\*< vote started \*/ | |
|  | mesh\_event\_root\_address\_t root\_addr; /\*\*< root address \*/ | |
|  | mesh\_event\_root\_switch\_req\_t switch\_req; /\*\*< root switch request \*/ | |
|  | mesh\_event\_root\_conflict\_t root\_conflict; /\*\*< other powerful root \*/ | |
|  | mesh\_event\_root\_fixed\_t root\_fixed; /\*\*< fixed root \*/ | |
|  | mesh\_event\_scan\_done\_t scan\_done; /\*\*< scan done \*/ | |
|  | mesh\_event\_network\_state\_t network\_state; /\*\*< network state, such as whether current mesh network has a root. \*/ | |
|  | mesh\_event\_find\_network\_t find\_network; /\*\*< network found that can join \*/ | |
|  | mesh\_event\_router\_switch\_t router\_switch; /\*\*< new router information \*/ | |
|  | mesh\_event\_ps\_duty\_t ps\_duty; /\*\*< PS duty information \*/ | |
|  | } mesh\_event\_info\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh option | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t type; /\*\*< option type \*/ | |
|  | uint16\_t len; /\*\*< option length \*/ | |
|  | uint8\_t \*val; /\*\*< option value \*/ | |
|  | } \_\_attribute\_\_((packed)) mesh\_opt\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh data for esp\_mesh\_send() and esp\_mesh\_recv() | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t \*data; /\*\*< data \*/ | |
|  | uint16\_t size; /\*\*< data size \*/ | |
|  | mesh\_proto\_t proto; /\*\*< data protocol \*/ | |
|  | mesh\_tos\_t tos; /\*\*< data type of service \*/ | |
|  | } mesh\_data\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Router configuration | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t ssid[32]; /\*\*< SSID \*/ | |
|  | uint8\_t ssid\_len; /\*\*< length of SSID \*/ | |
|  | uint8\_t bssid[6]; /\*\*< BSSID, if this value is specified, users should also specify "allow\_router\_switch". \*/ | |
|  | uint8\_t password[64]; /\*\*< password \*/ | |
|  | bool allow\_router\_switch; /\*\*< if the BSSID is specified and this value is also set, when the router of this specified BSSID | |
|  | fails to be found after "fail" (mesh\_attempts\_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. | |
|  | The default value is false. | |
|  | 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. \*/ | |
|  | } mesh\_router\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh softAP configuration | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t password[64]; /\*\*< mesh softAP password \*/ | |
|  | uint8\_t max\_connection; /\*\*< max number of stations allowed to connect in, max 10 \*/ | |
|  | } mesh\_ap\_cfg\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh initialization configuration | |
|  | \*/ | |
|  | typedef struct { | |
|  | uint8\_t channel; /\*\*< channel, the mesh network on \*/ | |
|  | bool allow\_channel\_switch; /\*\*< if this value is set, when "fail" (mesh\_attempts\_t) times is reached, device will change to | |
|  | a full channel scan for a network that could join. The default value is false. \*/ | |
|  | mesh\_addr\_t mesh\_id; /\*\*< mesh network identification \*/ | |
|  | mesh\_router\_t router; /\*\*< router configuration \*/ | |
|  | mesh\_ap\_cfg\_t mesh\_ap; /\*\*< mesh softAP configuration \*/ | |
|  | const mesh\_crypto\_funcs\_t \*crypto\_funcs; /\*\*< crypto functions \*/ | |
|  | } mesh\_cfg\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Vote address configuration | |
|  | \*/ | |
|  | typedef union { | |
|  | int attempts; /\*\*< max vote attempts before a new root is elected automatically by mesh network. (min:15, 15 by default) \*/ | |
|  | mesh\_addr\_t rc\_addr; /\*\*< a new root address specified by users for API esp\_mesh\_waive\_root() \*/ | |
|  | } mesh\_rc\_config\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Vote | |
|  | \*/ | |
|  | typedef struct { | |
|  | float percentage; /\*\*< vote percentage threshold for approval of being a root \*/ | |
|  | bool is\_rc\_specified; /\*\*< if true, rc\_addr shall be specified (Unimplemented). | |
|  | if false, attempts value shall be specified to make network start root election. \*/ | |
|  | mesh\_rc\_config\_t config; /\*\*< vote address configuration \*/ | |
|  | } mesh\_vote\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief The number of packets pending in the queue waiting to be sent by the mesh stack | |
|  | \*/ | |
|  | typedef struct { | |
|  | int to\_parent; /\*\*< to parent queue \*/ | |
|  | int to\_parent\_p2p; /\*\*< to parent (P2P) queue \*/ | |
|  | int to\_child; /\*\*< to child queue \*/ | |
|  | int to\_child\_p2p; /\*\*< to child (P2P) queue \*/ | |
|  | int mgmt; /\*\*< management queue \*/ | |
|  | int broadcast; /\*\*< broadcast and multicast queue \*/ | |
|  | } mesh\_tx\_pending\_t; | |
|  |  | |
|  | /\*\* | |
|  | \* @brief The number of packets available in the queue waiting to be received by applications | |
|  | \*/ | |
|  | typedef struct { | |
|  | int toDS; /\*\*< to external DS \*/ | |
|  | int toSelf; /\*\*< to self \*/ | |
|  | } mesh\_rx\_pending\_t; | |
|  |  | |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
|  | \* Variable Declaration | |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |
|  | /\* mesh IE crypto callback function \*/ | |
|  | extern const mesh\_crypto\_funcs\_t g\_wifi\_default\_mesh\_crypto\_funcs; | |
|  |  | |
|  | #define MESH\_INIT\_CONFIG\_DEFAULT() { \ | |
|  | .crypto\_funcs = &g\_wifi\_default\_mesh\_crypto\_funcs, \ | |
|  | } | |
|  |  | |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
|  | \* Function Definitions | |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |
|  | /\*\* | |
|  | \* @brief Mesh initialization | |
|  | \* - Check whether Wi-Fi is started. | |
|  | \* - Initialize mesh global variables with default values. | |
|  | \* | |
|  | \* @attention This API shall be called after Wi-Fi is started. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_init(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Mesh de-initialization | |
|  | \* | |
|  | \* - Release resources and stop the mesh | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_deinit(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Start mesh | |
|  | \* - Initialize mesh IE. | |
|  | \* - Start mesh network management service. | |
|  | \* - Create TX and RX queues according to the configuration. | |
|  | \* - Register mesh packets receive callback. | |
|  | \* | |
|  | \* @attention　　This API shall be called after mesh initialization and configuration. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \* - ESP\_ERR\_MESH\_NOT\_INIT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_CONFIG | |
|  | \* - ESP\_ERR\_MESH\_NO\_MEMORY | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_start(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Stop mesh | |
|  | \* - Deinitialize mesh IE. | |
|  | \* - Disconnect with current parent. | |
|  | \* - Disassociate all currently associated children. | |
|  | \* - Stop mesh network management service. | |
|  | \* - Unregister mesh packets receive callback. | |
|  | \* - Delete TX and RX queues. | |
|  | \* - Release resources. | |
|  | \* - Restore Wi-Fi softAP to default settings if Wi-Fi dual mode is enabled. | |
|  | \* - Set Wi-Fi Power Save type to WIFI\_PS\_NONE. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_stop(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Send a packet over the mesh network | |
|  | \* - Send a packet to any device in the mesh network. | |
|  | \* - Send a packet to external IP network. | |
|  | \* | |
|  | \* @attention This API is not reentrant. | |
|  | \* | |
|  | \* @param[in] to the address of the final destination of the packet | |
|  | \* - If the packet is to the root, set this parameter to NULL. | |
|  | \* - If the packet is to an external IP network, set this parameter to the IPv4:PORT combination. | |
|  | \* This packet will be delivered to the root firstly, then the root will forward this packet to the final IP server address. | |
|  | \* @param[in] data pointer to a sending mesh packet | |
|  | \* - Field size should not exceed MESH\_MPS. Note that the size of one mesh packet should not exceed MESH\_MTU. | |
|  | \* - Field proto should be set to data protocol in use (default is MESH\_PROTO\_BIN for binary). | |
|  | \* - Field tos should be set to transmission tos (type of service) in use (default is MESH\_TOS\_P2P for point-to-point reliable). | |
|  | \* @param[in] flag bitmap for data sent | |
|  | \* - Speed up the route search | |
|  | \* - If the packet is to the root and "to" parameter is NULL, set this parameter to 0. | |
|  | \* - If the packet is to an internal device, MESH\_DATA\_P2P should be set. | |
|  | \* - If the packet is to the root ("to" parameter isn't NULL) or to external IP network, MESH\_DATA\_TODS should be set. | |
|  | \* - If the packet is from the root to an internal device, MESH\_DATA\_FROMDS should be set. | |
|  | \* - Specify whether this API is block or non-block, block by default | |
|  | \* - If needs non-block, MESH\_DATA\_NONBLOCK should be set. | |
|  | \* - In the situation of the root change, MESH\_DATA\_DROP identifies this packet can be dropped by the new root | |
|  | \* for upstream data to external IP network, we try our best to avoid data loss caused by the root change, but | |
|  | \* there is a risk that the new root is running out of memory because most of memory is occupied by the pending data which | |
|  | \* isn't read out in time by esp\_mesh\_recv\_toDS(). | |
|  | \* | |
|  | \* Generally, we suggest esp\_mesh\_recv\_toDS() is called after a connection with IP network is created. Thus data outgoing | |
|  | \* to external IP network via socket is just from reading esp\_mesh\_recv\_toDS() which avoids unnecessary memory copy. | |
|  | \* | |
|  | \* @param[in] opt options | |
|  | \* - In case of sending a packet to a certain group, MESH\_OPT\_SEND\_GROUP is a good choice. | |
|  | \* In this option, the value field should be set to the target receiver addresses in this group. | |
|  | \* - Root sends a packet to an internal device, this packet is from external IP network in case the receiver device responds | |
|  | \* this packet, MESH\_OPT\_RECV\_DS\_ADDR is required to attach the target DS address. | |
|  | \* @param[in] opt\_count option count | |
|  | \* - Currently, this API only takes one option, so opt\_count is only supported to be 1. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \* - 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 | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_send(const mesh\_addr\_t \*to, const mesh\_data\_t \*data, | |
|  | int flag, const mesh\_opt\_t opt[], int opt\_count); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Receive a packet targeted to self over the mesh network | |
|  | \* | |
|  | \* @attention Mesh RX queue should be checked regularly to avoid running out of memory. | |
|  | \* - Use esp\_mesh\_get\_rx\_pending() to check the number of packets available in the queue waiting | |
|  | \* to be received by applications. | |
|  | \* | |
|  | \* @param[out] from the address of the original source of the packet | |
|  | \* @param[out] data pointer to the received mesh packet | |
|  | \* - Field proto is the data protocol in use. Should follow it to parse the received data. | |
|  | \* - Field tos is the transmission tos (type of service) in use. | |
|  | \* @param[in] timeout\_ms wait time if a packet isn't immediately available (0:no wait, portMAX\_DELAY:wait forever) | |
|  | \* @param[out] flag bitmap for data received | |
|  | \* - MESH\_DATA\_FROMDS represents data from external IP network | |
|  | \* - MESH\_DATA\_TODS represents data directed upward within the mesh network | |
|  | \* | |
|  | \* flag could be MESH\_DATA\_FROMDS or MESH\_DATA\_TODS. | |
|  | \* @param[out] opt options desired to receive | |
|  | \* - MESH\_OPT\_RECV\_DS\_ADDR attaches the DS address | |
|  | \* @param[in] opt\_count option count desired to receive | |
|  | \* - Currently, this API only takes one option, so opt\_count is only supported to be 1. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_START | |
|  | \* - ESP\_ERR\_MESH\_TIMEOUT | |
|  | \* - ESP\_ERR\_MESH\_DISCARD | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_recv(mesh\_addr\_t \*from, mesh\_data\_t \*data, int timeout\_ms, | |
|  | int \*flag, mesh\_opt\_t opt[], int opt\_count); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief 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. | |
|  | \* - If no socket connection is ready to send out the received packets and this esp\_mesh\_recv\_toDS() | |
|  | \* hasn't been called by applications, packets from the whole mesh network will be pending in toDS queue. | |
|  | \* | |
|  | \* Use esp\_mesh\_get\_rx\_pending() to check the number of packets available in the queue waiting | |
|  | \* to be received by applications in case of running out of memory in the root. | |
|  | \* | |
|  | \* Using esp\_mesh\_set\_xon\_qsize() users may configure the RX queue size, default:32. If this size is too large, | |
|  | \* and esp\_mesh\_recv\_toDS() isn't called in time, there is a risk that a great deal of memory is occupied | |
|  | \* by the pending packets. If this size is too small, it will impact the efficiency on upstream. How to | |
|  | \* decide this value depends on the specific application scenarios. | |
|  | \* | |
|  | \* @attention This API is only called by the root. | |
|  | \* | |
|  | \* @param[out] from the address of the original source of the packet | |
|  | \* @param[out] to the address contains remote IP address and port (IPv4:PORT) | |
|  | \* @param[out] data pointer to the received packet | |
|  | \* - Contain the protocol and applications should follow it to parse the data. | |
|  | \* @param[in] timeout\_ms wait time if a packet isn't immediately available (0:no wait, portMAX\_DELAY:wait forever) | |
|  | \* @param[out] flag bitmap for data received | |
|  | \* - MESH\_DATA\_TODS represents the received data target to external IP network. Root shall forward this data to external IP network via the association with router. | |
|  | \* | |
|  | \* flag could be MESH\_DATA\_TODS. | |
|  | \* @param[out] opt options desired to receive | |
|  | \* @param[in] opt\_count option count desired to receive | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_START | |
|  | \* - ESP\_ERR\_MESH\_TIMEOUT | |
|  | \* - ESP\_ERR\_MESH\_DISCARD | |
|  | \* - ESP\_ERR\_MESH\_RECV\_RELEASE | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_recv\_toDS(mesh\_addr\_t \*from, mesh\_addr\_t \*to, | |
|  | mesh\_data\_t \*data, int timeout\_ms, int \*flag, mesh\_opt\_t opt[], | |
|  | int opt\_count); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh stack configuration | |
|  | \* - Use MESH\_INIT\_CONFIG\_DEFAULT() to initialize the default values, mesh IE is encrypted by default. | |
|  | \* - Mesh network is established on a fixed channel (1-14). | |
|  | \* - Mesh event callback is mandatory. | |
|  | \* - Mesh ID is an identifier of an MBSS. Nodes with the same mesh ID can communicate with each other. | |
|  | \* - Regarding to the router configuration, if the router is hidden, BSSID field is mandatory. | |
|  | \* | |
|  | \* If BSSID field isn't set and there exists more than one router with same SSID, there is a risk that more | |
|  | \* roots than one connected with different BSSID will appear. It means more than one mesh network is established | |
|  | \* with the same mesh ID. | |
|  | \* | |
|  | \* Root conflict function could eliminate redundant roots connected with the same BSSID, but couldn't handle roots | |
|  | \* connected with different BSSID. Because users might have such requirements of setting up routers with same SSID | |
|  | \* for the future replacement. But in that case, if the above situations happen, please make sure applications | |
|  | \* implement forward functions on the root to guarantee devices in different mesh networks can communicate with each other. | |
|  | \* max\_connection of mesh softAP is limited by the max number of Wi-Fi softAP supported (max:10). | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started after mesh is initialized. | |
|  | \* | |
|  | \* @param[in] config pointer to mesh stack configuration | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_config(const mesh\_cfg\_t \*config); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh stack configuration | |
|  | \* | |
|  | \* @param[out] config pointer to mesh stack configuration | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_config(mesh\_cfg\_t \*config); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get router configuration | |
|  | \* | |
|  | \* @attention This API is used to dynamically modify the router configuration after mesh is configured. | |
|  | \* | |
|  | \* @param[in] router pointer to router configuration | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_router(const mesh\_router\_t \*router); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get router configuration | |
|  | \* | |
|  | \* @param[out] router pointer to router configuration | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_router(mesh\_router\_t \*router); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh network ID | |
|  | \* | |
|  | \* @attention This API is used to dynamically modify the mesh network ID. | |
|  | \* | |
|  | \* @param[in] id pointer to mesh network ID | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT: invalid argument | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_id(const mesh\_addr\_t \*id); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh network ID | |
|  | \* | |
|  | \* @param[out] id pointer to mesh network ID | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_id(mesh\_addr\_t \*id); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Designate device type over the mesh network | |
|  | \* - MESH\_IDLE: designates a device as a self-organized node for a mesh network | |
|  | \* - MESH\_ROOT: designates the root node for a mesh network | |
|  | \* - MESH\_LEAF: designates a device as a standalone Wi-Fi station that connects to a parent | |
|  | \* - MESH\_STA: designates a device as a standalone Wi-Fi station that connects to a router | |
|  | \* | |
|  | \* @param[in] type device type | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_type(mesh\_type\_t type); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get device type over mesh network | |
|  | \* | |
|  | \* @attention This API shall be called after having received the event MESH\_EVENT\_PARENT\_CONNECTED. | |
|  | \* | |
|  | \* @return mesh type | |
|  | \* | |
|  | \*/ | |
|  | mesh\_type\_t esp\_mesh\_get\_type(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set network max layer value | |
|  | \* - for tree topology, the max is 25. | |
|  | \* - for chain topology, the max is 1000. | |
|  | \* - Network max layer limits the max hop count. | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] max\_layer max layer value | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_max\_layer(int max\_layer); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get max layer value | |
|  | \* | |
|  | \* @return max layer value | |
|  | \*/ | |
|  | int esp\_mesh\_get\_max\_layer(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh softAP password | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] pwd pointer to the password | |
|  | \* @param[in] len password length | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_ap\_password(const uint8\_t \*pwd, int len); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh softAP authentication mode | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] authmode authentication mode | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_ap\_authmode(wifi\_auth\_mode\_t authmode); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh softAP authentication mode | |
|  | \* | |
|  | \* @return authentication mode | |
|  | \*/ | |
|  | wifi\_auth\_mode\_t esp\_mesh\_get\_ap\_authmode(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh softAP max connection value | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] connections the number of max connections | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_ap\_connections(int connections); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh softAP max connection configuration | |
|  | \* | |
|  | \* @return the number of max connections | |
|  | \*/ | |
|  | int esp\_mesh\_get\_ap\_connections(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get current layer value over the mesh network | |
|  | \* | |
|  | \* @attention This API shall be called after having received the event MESH\_EVENT\_PARENT\_CONNECTED. | |
|  | \* | |
|  | \* @return layer value | |
|  | \* | |
|  | \*/ | |
|  | int esp\_mesh\_get\_layer(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the parent BSSID | |
|  | \* | |
|  | \* @attention This API shall be called after having received the event MESH\_EVENT\_PARENT\_CONNECTED. | |
|  | \* | |
|  | \* @param[out] bssid pointer to parent BSSID | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_parent\_bssid(mesh\_addr\_t \*bssid); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Return whether the device is the root node of the network | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_is\_root(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Enable/disable self-organized networking | |
|  | \* - Self-organized networking has three main functions: | |
|  | \* select the root node; | |
|  | \* find a preferred parent; | |
|  | \* initiate reconnection if a disconnection is detected. | |
|  | \* - Self-organized networking is enabled by default. | |
|  | \* - If self-organized is disabled, users should set a parent for the device via esp\_mesh\_set\_parent(). | |
|  | \* | |
|  | \* @attention This API is used to dynamically modify whether to enable the self organizing. | |
|  | \* | |
|  | \* @param[in] enable enable or disable self-organized networking | |
|  | \* @param[in] select\_parent Only valid when self-organized networking is enabled. | |
|  | \* - if select\_parent is set to true, the root will give up its mesh root status and search for a new parent | |
|  | \* like other non-root devices. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_self\_organized(bool enable, bool select\_parent); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Return whether enable self-organized networking or not | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_get\_self\_organized(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Cause the root device to give up (waive) its mesh root status | |
|  | \* - A device is elected root primarily based on RSSI from the external router. | |
|  | \* - If external router conditions change, users can call this API to perform a root switch. | |
|  | \* - In this API, users could specify a desired root address to replace itself or specify an attempts value | |
|  | \* to ask current root to initiate a new round of voting. During the voting, a better root candidate would | |
|  | \* be expected to find to replace the current one. | |
|  | \* - If no desired root candidate, the vote will try a specified number of attempts (at least 15). If no better | |
|  | \* root candidate is found, keep the current one. If a better candidate is found, the new better one will | |
|  | \* send a root switch request to the current root, current root will respond with a root switch acknowledgment. | |
|  | \* - After that, the new candidate will connect to the router to be a new root, the previous root will disconnect | |
|  | \* with the router and choose another parent instead. | |
|  | \* | |
|  | \* Root switch is completed with minimal disruption to the whole mesh network. | |
|  | \* | |
|  | \* @attention This API is only called by the root. | |
|  | \* | |
|  | \* @param[in] vote vote configuration | |
|  | \* - If this parameter is set NULL, the vote will perform the default 15 times. | |
|  | \* | |
|  | \* - Field percentage threshold is 0.9 by default. | |
|  | \* - Field is\_rc\_specified shall be false. | |
|  | \* - Field attempts shall be at least 15 times. | |
|  | \* @param[in] reason only accept MESH\_VOTE\_REASON\_ROOT\_INITIATED for now | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_QUEUE\_FULL | |
|  | \* - ESP\_ERR\_MESH\_DISCARD | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_waive\_root(const mesh\_vote\_t \*vote, int reason); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set vote percentage threshold for approval of being a root (default:0.9) | |
|  | \* - During the networking, only obtaining vote percentage reaches this threshold, | |
|  | \* the device could be a root. | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] percentage vote percentage threshold | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_vote\_percentage(float percentage); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get vote percentage threshold for approval of being a root | |
|  | \* | |
|  | \* @return percentage threshold | |
|  | \*/ | |
|  | float esp\_mesh\_get\_vote\_percentage(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh softAP associate expired time (default:10 seconds) | |
|  | \* - If mesh softAP hasn't received any data from an associated child within this time, | |
|  | \* mesh softAP will take this child inactive and disassociate it. | |
|  | \* - If mesh softAP is encrypted, this value should be set a greater value, such as 30 seconds. | |
|  | \* | |
|  | \* @param[in] seconds the expired time | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_ap\_assoc\_expire(int seconds); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh softAP associate expired time | |
|  | \* | |
|  | \* @return seconds | |
|  | \*/ | |
|  | int esp\_mesh\_get\_ap\_assoc\_expire(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get total number of devices in current network (including the root) | |
|  | \* | |
|  | \* @attention The returned value might be incorrect when the network is changing. | |
|  | \*\* | |
|  | \* @return total number of devices (including the root) | |
|  | \*/ | |
|  | int esp\_mesh\_get\_total\_node\_num(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the number of devices in this device's sub-network (including self) | |
|  | \* | |
|  | \* @return the number of devices over this device's sub-network (including self) | |
|  | \*/ | |
|  | int esp\_mesh\_get\_routing\_table\_size(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get routing table of this device's sub-network (including itself) | |
|  | \* | |
|  | \* @param[out] mac pointer to routing table | |
|  | \* @param[in] len routing table size(in bytes) | |
|  | \* @param[out] size pointer to the number of devices in routing table (including itself) | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_routing\_table(mesh\_addr\_t \*mac, int len, int \*size); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Post the toDS state to the mesh stack | |
|  | \* | |
|  | \* @attention This API is only for the root. | |
|  | \* | |
|  | \* @param[in] reachable this state represents whether the root is able to access external IP network | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_post\_toDS\_state(bool reachable); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Return the number of packets pending in the queue waiting to be sent by the mesh stack | |
|  | \* | |
|  | \* @param[out] pending pointer to the TX pending | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_tx\_pending(mesh\_tx\_pending\_t \*pending); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Return the number of packets available in the queue waiting to be received by applications | |
|  | \* | |
|  | \* @param[out] pending pointer to the RX pending | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_rx\_pending(mesh\_rx\_pending\_t \*pending); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Return the number of packets could be accepted from the specified address | |
|  | \* | |
|  | \* @param[in] addr self address or an associate children address | |
|  | \* @param[out] xseqno\_in sequence number of the last received packet from the specified address | |
|  | \* | |
|  | \* @return the number of upQ for a certain address | |
|  | \*/ | |
|  | int esp\_mesh\_available\_txupQ\_num(const mesh\_addr\_t \*addr, uint32\_t \*xseqno\_in); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set the number of queue | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] qsize default:32 (min:16) | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_xon\_qsize(int qsize); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get queue size | |
|  | \* | |
|  | \* @return the number of queue | |
|  | \*/ | |
|  | int esp\_mesh\_get\_xon\_qsize(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set whether allow more than one root existing in one network | |
|  | \* | |
|  | \* @param[in] allowed allow or not | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_WIFI\_ERR\_NOT\_INIT | |
|  | \* - ESP\_WIFI\_ERR\_NOT\_START | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_allow\_root\_conflicts(bool allowed); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Check whether allow more than one root to exist in one network | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_is\_root\_conflicts\_allowed(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set group ID addresses | |
|  | \* | |
|  | \* @param[in] addr pointer to new group ID addresses | |
|  | \* @param[in] num the number of group ID addresses | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_group\_id(const mesh\_addr\_t \*addr, int num); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Delete group ID addresses | |
|  | \* | |
|  | \* @param[in] addr pointer to deleted group ID address | |
|  | \* @param[in] num the number of group ID addresses | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_delete\_group\_id(const mesh\_addr\_t \*addr, int num); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the number of group ID addresses | |
|  | \* | |
|  | \* @return the number of group ID addresses | |
|  | \*/ | |
|  | int esp\_mesh\_get\_group\_num(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get group ID addresses | |
|  | \* | |
|  | \* @param[out] addr pointer to group ID addresses | |
|  | \* @param[in] num the number of group ID addresses | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_group\_list(mesh\_addr\_t \*addr, int num); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Check whether the specified group address is my group | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_is\_my\_group(const mesh\_addr\_t \*addr); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh network capacity (max:1000, default:300) | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] num mesh network capacity | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_capacity\_num(int num); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh network capacity | |
|  | \* | |
|  | \* @return mesh network capacity | |
|  | \*/ | |
|  | int esp\_mesh\_get\_capacity\_num(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh IE crypto functions | |
|  | \* | |
|  | \* @attention This API can be called at any time after mesh is initialized. | |
|  | \* | |
|  | \* @param[in] crypto\_funcs crypto functions for mesh IE | |
|  | \* - If crypto\_funcs is set to NULL, mesh IE is no longer encrypted. | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_ie\_crypto\_funcs(const mesh\_crypto\_funcs\_t \*crypto\_funcs); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh IE crypto key | |
|  | \* | |
|  | \* @attention This API can be called at any time after mesh is initialized. | |
|  | \* | |
|  | \* @param[in] key ASCII crypto key | |
|  | \* @param[in] len length in bytes, range:8~64 | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_ie\_crypto\_key(const char \*key, int len); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh IE crypto key | |
|  | \* | |
|  | \* @param[out] key ASCII crypto key | |
|  | \* @param[in] len length in bytes, range:8~64 | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_ie\_crypto\_key(char \*key, int len); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set delay time before starting root healing | |
|  | \* | |
|  | \* @param[in] delay\_ms delay time in milliseconds | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_root\_healing\_delay(int delay\_ms); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get delay time before network starts root healing | |
|  | \* | |
|  | \* @return delay time in milliseconds | |
|  | \*/ | |
|  | int esp\_mesh\_get\_root\_healing\_delay(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Enable network Fixed Root Setting | |
|  | \* - Enabling fixed root disables automatic election of the root node via voting. | |
|  | \* - All devices in the network shall use the same Fixed Root Setting (enabled or disabled). | |
|  | \* - If Fixed Root is enabled, users should make sure a root node is designated for the network. | |
|  | \* | |
|  | \* @param[in] enable enable or not | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_fix\_root(bool enable); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Check whether network Fixed Root Setting is enabled | |
|  | \* - Enable/disable network Fixed Root Setting by API esp\_mesh\_fix\_root(). | |
|  | \* - Network Fixed Root Setting also changes with the "flag" value in parent networking IE. | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_is\_root\_fixed(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set a specified parent for the device | |
|  | \* | |
|  | \* @attention This API can be called at any time after mesh is configured. | |
|  | \* | |
|  | \* @param[in] parent parent configuration, the SSID and the channel of the parent are mandatory. | |
|  | \* - If the BSSID is set, make sure that the SSID and BSSID represent the same parent, | |
|  | \* otherwise the device will never find this specified parent. | |
|  | \* @param[in] parent\_mesh\_id parent mesh ID, | |
|  | \* - If this value is not set, the original mesh ID is used. | |
|  | \* @param[in] my\_type mesh type | |
|  | \* - MESH\_STA is not supported. | |
|  | \* - If the parent set for the device is the same as the router in the network configuration, | |
|  | \* then my\_type shall set MESH\_ROOT and my\_layer shall set MESH\_ROOT\_LAYER. | |
|  | \* @param[in] my\_layer mesh layer | |
|  | \* - my\_layer of the device may change after joining the network. | |
|  | \* - If my\_type is set MESH\_NODE, my\_layer shall be greater than MESH\_ROOT\_LAYER. | |
|  | \* - If my\_type is set MESH\_LEAF, the device becomes a standalone Wi-Fi station and no longer | |
|  | \* has the ability to extend the network. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_CONFIG | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_parent(const wifi\_config\_t \*parent, const mesh\_addr\_t \*parent\_mesh\_id, mesh\_type\_t my\_type, int my\_layer); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh networking IE length of one AP | |
|  | \* | |
|  | \* @param[out] len mesh networking IE length | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_WIFI\_NOT\_INIT | |
|  | \* - ESP\_ERR\_WIFI\_ARG | |
|  | \* - ESP\_ERR\_WIFI\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_scan\_get\_ap\_ie\_len(int \*len); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get AP record | |
|  | \* | |
|  | \* @attention Different from esp\_wifi\_scan\_get\_ap\_records(), this API only gets one of APs scanned each time. | |
|  | \* See "manual\_networking" example. | |
|  | \* | |
|  | \* @param[out] ap\_record pointer to one AP record | |
|  | \* @param[out] buffer pointer to the mesh networking IE of this AP | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_WIFI\_NOT\_INIT | |
|  | \* - ESP\_ERR\_WIFI\_ARG | |
|  | \* - ESP\_ERR\_WIFI\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_scan\_get\_ap\_record(wifi\_ap\_record\_t \*ap\_record, void \*buffer); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Flush upstream packets pending in to\_parent queue and to\_parent\_p2p queue | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_flush\_upstream\_packets(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the number of nodes in the subnet of a specific child | |
|  | \* | |
|  | \* @param[in] child\_mac an associated child address of this device | |
|  | \* @param[out] nodes\_num pointer to the number of nodes in the subnet of a specific child | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_NOT\_START | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_subnet\_nodes\_num(const mesh\_addr\_t \*child\_mac, int \*nodes\_num); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get nodes in the subnet of a specific child | |
|  | \* | |
|  | \* @param[in] child\_mac an associated child address of this device | |
|  | \* @param[out] nodes pointer to nodes in the subnet of a specific child | |
|  | \* @param[in] nodes\_num the number of nodes in the subnet of a specific child | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_MESH\_NOT\_START | |
|  | \* - ESP\_ERR\_MESH\_ARGUMENT | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_subnet\_nodes\_list(const mesh\_addr\_t \*child\_mac, mesh\_addr\_t \*nodes, int nodes\_num); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Disconnect from current parent | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_disconnect(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Connect to current parent | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_connect(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Flush scan result | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_flush\_scan\_result(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Cause the root device to add Channel Switch Announcement Element (CSA IE) to beacon | |
|  | \* - Set the new channel | |
|  | \* - Set how many beacons with CSA IE will be sent before changing a new channel | |
|  | \* - Enable the channel switch function | |
|  | \* | |
|  | \* @attention This API is only called by the root. | |
|  | \* | |
|  | \* @param[in] new\_bssid the new router BSSID if the router changes | |
|  | \* @param[in] csa\_newchan the new channel number to which the whole network is moving | |
|  | \* @param[in] csa\_count channel switch period(beacon count), unit is based on beacon interval of its softAP, the default value is 15. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_switch\_channel(const uint8\_t \*new\_bssid, int csa\_newchan, int csa\_count); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the router BSSID | |
|  | \* | |
|  | \* @param[out] router\_bssid pointer to the router BSSID | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_WIFI\_NOT\_INIT | |
|  | \* - ESP\_ERR\_WIFI\_ARG | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_router\_bssid(uint8\_t \*router\_bssid); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the TSF time | |
|  | \* | |
|  | \* @return the TSF time | |
|  | \*/ | |
|  | int64\_t esp\_mesh\_get\_tsf\_time(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set mesh topology. The default value is MESH\_TOPO\_TREE | |
|  | \* - MESH\_TOPO\_CHAIN supports up to 1000 layers | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @param[in] topo MESH\_TOPO\_TREE or MESH\_TOPO\_CHAIN | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_MESH\_ERR\_ARGUMENT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_topology(esp\_mesh\_topology\_t topo); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get mesh topology | |
|  | \* | |
|  | \* @return MESH\_TOPO\_TREE or MESH\_TOPO\_CHAIN | |
|  | \*/ | |
|  | esp\_mesh\_topology\_t esp\_mesh\_get\_topology(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Enable mesh Power Save function | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_WIFI\_NOT\_INIT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_enable\_ps(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Disable mesh Power Save function | |
|  | \* | |
|  | \* @attention This API shall be called before mesh is started. | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_ERR\_WIFI\_NOT\_INIT | |
|  | \* - ESP\_ERR\_MESH\_NOT\_ALLOWED | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_disable\_ps(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Check whether the mesh Power Save function is enabled | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_is\_ps\_enabled(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Check whether the device is in active state | |
|  | \* - If the device is not in active state, it will neither transmit nor receive frames. | |
|  | \* | |
|  | \* @return true/false | |
|  | \*/ | |
|  | bool esp\_mesh\_is\_device\_active(void); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set the device duty cycle and type | |
|  | \* - The range of dev\_duty values is 1 to 100. The default value is 12. | |
|  | \* - dev\_duty = 100, the PS will be stopped. | |
|  | \* - dev\_duty is better to not less than 5. | |
|  | \* - dev\_duty\_type could be MESH\_PS\_DEVICE\_DUTY\_REQUEST or MESH\_PS\_DEVICE\_DUTY\_DEMAND. | |
|  | \* - If dev\_duty\_type is set to MESH\_PS\_DEVICE\_DUTY\_REQUEST, the device will use a nwk\_duty provided by the network. | |
|  | \* - If dev\_duty\_type is set to MESH\_PS\_DEVICE\_DUTY\_DEMAND, the device will use the specified dev\_duty. | |
|  | \* | |
|  | \* @attention This API can be called at any time after mesh is started. | |
|  | \* | |
|  | \* @param[in] dev\_duty device duty cycle | |
|  | \* @param[in] dev\_duty\_type device PS duty cycle type, not accept MESH\_PS\_NETWORK\_DUTY\_MASTER | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_active\_duty\_cycle(int dev\_duty, int dev\_duty\_type); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get device duty cycle and type | |
|  | \* | |
|  | \* @param[out] dev\_duty device duty cycle | |
|  | \* @param[out] dev\_duty\_type device PS duty cycle type | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_active\_duty\_cycle(int\* dev\_duty, int\* dev\_duty\_type); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Set the network duty cycle, duration and rule | |
|  | \* - The range of nwk\_duty values is 1 to 100. The default value is 12. | |
|  | \* - nwk\_duty is the network duty cycle the entire network or the up-link path will use. A device that successfully | |
|  | \* sets the nwk\_duty is known as a NWK-DUTY-MASTER. | |
|  | \* - duration\_mins specifies how long the specified nwk\_duty will be used. Once duration\_mins expires, the root will take | |
|  | \* over as the NWK-DUTY-MASTER. If an existing NWK-DUTY-MASTER leaves the network, the root will take over as the | |
|  | \* NWK-DUTY-MASTER again. | |
|  | \* - duration\_mins = (-1) represents nwk\_duty will be used until a new NWK-DUTY-MASTER with a different nwk\_duty appears. | |
|  | \* - Only the root can set duration\_mins to (-1). | |
|  | \* - applied\_rule could be MESH\_PS\_NETWORK\_DUTY\_APPLIED\_ENTIRE or MESH\_PS\_NETWORK\_DUTY\_APPLIED\_UPLINK. | |
|  | \* - If applied\_rule is set to MESH\_PS\_NETWORK\_DUTY\_APPLIED\_ENTIRE, the nwk\_duty will be used by the entire network. | |
|  | \* - If applied\_rule is set to MESH\_PS\_NETWORK\_DUTY\_APPLIED\_UPLINK, the nwk\_duty will only be used by the up-link path nodes. | |
|  | \* - The root does not accept MESH\_PS\_NETWORK\_DUTY\_APPLIED\_UPLINK. | |
|  | \* - A nwk\_duty with duration\_mins(-1) set by the root is the default network duty cycle used by the entire network. | |
|  | \* | |
|  | \* @attention This API can be called at any time after mesh is started. | |
|  | \* - In self-organized network, if this API is called before mesh is started in all devices, (1)nwk\_duty shall be set to the | |
|  | \* same value for all devices; (2)duration\_mins shall be set to (-1); (3)applied\_rule shall be set to | |
|  | \* MESH\_PS\_NETWORK\_DUTY\_APPLIED\_ENTIRE; after the voted root appears, the root will become the NWK-DUTY-MASTER and broadcast | |
|  | \* the nwk\_duty and its identity of NWK-DUTY-MASTER. | |
|  | \* - If the root is specified (FIXED-ROOT), call this API in the root to provide a default nwk\_duty for the entire network. | |
|  | \* - After joins the network, any device can call this API to change the nwk\_duty, duration\_mins or applied\_rule. | |
|  | \* | |
|  | \* @param[in] nwk\_duty network duty cycle | |
|  | \* @param[in] duration\_mins duration (unit: minutes) | |
|  | \* @param[in] applied\_rule MESH\_PS\_NETWORK\_DUTY\_APPLIED\_ENTIRE or MESH\_PS\_NETWORK\_DUTY\_APPLIED\_UPLINK | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \* - ESP\_FAIL | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_set\_network\_duty\_cycle(int nwk\_duty, int duration\_mins, int applied\_rule); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the network duty cycle, duration, type and rule | |
|  | \* | |
|  | \* @param[out] nwk\_duty current network duty cycle | |
|  | \* @param[out] duration\_mins the duration of current nwk\_duty | |
|  | \* @param[out] dev\_duty\_type if it includes MESH\_PS\_DEVICE\_DUTY\_MASTER, this device is the current NWK-DUTY-MASTER. | |
|  | \* @param[out] applied\_rule MESH\_PS\_NETWORK\_DUTY\_APPLIED\_ENTIRE or MESH\_PS\_NETWORK\_DUTY\_APPLIED\_UPLINK | |
|  | \* | |
|  | \* @return | |
|  | \* - ESP\_OK | |
|  | \*/ | |
|  | esp\_err\_t esp\_mesh\_get\_network\_duty\_cycle(int\* nwk\_duty, int\* duration\_mins, int\* dev\_duty\_type, int\* applied\_rule); | |
|  |  | |
|  | /\*\* | |
|  | \* @brief Get the running active duty cycle | |
|  | \* - The running active duty cycle of the root is 100. | |
|  | \* - If duty type is set to MESH\_PS\_DEVICE\_DUTY\_REQUEST, the running active duty cycle is nwk\_duty provided by the network. | |
|  | \* - If duty type is set to MESH\_PS\_DEVICE\_DUTY\_DEMAND, the running active duty cycle is dev\_duty specified by the users. | |
|  | \* - In a mesh network, devices are typically working with a certain duty-cycle (transmitting, receiving and sleep) to | |
|  | \* reduce the power consumption. The running active duty cycle decides the amount of awake time within a beacon interval. | |
|  | \* At each start of beacon interval, all devices wake up, broadcast beacons, and transmit packets if they do have pending | |
|  | \* packets for their parents or for their children. Note that Low-duty-cycle means devices may not be active in most of | |
|  | \* the time, the latency of data transmission might be greater. | |
|  | \* | |
|  | \* @return the running active duty cycle | |
|  | \*/ | |
|  | int esp\_mesh\_get\_running\_active\_duty\_cycle(void); | |
|  |  | |
|  | #ifdef \_\_cplusplus | |
|  | } | |
|  | #endif | |
|  | #endif /\* \_\_ESP\_MESH\_H\_\_ \*/ | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |