This application note describes the Wi-Fi Unassociated mode transmission procedure. The WCM APIs used in the accompanying sample code describes scan parameters which can be set, power management configuration and transmission of unassociated frames to the AP.

# Topology

**Access Point**

**Talaria TWO**

Probe Request/Response

Figure 1: Topology

# Wi-Fi Unassociated Mode Functionalities

The Wi-Fi Unassociated mode contains functions for transmitting data over the probe request frames without associating with an AP.

Following functionalities are achieved through these APIs:

* 1. Creating and destroying Wi-Fi interface
  2. Initializing the default scan parameters
  3. Disabling beacon reception
  4. Setting power saving options
  5. Wi-Fi probe transmission

Accompanying sample code provides more details on achieving some of the listed functionalities.

# Wi-Fi Unassociated Mode APIs

## wifi\_if\_create()

Creates a Wi-Fi interface with the specified operation mode.

|  |
| --- |
| struct wifi\_net \*  wifi\_if\_create(enum wifi\_opmode mode, const uint8\_t \*hwaddr) |

1. mode: Interface operation mode as specified by wifi\_opmode.

|  |
| --- |
| enum wifi\_opmode  {  WIFI\_MODE\_STA,  WIFI\_MODE\_P2P,  WIFI\_MODE\_HOSTAP,  WIFI\_MODE\_MONITOR,  WIFI\_MODE\_SCAN  } |

1. hwaddr: Specifies the hardware MAC address to be used for this interface. This parameter is NULL (or points to an invalid station MAC address) and a random address will be generated for the interface.

Returns pointer to the newly created interface in the form of wifi\_net.

## wifi\_if\_destroy()

Destroys a Wi-Fi interface.

|  |
| --- |
| void  wifi\_if\_destroy(struct wifi\_net \*net) |

1. net: Pointer to a struct wifi\_net created by wifi\_if\_create() API.

Returns void.

## wifi\_init\_scan\_default()

Initializes the scan parameters with default values.

|  |
| --- |
| void wifi\_init\_scan\_default(struct wifi\_scan\_param \*param) |

1. param: pointer to the Wi-Fi scan param structure.

Returns void.

## wifi\_ssid\_from\_bytes()

Initializes SSID from a byte string.

|  |
| --- |
| int wifi\_ssid\_from\_bytes(struct wifi\_ssid \*ssid, const void \*ssid\_bytes, size\_t ssid\_length) |

This function initializes a SSID structure from a byte string. SSID is a sequence of bytes, not always representable as a zero terminated string. This function will always generate a zero-terminated result but treating it as such will cause problems with SSID's having embedded zero bytes (although this is uncommon). If the input byte sequence is too long (more than IEEE80211\_NWID\_LEN bytes), this function will truncate the value, and return a negative result.

1. ssid: ssid structure to initialize.
2. ssid\_bytes: pointer to SSID data.
3. ssid\_length: length of ssid\_bytes.

Returns zero on success -EINVAL ssid\_length exceeds IEEE80211\_NWID\_LEN.

## wifi\_set\_pm()

Configures Wi-Fi power-save parameters.

|  |
| --- |
| int wifi\_pm\_flags  wifi\_set\_pm(struct wifi\_net \*net, uint32\_t listen\_interval, uint32\_t traffic\_tmo, uint32\_t pm\_flags) |

Initiate a network scan procedure on the specified WCM interface.

1. struct wifi\_net \*net: Pointer to wifi\_net. This struct is representing a Wi-Fi network (or interface).
2. listen\_interval: Beacon listen interval.
3. traffic\_tmo: Traffic timeout in milliseconds. The traffic timeout parameter specifies the amount of time (in ms) for which the device should stay active.
4. pm\_flags: Using this parameter, following flags can be enabled:
   * WIFI\_PM\_PS\_POLL

Send ps poll if a beacon was missed.

* + WIFI\_PM\_DYN\_LISTEN\_INT

Dynamic listen interval. Listen to each beacon if there has been traffic recently.

* + WIFI\_PM\_STA\_RX\_NAP

Turn off receiver for uninteresting frames for station.

* + WIFI\_PM\_STA\_ONLY\_BROADCAST

Do not receive multicast frames that are not broadcast.

* + WIFI\_PM\_TX\_PS

Send outgoing frames without leaving Wi-Fi power save.

* + WIFI\_PM\_MCAST\_DONT\_CARE

Ignore the multicast flag in beacons. Incoming broadcast ARPs or other important broadcast/multicast traffic may be missed.

## wifi\_scan()

Initiates a network scan procedure to send the unassociated frame (probe request) on the specified Wi-Fi connection interface.

|  |
| --- |
| int  wifi\_scan(struct wifi\_net \*net, const struct wifi\_scan\_param \*param) |

Initiate a network scan procedure.

1. struct wifi\_net \*net: Pointer to wifi\_net. This struct is representing a Wi-Fi network (or interface).
2. const struct wifi\_scan\_param \*param: Pointer to wifi\_scan\_param which contains multiple parameters that tunes the behavior of the scan operation.

Returns zero on success, error value otherwise.

Details about struct wifi\_scan\_param and working of this API are available in section 7.1.

# States and Events of Unassociated Mode

## Unassociated Tx Procedure

When the application thread calls the API wifi\_scan(),the message to the Wi-Fi stack is sent to start the scan (send probe request) with programmed scan parameters.

struct wifi\_scan\_param contains multiple parameters that are used to fine tune the behaviour of the scan operation.

|  |
| --- |
| /\*\* Parametes for WiFi scan operaion \*/  struct wifi\_scan\_param {  /\*\* List of channels to scan \*/  uint8\_t channel\_mask[8];  /\*\* Destination address and BSSID for probe requests \*/  uint8\_t bssid[IEEE80211\_ADDR\_LEN];  /\*\* Rate to use for sending probe requests \*/  rate\_t txrate;  /\*\* Scan for specific SSID (set to empty string for ANY). \*/  struct wifi\_ssid ssid;  /\*\* The amount of time (in microseconds) to stay on the channel after  transmitting the probe request and listening for responses \*/  uint32\_t dwelltime;  /\*\* Idle time between each channel (giving other parties access to the  media) \*/  uint32\_t waittime;  /\*\* Length of optional extra information elements included in the probe  request frames \*/  size\_t ie\_len;  /\*\* Buffer with information elements that will be inserted in each probe  request frame. \*/  uint8\_t ie\_list[0];  }; |

wifi\_init\_scan\_default()initiates wifi\_scan\_param structure which is used when scan parameter is passed as NULL which has the following default values:

1. Sending an unassociated frame for a specific SSID can be performed by initializing the SSID field. By default, it is empty and set for scanning any SSID.
2. Sending an unassociated frame for any specific channel can be done by initializing the channel mask. By default, it is set to 0xff and is set for all the channels.
3. By default, dwell-time is set to 25µs, wait time to 0 and no other additional information elements are included.

struct wifi\_netinfo holds the results and information about scanned networks. This parameter structure is not valid for the unassociated mode.

# Code Walkthrough

## Unassociated mode Tx Example

### Overview

The sample code in the path *example\unassoc\src\main.c* showcases the unassociated mode transmission.

### Sample Code Walkthrough

This example code initializes the scan parameters as default. It also gets the boot arguments and stores it in scan parameters.

|  |
| --- |
| // Allocate memory for a @ref wifi\_scan\_param specifying the length  of the ie\_list  struct wifi\_scan\_param \*param = os\_alloc(sizeof (struct  wifi\_scan\_param) + ie\_len);  // Initiate the default scan param values  wifi\_init\_scan\_default(param);  // Update the ie\_list  if(ie\_len > 0) {  param->ie\_len = ie\_len;  memcpy(param->ie\_list, ie\_list\_output, ie\_len);  }  // Number of probes to send can be specified by the boot arg  num\_probes, 0 for infinity  uint32\_t num\_probes = os\_get\_boot\_arg\_int("num\_probes", 0);  interval = os\_get\_boot\_arg\_int("interval\_ms", 1000);  // Specify a target bssid, defaults to ANY  const char \*tmp;  tmp = os\_get\_boot\_arg\_str("addr");  if (tmp)  parse\_macaddr(tmp, param->bssid);  // Specify the rate to send probes, generally 11b->11g (RATE\_1 to  RATE\_6)  param->txrate = os\_get\_boot\_arg\_int("rate", RATE\_6);  // Specify a target SSID, defaults to ANY  tmp = os\_get\_boot\_arg\_str("scan\_ssid");  if (tmp)  wifi\_ssid\_from\_bytes(&param->ssid, tmp, strlen(tmp));  // Enable device suspend (deep sleep) via boot argument  bool suspend = os\_get\_boot\_arg\_int("suspend", 0);  if (suspend == 1) {  os\_printf("deep sleep enabled.\n");  os\_suspend\_enable();  } else {  os\_printf("deep sleep disabled.\n");  } |

The following code creates the Wi-Fi interface in scan mode. It also sets the power management feature and sends unassociated frames in a while loop.

|  |
| --- |
| struct wifi\_net \*net = wifi\_if\_create(WIFI\_MODE\_SCAN, NULL);  // Set the shortest traffic\_tmo and hope to go to suspend early  wifi\_set\_pm(net, 0, 1, 0);  uint32\_t num\_probe\_sent = 0;  callout\_init(&probe\_callout, send\_unassoc\_probe);  os\_sem\_init(&send\_probe\_sem, 0);  for(;;) {  os\_printf("[%u] Sending probe\n", num\_probe\_sent);  int result = wifi\_probe\_send(net, param);  if(result == 0)  num\_probe\_sent++;  else  os\_printf("[%u] Failed to send probe due to error %d\n",  num\_probe\_sent, -result);  start\_timeout();  os\_sem\_wait(&send\_probe\_sem);  if((num\_probes != 0) &&  (num\_probes == num\_probe\_sent)) {  os\_printf("Probe sending complete.\n");  break;  } }  os\_printf("Sent %d out of %d probes.\n", num\_probe\_sent,  num\_probes);  wifi\_if\_destroy(net);  os\_free(param);  while(1) {  os\_sem\_wait(&send\_probe\_sem);  } } |

### Running the Example

Program unassoc.elf *(freertos\_sdk\_x.y\examples\unassoc\bin)* using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the unassoc.elf by clicking on Select ELF File.
   3. Boot Arguments: Pass the following boot arguments:

|  |
| --- |
| scan\_ssid=InnoPhase\_AE\_AP,ie\_list=0x11:0x02:0x33:0x44:0x12:0x04:0x77:0x88:0x99:0xaa:0x13:0x07:0xa0:0xa1:0xa2:0xa3:0xa4:0xa5:0xa6,rate=0,num\_probes=3,suspend=1,interval\_ms=1500,verbose=1 |

**Note**: For one, two and three custom ies, refer the README file in the following location: *freertos\_sdk\_x.y\examples\unassoc\doc*.

* 1. Programming: Click on Prog Flash.

### Expected Output

unassoc.elf provides the following console output in different scenarios when programmed to Talaria TWO.

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWWAE  Build $Id: git-df9b9ef $  Flash detected. flash.hw.uuid: 39483937-3207-00b0-0064-ffffffffffff  Bootargs: scan\_ssid=Lucy ie\_list=0x11:0x02:0x33:0x44:0x12:0x04:0x77:0x88:0x99:0xaa:0x13:0x07:0xa0:0xa1:0xa2:0xa3:0xa4:0xa5:0xa6 rate=0 num\_probes=3 suspend=1 interval\_ms=1500 verbose=1  $App:git-6600fea  SDK Ver: FREERTOS\_SDK\_1.0  Un-Assoc Tx Example  ie\_len=94  ie\_len=19  Hexdump of IES, len=19  11 02 33 44 12 04 77 88 99 AA 13 07 A0 A1 A2 A3 | ..3D..w.........  A4 A5 A6 | ...  Hexdump of ie tag:11, len=2  33 44 | 3D  Hexdump of ie tag:12, len=4  77 88 99 AA | w...  Hexdump of ie tag:13, len=7  A0 A1 A2 A3 A4 A5 A6 | .......  deep sleep enabled.  [1] Sending probe.  [1] Probe completed.  [2] Sending probe.  [2] Probe completed.  [3] Sending probe.  [3] Probe completed.  Done sending probes!  Done |

Wireshark output in Figure 2 shows the 3 probe requests sent to configured SSID for interval of 1500ms.

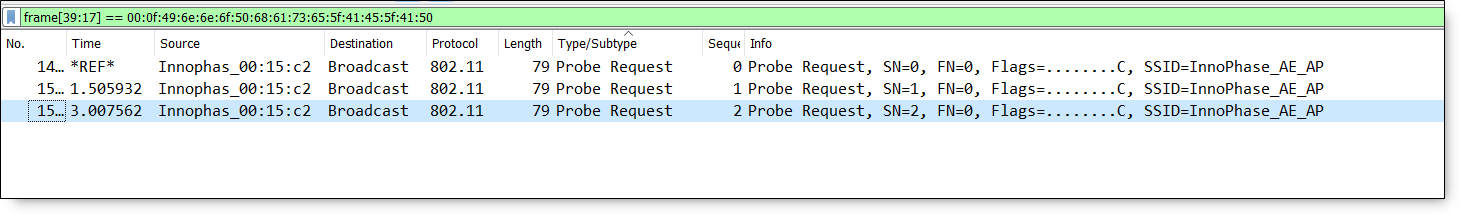


Figure : Probe requests sent to configured SSID – Wireshark output