Wi-Fi Connection Manager Multi-AP

# Introduction

This application note demonstrates the wcm\_multi\_ap example which performs the connection and disconnections with multiple access points based on the priority of the access point. The priority of the access point to connect to is set by the application based on the order that the SSID and passphrase details of the access point are provided through the boot arguments and also on the Access Point's entry in Talaria TWO’s scan results.

# Wi-Fi Connection with Multi-AP Functionalities

## Overview of wcm\_multi\_ap

wcm\_multi\_ap takes user entry for APs (SSID and passphrase) the priority of which is decided depending on the order of the provided AP details, in the form of boot arguments. It utilizes the functionalities provided by WCM on the Wi-Fi connection management and provides the following features:

1. Configuration and management of user-entered, multiple APs.
2. Priority-oriented list for AP connection choice.

## Overview of WCM

The Wi-Fi connection manager (WCM) contains functions for establishing and tearing down associations to Wi-Fi access points. It initiates a four-way handshake for encrypted connections and starts the DHCP client from LWIP for getting an allocation of the IP address.

Following functionalities are achieved through these APIs:

* 1. Creating and destroying interfaces.
  2. Scanning with different scan parameters and their indication structure.
  3. Connecting and disconnecting.
  4. Adding network to connect asynchronously and removing network.
  5. Querying connection information.

Accompanying sample code provides more details on how to achieve some of the listed functionalities.

# WCM APIs used in wcm\_multi\_ap

1. wcm\_auto\_connect() - Starts or stops auto connect.

A network must first be added with reference to wcm\_add\_network.

1. wcm\_delete\_network\_profile () - Deletes the currently active network profile that was previously added .
2. wifi\_init\_scan\_default() - Gets the default parameters for scan operation.

The wifi\_scan\_param will be updated with the default values for the scan operation. This function is used to get the default values and alter the parameters which need to have values other than the default ones. Finally, provide this parameter when calling wcm\_scan.

1. wifi\_netinfo\_get\_ssid() - Gets SSID from netinfo.
2. wifi\_netinfo\_get\_chan() - Gets channel from netinfo
3. wcm\_free\_scanresult() - Frees the memory allocated by wcm\_scan
4. os\_free() - Frees allocated memory.

Returns allocated memory to the heap. If the memory has more than one reference, the count is simply dropped by one.

1. os\_msg\_release() - Frees an allocated message.

Frees a message previously allocated using os\_msg\_alloc().

1. wcm\_scan() - Scans for Wi-Fi networks.

Initiates a network scan procedure on the specified Wi-Fi Connection Manager interface.

1. Wcm\_create() - Creates a Wi-Fi network interface.

Only one wcm\_handle instance is supported at the time of writing

1. wcm\_add\_network\_profile() - Adds a network profile to wcm.

# Code Walkthrough WCM\_MULTI\_AP

* 1. Application Flow

When the app starts, the following sequence of initializations happens in wcm\_multi\_ap:

1. User provides the boot arguments consisting of a list of AP entries in the form of SSID and the passphrase for each AP. The order of the AP as they are entered indicates the priority of the connection.
2. App reads the boot arguments and saves them into the buffer - AP manager.
3. App starts a thread with the following functionality:
   1. Create a Wi-Fi network interface.
   2. Pass to the WCM, the list of user-entered SSID’s.
   3. Register the call-back function with the WCM notification.
   4. Start a connection with the most recently used or most recently added AP in the list.
4. Enters the loop with the detection for reconnection.

## Sample code walkthrough

1. User Data Struct: AP Manager

A user-defined data struct is created to store the data of the multi APs:

|  |
| --- |
| #define AP\_CNT 5  #define MAX\_RETRY 3  #define MRU\_RETRY 3  #define MAX\_NETS 16  #define AP\_DISCONNECTED 0  #define AP\_CONNECTED 1  #define AP\_CONNECTING 2  struct ap\_param  {  char ssid[32];  uint8\_t bssid[6];  char passphrase[64];  };  struct ap\_manager  {  struct ap\_param ap\_param[AP\_CNT];  int priority[AP\_CNT];  int mru; //index of MRU AP  int rescan\_interval;  int cnt;  int index;  int mru\_retries;  int connect\_retries;  } ap\_manager; |

1. How the AP List is Created and Maintained

When the AP manager is populated with a list of SSID from the boot arguments, the AP manager starts to use the AP’s order as the initial priority order (in ascending order) of selection for Wi-Fi connection with the AP of the highest number (number 0) as the starting connection.

* 1. AP Addition and Initialization of the List

The priority variable is initialized with the order of the AP in the function wcma\_add\_network().

|  |
| --- |
| int wcma\_add\_network(struct wcm\_handle \*handle, const char ssid[32],const char bssid[17], const char passphrase[64], struct ap\_manager \*manager)  .  .  for(int i = 0; i < AP\_CNT; i++)  {  if(manager->ap\_param[i].ssid[0] == 0)  {  os\_printf("adding %s to list\n", ssid);  memcpy(manager->ap\_param[i].ssid, ssid, ssid\_len);  if(bssid != NULL)  {  os\_printf("BSSID: %s\n", bssid);  parse\_macaddr(bssid, manager->ap\_param[i].bssid);  }  else  {  memset(manager->ap\_param[i].bssid, 0, 6);  } |

1. Update of the Priority Numbers of APs

The priority order of the AP determines which AP from the AP list will be chosen for the next connection to be made. This list is managed automatically by the app.

The order of APs can be changed from time to time. For instance, based on the change of status of the connection with the current AP, if the connection gets dropped , the AP manager will attempt to connect to the AP that is next on the priority list. The priority list of APs in the AP manager will be updated accordingly in the manager based on the index variable value.

|  |
| --- |
| /\* perform scan \*/  wcma\_scan\_retry(handle, 3, manager);  /\* get AP with highest priority \*/  int highest\_priority = manager->cnt+1;  manager->index = -1;  for(int i = 0; i < manager->cnt; i++)  {  if((manager->priority[i] >=0) && (manager->priority[i] < highest\_priority))  {  highest\_priority = manager->priority[i];  manager->index = i;  }  } |

1. Last Used Entry (LRU)

AP manager uses the ap\_mamager,mru variable as the index of LRU AP.

|  |
| --- |
| static void my\_wcm\_notify\_cb(void \*ctx, struct os\_msg \*msg)  {  switch(msg->msg\_type)  .  .  {  case(WCM\_NOTIFY\_MSG\_ADDRESS):  os\_printf("wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS\n");  // set most recently used AP and updated priority in list  ap\_manager.mru = ap\_manager.index;  ap\_manager.priority[ap\_manager.index] = 0;  ap\_manager.connect\_retries = 0;  ap\_manager.mru\_retries = MRU\_RETRY;  connection\_status = AP\_CONNECTED;  last\_connect\_time = os\_systime();  break; |

It is set when the call-back from WCM is received indicating that the connection to the AP has been made. It is used to set the AP entry as the AP to connect to.

1. Selection of AP to Connect to

The selection for an AP to connect to is decided based on the index and mru variable values in the AP manager:

* 1. ap\_manager: index contains the index to the AP that the next connection will be made to.
  2. ap\_manager: mru contains the index to the entry of used last time, or the retry attempts.

|  |
| --- |
| void wcma\_auto\_connect(struct wcm\_handle \*handle, struct ap\_manager \*manager)  {  reconnect\_next\_ap = false;  last\_disconnect\_time = 0xFFFFFFFF;  if(manager->cnt > 0)  {  if((manager->mru >= 0) && manager->mru\_retries >= 0)  {  os\_printf("connecting to most recently used AP, SSID: %s PASS: %s\n", manager->ap\_param[manager->mru].ssid, manager->ap\_param[manager->mru].passphrase);  if(manager->ap\_param[manager->mru].passphrase[0] == 0)  {  rval = network\_profile\_new\_from\_ssid\_bssid\_pw(&profile, manager->ap\_param[manager->mru].ssid, NULL, NULL); |

wcm\_add\_network\_profile()API is used to add the network.

|  |
| --- |
| if (rval < 0) {  pr\_err("could not create network profile %d\n", rval);  return 0;  }  add\_ntwk = wcm\_add\_network\_profile(handle, profile);  }  else if(manager->ap\_param[manager->mru].bssid[0] != 0)  {  rval = network\_profile\_new\_from\_ssid\_bssid\_pw(&profile,manager->ap\_param[manager->mru].ssid, (uint8\_t \*)manager->ap\_param[manager->mru].bssid, manager->ap\_param[manager->mru].passphrase); |

# Running the Application

Program wcma.elf (sdk\_x.y\examples\wcm\_multi\_ap\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 wcma.elf by clicking on Select ELF File.
   3. Boot arguments: Pass the following boot arguments:

|  |
| --- |
| hssid1=Asus\_Qos\_2G,passphrase1=99999999,ssid2=manasvi,passphrase2=manasvi123,hssid3=TP-Link\_2G,passphrase3=12345678 |

* 1. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y/pc\_tools/Download\_Tool/doc*).

**Note**: x and y refer to the SDK release version. For example: sdk\_2.4/doc.

* 1. Expected Output

wcma.elf is created when compiling this code and gives the following console output when programmed to Talaria TWO.

Initially, the Talaria TWO app connects with the AP mentioned in “ssid1”. If the AP with “ssid1” goes down, then the Talaria TWO app retries to reconnect with the same, after which it connects with the next in the list “ssid2”.

If AP is in hidden network, SSID should be mentioned as “hssid1” and passphrase as “hpassphrase1”.

|  |
| --- |
| UART:NWWWWWAEBuild $Id: git-b664be2af $  app=gordon  flash: Gordon ready!  Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-b664be2af $  hssid1=Asus\_Qos\_2G passphrase1=99999999 ssid2=manasvi passphrase2=manasvi123 hssid3=TP-Link\_2G passphrase3=12345678  $App:git-6e423223  SDK Ver: sdk\_2.4  Wifi Multi AP Demo App  adding Asus\_Qos\_2G(\*) to list  passphrase is NULL!!!  adding manasvi to list  adding TP-Link\_2G(\*) to list  passphrase is NULL!!!  AP 0, SSID: Asus\_Qos\_2G Passphrase:  AP 1, SSID: manasvi Passphrase: manasvi123  AP 2, SSID: TP-Link\_2G Passphrase:  addr e0:69:3a:00:00:dc  Found 3 nets:  3c:84:6a:f4:4e:b0 on channel 5 @ -31 'manasvi'  b4:43:26:4b:4c:d4 on channel 10 @ -64 'Ananth Krishna'  5c:f9:fd:6b:e9:09 on channel 5 @ -79 'manasvi'  Found 2 nets:  3c:84:6a:f4:4e:b0 on channel 5 @ -30 'manasvi'  b4:43:26:4b:4c:d4 on channel 10 @ -61 'Ananth Krishna'  Found 3 nets:  3c:84:6a:f4:4e:b0 on channel 5 @ -32 'manasvi'  b4:43:26:4b:4c:d4 on channel 10 @ -62 'Ananth Krishna'  a0:ab:1b:27:99:4c on channel 9 @ -67 'Vinoth\_room2.4g'  Hidden network: Asus\_Qos\_2G  Hidden network: TP-Link\_2G  Found 4 nets:  3c:84:6a:f4:4e:b0 on channel 5 @ -30 'manasvi'  b4:43:26:4b:4c:d4 on channel 10 @ -63 'Ananth Krishna' |