Using Wi-Fi

# Introduction

This application note describes the Wi-Fi Connection Manager APIs available in the SDK, call-back events, notifications and the data structures associated with them. The accompanying sample codes help achieve some of these functionalities using WCM APIs.

# Topology

Diagram

Description automatically generated

Figure 1: Wi-Fi Connection Manager – Topology

# Wi-Fi Connection Manager Functionalities

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 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 and removing network
  5. Querying connection information

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

# List of APIs

1. os\_sem\_post() - Unlocks the semaphore. Increments the value of a semaphore and wakes the first thread waiting for this semaphore.
2. os\_msg\_release()- Frees an allocated message. Frees a message which was previously allocated using os\_msg\_alloc().
3. os\_get\_boot\_arg\_str() - Returns the value of a boot argument as a null-terminated string.
4. os\_sem\_init()- Initializes a semaphore. This API must be called before calling any other semaphore related APIs. It is possible to statically initialize a semaphore with the OS\_SEM\_INITALIZER macro.
5. wcm\_create()- Creates a Wi-Fi network interface. Only one wcm\_handle instance is supported at the time of writing.
6. wifi\_connect\_to\_network()- Adds a Wi-Fi network .
7. os\_sem\_wait()- Locks a semaphore. If the value of the semaphore is greater than 0, it decrements the counter. If the value is 0, it puts the current thread to sleep until the value becomes positive. This function cannot be used in interrupt context.
8. os\_msg\_release()- Frees an allocated message. Free a message previously allocated using os\_msg\_alloc().
9. os\_create\_thread()- Creates a new thread.

This creates a new thread with priority specified in flags (via OS\_CRTHREAD\_PRIO macro). The thread is placed in the run queue, but there is no immediate reschedule. The thread continues to run until the entry point returns, at a point where the return value (a pointer) can be obtained with os\_join\_thread().

If the return value is of no consequence, OS\_CRTHREAD\_DETACHED can be passed in flags. This causes the OS to reap the thread. Returns pointer to struct os\_thread if successful, NULL otherwise.

1. os\_join\_thread()- Waits for a thread to terminate and destroy the thread.

Calling this function will suspend execution of the calling thread until the target thread exits. The memory used to hold the threads stack and control block are freed.

1. os\_suspend\_enable()- Enables system suspend when idle.

Calling os\_suspend\_enable() will enable the suspend (deep sleep) for the system. If enabled, the system will go into suspended mode when there is nothing to do for the processor. Entering and leaving suspend mode takes some extra time, so enabling this will affect the real-time response of the system. When in suspended state, the system will still wakeup if an interrupt occurs, but the latency will be much larger compared to running with suspend mode disabled.

1. os\_get\_boot\_arg\_int()- Returns the value of a boot argument as an integer
2. wifi\_init\_scan\_default()- Gets the default parameters for the 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 whose values are different from the default. Finally, provide these parameters while calling the wcm\_scan() API.
3. wcm\_scan()- Scans for Wi-Fi networks. Returns the number of networks found during the scan operation. In case the scan could not be performed, an error code (negative value) is returned. Initiate a network scan procedure on the specified Wi-Fi Connection Manager interface.
4. wifi\_netinfo\_get\_ssid()-Gets the SSID information from netinfo.
5. wifi\_netinfo\_get\_chan()-Gets the channel information from netinfo.
6. wcm\_free\_scanresult() - Frees the memory allocated by wcm\_scan().
7. wcm\_set\_hostname() - Sets host name for DHCP client.
8. wcm\_get\_hostname() - Gets host name of DHCP client.

# States and Events of Wi-Fi Connection Manager

The architecture of the WCM, its calls and events with LWIP and Wi-Fi stacks are described in detail through the API flow.

WCM states are explained and notifications passed to application programmer are described.

## Initialization

When the API wcm\_create() is called, the following sequence of initializations happen on the WCM:

1. LWIP stack is initiated.
2. Wi-Fi interface and resources are created with the passed hwaddr.
3. Wi-Fi interface is created in WIFI\_MODE\_SCAN mode and a scan client is attached with scan\_notify() and scan\_done() call-backs to get scan results.
4. LWIP TCP/IP stack is started with the Wi-Fi interface in WIFI\_MODE\_STA mode.
5. wcm\_notify\_handler() is registered with the Wi-Fi stack for listening to the following link status notifications:

|  |
| --- |
| #define WIFI\_NOTIFY\_MSG\_LEAVE 100 /\* AP disconnected us \*/  #define WIFI\_NOTIFY\_MSG\_LOST 101 /\* Lost tracking of AP \*/  #define WIFI\_NOTIFY\_MSG\_RESTARTED 102 /\* Associated AP restarted \*/ |

1. wcm\_netif\_callback() is registered with LWIP stack for listening IP address change events.
2. A dedicated wcm\_thread is created which takes care of WCM’s state transitions at various stages of Wi-Fi connection procedure based on the previously described call-backs from the Wi-Fi stack, scanning interface and LWIP stack.

## States of WCM

All the possible states of the WCM are defined as follows:

|  |
| --- |
| enum c\_state {  C\_INITIAL,  C\_SCANNING,  C\_AUTHENTICATING,  C\_ASSOCIATING,  C\_WAIT\_4WAY,  C\_WAIT\_DHCP,  C\_CONNECTED,  C\_FAILED, }; |

The message pointer \*msg has a message type associated with it. msg->msg\_type can have a value from the enum wcm\_notify\_msg\_type, containing message types of WCM. This will notify message callbacks as shown:

|  |
| --- |
| enum wcm\_notify\_msg\_type {  /\*\* WiFi link is up \*/  WCM\_NOTIFY\_MSG\_LINK\_UP = 200,  /\*\* WiFi link is down \*/  WCM\_NOTIFY\_MSG\_LINK\_DOWN = 201,  /\*\* Address has changed.  WCM\_NOTIFY\_MSG\_ADDRESS = 202  }; |

If the notification WCM\_NOTIFY\_MSG\_ADDRESS occurs, then the message received is of the type wcm\_address\_event. The structure definition for wcm\_address\_event is as follows:

|  |
| --- |
| struct wcm\_address\_event {  /\*\* Message header \*/  struct os\_msg iev\_hdr;  /\*\* Typically AF\_INET or AF\_INET6 \*/  unsigned int iev\_af;  /\*\* The address of up to 16 bytes (IPv6) \*/  unsigned char iev\_address[0];  }; |

Address type and address data can be retrieved by typecasting the message to struct wcm\_address\_event.

# Source Code Walkthrough

## Wifi\_Connect

### Overview

The sample code in the path *apps\using\_wifi\src\wifi\_connect.c* showcases simple connecting to a network with the API wifi\_connect\_to\_network().

**Note:** Upon disconnection of station (Talaria TWO module) from the AP due to various reasons such as AP power off, module tries to reconnect by sending a probe request packet. Each failed connection attempt will increase the reconnect backoff time exponentially as 1, 2, 4, 8, 16, 32, 60 seconds.

After 60 seconds, module tries to reconnect indefinitely at every 60 seconds. Below sniffer capture shows Talaria TWO’s exponential reconnection method.

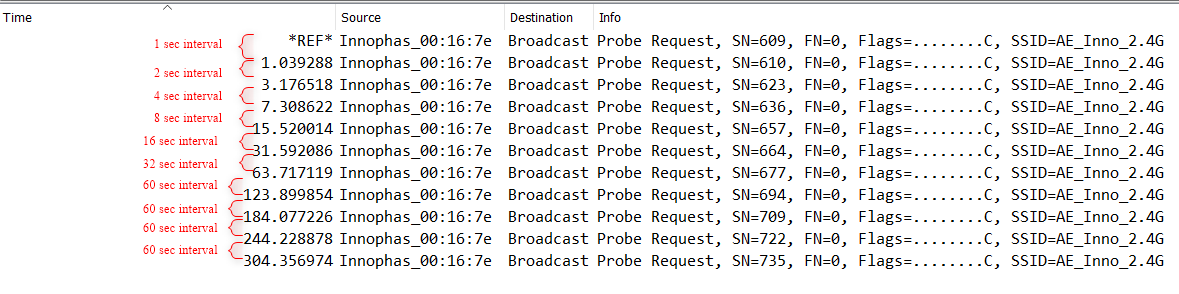


Figure 2: Sniffer Capture – Wi-Fi Reconnection

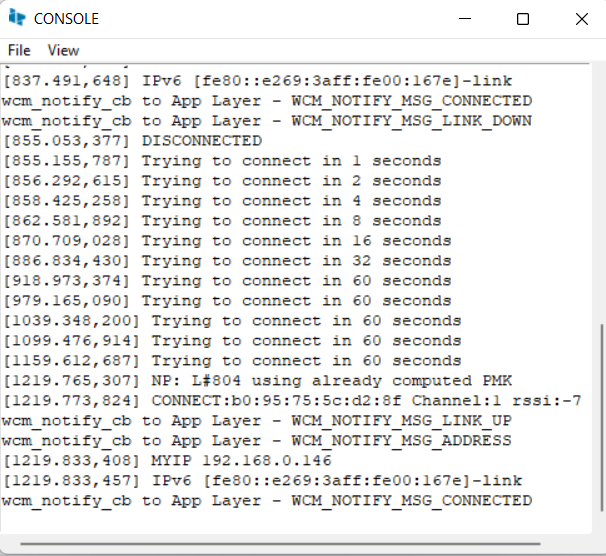


Figure 3: Talaria TWO Console Logs – Wi-Fi Reconnection

### Sample Code Walkthrough

To connect to a Wi-Fi network, wcm\_create()API from the Wi-Fi Connection Manager is used. Initially, the Wi-Fi network interface is created using wcm\_create().

|  |
| --- |
| h = wcm\_create(NULL); |

wcm\_set\_hostname() API is used to set the host name in DHCP client. It is used for identifying the client device (Talaria TWO) in the network/AP.

Note: Default host name of Talaria TWO is talaria2.

|  |
| --- |
| if(host\_name != NULL)  wcm\_set\_hostname(wcm\_handle, host\_name);  host\_name = wcm\_get\_hostname(wcm\_handle);  os\_printf("host name %s\n", host\_name); |

wifi\_connect\_to\_network()API, from components library, connects to the Wi-Fi network using the AP credentials provided.

|  |
| --- |
| rval = wifi\_connect\_to\_network(&h, WCM\_CONN\_WAIT\_INFINITE, &wcm\_connect\_success);  if(rval < 0) {  os\_printf("\nError: Unable to connect to network\n");  return 0;  } |

### Running the Application

Program wifi\_connect.elf(*sdk\_x.y\examples\using\_wifi\bin*)using the Download tool(*sdk\_x.y\pc\_tools\Download\_Tool*)provided with InnoPhase Talaria TWO SDK.

1. Launch the Download tool.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the wifi\_connect.elf by clicking on Select ELF File.
   3. AP Options: Pass the appropriate SSID and passphrase to connect to an Access Point.
   4. Boot Arguments: Add the host name as a boot argument.

|  |
| --- |
| host\_name=<host\_name> |

Note: Default host name is talaria2.

* 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.

### Expected Output

wifi\_connect.elf execution displays the following output on the console for different scenarios:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-d198c0771 $  np\_conf\_path=/data/nprofile.json ssid=InnoPhase passphrase=Inno@1234  $App:git-200f5dc1  SDK Ver: sdk\_2.6.2  Wifi Connect Demo App  addr e0:69:3a:00:2d:fc  host name talaria2  Connecting to added network : InnoPhase  [0.987,027] CONNECT:98:da:c4:73:b7:76 Channel:10 rssi:-45 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [1.261,104] MYIP 192.168.1.222  [1.261,268] IPv6 [fe80::e269:3aff:fe00:2dfc]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Connected to added network : InnoPhase  ------------ Program Exit ------------------- |

## Wifi\_Connect\_Disconnect

### Overview

Sample code in the path:

*apps\using\_wifi\src\wifi\_connect\_disconnect.c* showcases connecting to and disconnecting from a network asynchronously with the API wcm\_auto\_connect() and wcm\_add\_network\_profile().

### Sample Code Walkthrough

This example code initializes two semaphores and creates two threads.

|  |
| --- |
| /\* initializes the semaphores \*/  os\_sem\_init( &my\_sem1, 0 );  os\_sem\_init( &my\_sem2, 0 );  /\* creates a thread \*/  thread1 = os\_create\_thread("thread1", my\_app\_thread\_func1,  (void\*)my\_arg1, MY\_APP\_THREAD\_PRIO, MY\_APP\_THREAD\_STACK\_SIZE);  /\* creates a thread \*/  thread2 = os\_create\_thread("thread2", my\_app\_thread\_func2,  (void\*)my\_arg2, MY\_APP\_THREAD\_PRIO, MY\_APP\_THREAD\_STACK\_SIZE); |

thread2 on running enters a loop where it first waits for a semaphore from thread1.

thread1 on running attempts wcm\_add\_network\_profile()and wcm\_auto\_connect()with param bool enable as 1 to connect, and waits for 10 seconds and finally enters a loop where it:

1. calls wcm\_auto\_connect()with param bool enable as 0 to asynchronously disconnect, without removing the network
2. waits for 10 seconds and unblocks thread2 by posting a semaphore
3. Finally waits on a semaphore posted from thread2, before looping back again to asynchronously disconnect attempt using wcm\_auto\_connect().

|  |
| --- |
| /\*the thread function \*/  static void\* my\_app\_thread\_func1(void\* arg)  {  os\_printf("thread1 prints -- %s\n", (char\*)arg);  /\*Create a Wi-Fi network interface\*/  my\_wcm\_handle = wcm\_create(NULL);  wcm\_notify\_enable(my\_wcm\_handle, my\_wcm\_notify\_cb, NULL);  /\*"/data/nprofile.json"\*/  const char \*np\_conf\_path = os\_get\_boot\_arg\_str("np\_conf\_path")?: NULL;  struct network\_profile \*profile;  int rval;  if (np\_conf\_path != NULL) {  /\* Create a Network Profile from a configuration file in  \*the file system\*/  rval = network\_profile\_new\_from\_file\_system(&profile, np\_conf\_path);  } else {  /\* Create a Network Profile using BOOT ARGS\*/  rval = network\_profile\_new\_from\_boot\_args(&profile);  }  if (rval < 0) {  pr\_err("could not create network profile %d\n", rval);  return NULL;  }  rval = wcm\_add\_network\_profile(my\_wcm\_handle, profile);  if (rval < 0) {  pr\_err("could not associate network profile to wcm %d\n", rval);  return NULL;  }  os\_sleep\_us(2000000, OS\_TIMEOUT\_NO\_WAKEUP);  os\_printf("thread1 prints -- calling api wcm\_auto\_connect(1) to connect to already added network\n");  wcm\_auto\_connect(my\_wcm\_handle, 1);  os\_printf("thread1 prints -- will try a disconnect after 10 seconds... \n");  os\_sleep\_us(10000000, OS\_TIMEOUT\_NO\_WAKEUP);  while(1){  os\_printf("thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..\n");  wcm\_auto\_connect(my\_wcm\_handle, 0);  os\_sleep\_us(10000000, OS\_TIMEOUT\_NO\_WAKEUP);  /\*unlock Thread2 \*/  os\_sem\_post( &my\_sem2 );  /\*block until thread 2 unblocks us \*/  os\_sem\_wait( &my\_sem1 );  }  return NULL;  } |

As noted earlier, thread2 is in a loop, and when unblocked:

1. Calls wcm\_auto\_connect()with param bool enable as 1 to asynchronously connect to previously added network
2. waits for 10 seconds and unblocks thread1 by posting a semaphore
3. enters its loop again where it finally waits on a semaphore posted from thread1, before trying to asynchronously connect attempt using wcm\_auto\_connect().

|  |
| --- |
| /\* the thread function \*/  static void\* my\_app\_thread\_func2(void\* arg)  {  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP);  os\_printf("thread2 prints -- %s\n", (char\*)arg);  while(1){  /\*block until thread 1 unblocks us \*/  os\_sem\_wait( &my\_sem2 );  os\_printf("thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network\n");  wcm\_auto\_connect(my\_wcm\_handle, 1);  os\_sleep\_us(10000000, OS\_TIMEOUT\_NO\_WAKEUP);  /\*unlock Thread1 \*/  os\_sem\_post( &my\_sem1 );  }  return NULL;  } |

In this example, the Talaria TWO module connects and disconnects from the network approximately every 10 seconds without removing the network name.

Notifications from WCM are registered and gets printed just like the previous example.

### Running the Application

Program wifi\_connect\_diconnect.elf (*sdk\_x.y\examples\using\_wifi\bin*) using the Download tool (*sdk\_x.y\pc\_tools\Download\_Tool*) provided with InnoPhase Talaria TWO SDK.

Refer steps mentioned in section 8.1.3 for more details.

### PMK Caching

When Talaria TWO connects to an Access Point, the PMK generated after 802.1X authentication method will be stored in Talaria TWO’s flash and this cached PMK will be used for subsequent connections.

This ensures minimal connection latency between the Access Point and Talaria TWO as it avoids recomputing of PMK for each connection. PMK cache feature is supported only on WPA2-PSK or Mixed mode.

### Expected Output

wifi\_connect\_disconnect.elf execution displays the following output on the console for different scenarios:

#### Case 1

AP is already ON at connection attempt, connect success, alternate connect disconnect with autoconnect API, and add network and remove network, as expected. Background notifications are received.

|  |
| --- |
| UART:SNWWWWAE  Build $Id: git-d198c0771 $  hio.baudrate=921600  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-d198c0771 $  np\_conf\_path=/data/nprofile.json ssid=Asus\_CC\_2G1 passphrase=12345678  SDK Ver: sdk\_2.6.2  Wifi Async Connect Demo App  thread1 prints -- application thread1 will attempt wcm\_add\_network\_profile() and wcm\_auto\_connect(1) and then disconnect with wcm\_auto\_connect(0) 10 seconds after every connection  addr e0:69:3a:00:01:05  thread1 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  thread1 prints -- will try a disconnect after 10 seconds...  [2.730,711] CONNECT:24:4b:fe:5e:fd:d8 Channel:1 rssi:-36 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  [2.769,323] MYIP 192.168.1.4  [2.769,486] IPv6 [fe80::e269:3aff:fe00:105]-link  thread2 prints -- application thread2 will attempt connect using wcm\_auto\_connect(1).  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_DOWN  [12.209,898] DISCONNECTED  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [22.255,835] CONNECT:24:4b:fe:5e:fd:d8 Channel:1 rssi:-37 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [22.303,722] MYIP 192.168.1.4  [22.304,002] IPv6 [fe80::e269:3aff:fe00:105]-link  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_DOWN  [32.210,984] DISCONNECTED  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [42.253,134] CONNECT:24:4b:fe:5e:fd:d8 Channel:1 rssi:-37 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [42.296,531] MYIP 192.168.1.4  [42.296,694] IPv6 [fe80::e269:3aff:fe00:105]-link |

#### Case 2

AP is already ON at connection attempt, wrong password given, connect failure, alternate connect disconnect and add network and remove network as expected, without any thread hanging.

|  |
| --- |
| UART:SNWWWWAE  Build $Id: git-d198c0771 $  hio.baudrate=921600  flash: Gordon ready!  [9.230,063] partitions mounted  UART:SNWWWWAE  Build $Id: git-d198c0771 $  hio.baudrate=921600  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-d198c0771 $  np\_conf\_path=/data/nprofile.json ssid=Asus\_CC\_2G1 passphrase=123456789  SDK Ver: sdk\_2.6.2  Wifi Async Connect Demo App  thread1 prints -- application thread1 will attempt wcm\_add\_network\_profile() and wcm\_auto\_connect(1) and then disconnect with wcm\_auto\_connect(0) 10 seconds after every connection  addr e0:69:3a:00:01:05  thread1 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  thread1 prints -- will try a disconnect after 10 seconds...  [2.741,498] CONNECT:24:4b:fe:5e:fd:d8 Channel:6 rssi:-34 dBm  thread2 prints -- application thread2 will attempt connect using wcm\_auto\_connect(1).  [10.742,444] DISCONNECTED during key negotiation, wrong key?  [10.764,455] Trying to connect in 2 seconds  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [22.378,399] CONNECT:24:4b:fe:5e:fd:d8 Channel:6 rssi:-34 dBm  [30.378,648] DISCONNECTED during key negotiation, wrong key?  [30.400,425] Trying to connect in 2 seconds  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [42.375,801] CONNECT:24:4b:fe:5e:fd:d8 Channel:6 rssi:-33 dBm  [50.376,281] DISCONNECTED during key negotiation, wrong key?  [50.397,366] Trying to connect in 2 seconds |

#### Case 3

AP is switched OFF at connection attempt, alternate connect disconnect and add network and remove network as expected, without any thread hanging.

Later, AP is switched ON, connect success, alternate connect disconnect and add remove as expected.

|  |
| --- |
| UART:SNWWWWAE  Build $Id: git-d198c0771 $  hio.baudrate=921600  flash: Gordon ready!  [9.426,279] partitions mounted  UART:SNWWWWAE  Build $Id: git-d198c0771 $  hio.baudrate=921600  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-d198c0771 $  np\_conf\_path=/data/nprofile.json ssid=Asus\_CC\_2G1 passphrase=12345678  SDK Ver: sdk\_2.6.2  Wifi Async Connect Demo App  thread1 prints -- application thread1 will attempt wcm\_add\_network\_profile() and wcm\_auto\_connect(1) and then disconnect with wcm\_auto\_connect(0) 10 seconds after every connection  addr e0:69:3a:00:01:05  thread1 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  thread1 prints -- will try a disconnect after 10 seconds...  [2.742,962] CONNECT:24:4b:fe:5e:fd:d8 Channel:6 rssi:-34 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [4.689,892] MYIP 192.168.1.4  [4.690,055] IPv6 [fe80::e269:3aff:fe00:105]-link  thread2 prints -- application thread2 will attempt connect using wcm\_auto\_connect(1).  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_DOWN  [12.205,311] DISCONNECTED  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [22.613,616] Trying to connect in 1 seconds  [23.739,080] Trying to connect in 2 seconds  [25.864,544] Trying to connect in 4 seconds  [29.990,008] Trying to connect in 8 seconds  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [42.717,590] Trying to connect in 1 seconds  [43.843,054] Trying to connect in 2 seconds  [45.968,518] Trying to connect in 4 seconds  [50.093,983] Trying to connect in 8 seconds  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  thread2 prints -- calling api wcm\_auto\_connect(1) to connect to already added network  [62.389,674] CONNECT:24:4b:fe:5e:fd:d8 Channel:6 rssi:-33 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [62.503,180] MYIP 192.168.1.4  [62.503,635] IPv6 [fe80::e269:3aff:fe00:105]-link  thread1 prints -- calling api wcm\_auto\_connect(0) to just disconnect without removing network..  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_DOWN  [72.207,219] DISCONNECTED |

## Wi-Fi Scan

### Overview

The sample code in the path *example\using\_wifi\src\wifi\_scan.c* scans for available Wi-Fi networks and prints them out.

Following are the steps:

1. Create a data structure to store the parameters and results of scan.
2. Set the default parameter for scanning using the API wifi\_init\_scan\_default()
3. In a loop, let the code scan and print the nearby networks from vicinity for every 10 second interval of time.

### Sample Code Walkthrough

The necessary data structures to store parameters and result of scan are created as shown:

|  |
| --- |
| int main(void)  {  struct wcm\_handle \*h;  const size\_t max\_nets = 64;  struct wifi\_netinfo \*\*scan\_result;  struct wifi\_scan\_param param;  …  …  …  scan\_result = os\_alloc(max\_nets \* sizeof(void \*));  assert(scan\_result != NULL); |

wifi\_init\_scan\_default()API is used to set the default parameters for the scanning, and wcm\_scan() is used with explained parameters to start scanning.

The example code runs in a loop, scans and prints the results using wifi\_netinfo\_get\_ssid() and wifi\_netinfo\_get\_chan(), frees up the memory allocated for scan result using wcm\_free\_scanresult() and waits for 10 seconds before scanning and printing again as shown in the following code:

|  |
| --- |
| wifi\_init\_scan\_default(&param);    // print scan parameters  print\_scan\_params(param);    for (;;) {  // perform scan  int n = wcm\_scan(h, &param, scan\_result, max\_nets);  // print out results of scan  os\_printf("Found %d nets:\n", n);  for (int i = 0; i < n; i++) {  uint8\_t chan;  struct wifi\_ssid ssid;  wifi\_netinfo\_get\_ssid(scan\_result[i], &ssid);  wifi\_netinfo\_get\_chan(scan\_result[i], &chan);  os\_printf("%6pM on channel %2d @ %3d '%s'\n",  scan\_result[i]->ni\_bssid, chan, scan\_result[i]->ni\_rssi, ssid.ws\_ssid);  }  wcm\_free\_scanresult(scan\_result, n);  os\_sleep\_us(10000000, OS\_TIMEOUT\_NO\_WAKEUP);  } |

### Running the Application

Program wifi\_scan.elf (*sdk\_x.y\examples\using\_wifi\bin*)using the Download tool(*sdk\_x.y\pc\_tools\Download\_Tool*)provided with InnoPhase Talaria TWO SDK.

Refer steps mentioned in section 8.1.3 for more details.

### Expected Output

wifi\_scan.elf execution displays the following output on the console for different scenarios:

|  |
| --- |
| UART:SNWWWWWAEBuild $Id: git-ba65998b7 $  np\_conf\_path=/data/nprofile.json ssid=InnoPhase passphrase=43083191  SDK Ver: sdk\_2.5  Wifi Scan Demo App  addr e0:69:3a:00:13:90  Scan parameters:  channel\_masks: 255 255 255 255 255 255 255 255  bssid: 0xFFFFFFFFFFFF  txrate: 0  waittime: 0  ie list: 0x  Found 6 nets:  6e:8a:ce:99:b1:c8 on channel 6 @ -39 'Inno' 'OPEN'  00:5f:67:cd:c5:a6 on channel 11 @ -46 'InnoPhase' 'WPA2-PSK'  e4:a7:c5:d4:ea:86 on channel 6 @ -74 'Airtel-E5573-EA86' 'WPA2-PSK'  70:4f:57:77:7e:d4 on channel 2 @ -74 'Rahul' 'WPA2-PSK'  d8:0f:99:72:13:65 on channel 5 @ -79 'JioFi3\_721365' 'WPA-PSK/WPA2-PSK Mixed Mode'  d8:47:32:2e:e1:e0 on channel 11 @ -80 'GPMH' 'WPA2-PSK'  Found 7 nets:  6e:8a:ce:99:b1:c8 on channel 6 @ -34 'Inno' 'OPEN'  00:5f:67:cd:c5:a6 on channel 11 @ -46 'InnoPhase' 'WPA2-PSK'  ba:6b:ad:62:6d:8b on channel 11 @ -72 'DESKTOP-9B1DNVC 1786' 'WPA2-PSK'  34:0a:33:70:f3:a2 on channel 1 @ -76 'Siddusm' 'WPA2-PSK'  dc:71:37:56:91:b0 on channel 8 @ -77 'Hathway\_Raghuram' 'WPA-PSK/WPA2-PSK Mixed Mode'  d8:0f:99:72:13:65 on channel 5 @ -78 'JioFi3\_721365' 'WPA-PSK/WPA2-PSK Mixed Mode'  ac:84:c6:88:10:46 on channel 13 @ -85 'Lakshmi pg 3rd floor ' 'WPA2-PSK'  Found 6 nets:  6e:8a:ce:99:b1:c8 on channel 6 @ -33 'Inno' 'OPEN'  00:5f:67:cd:c5:a6 on channel 11 @ -47 'InnoPhase' 'WPA2-PSK'  ba:6b:ad:62:6d:8b on channel 11 @ -71 'DESKTOP-9B1DNVC 1786' 'WPA2-PSK'  d8:0f:99:72:13:65 on channel 5 @ -77 'JioFi3\_721365' 'WPA-PSK/WPA2-PSK Mixed Mode'  b0:a7:b9:73:8e:51 on channel 4 @ -78 'Lakshmi pg 2nd floor 2g' 'WPA2-PSK'  dc:71:37:56:91:b0 on channel 8 @ -79 'Hathway\_Raghuram' 'WPA-PSK/WPA2-PSK Mixed Mode' |

### Power Optimization with Rx Nap Scan

In a crowded environment, Rx Nap Scan feature can be enabled to save power during the scan process. In this mode, the Rx nap function will turn the Talaria TWO’s receiver OFF (*takes a nap*) for the duration of the frame when there are uninteresting frames with high signal strength. The frames of interest are probe responses and beacon frames.

This mode is enabled by default in wifi\_scan.elf and can be disabled using the following boot argument:

|  |
| --- |
| wifi.nap\_scan=0 |

As shown in Figure 4, Talaria TWO takes naps when there are uninteresting frames. During the naps, current consumption dips to 8mA whereas during the scan period, the average Rx current remains at ~33mA.



Figure 4: Talaria TWO naps during uninteresting frames

Rx nap scan mode disabled is as shown in Figure 5.

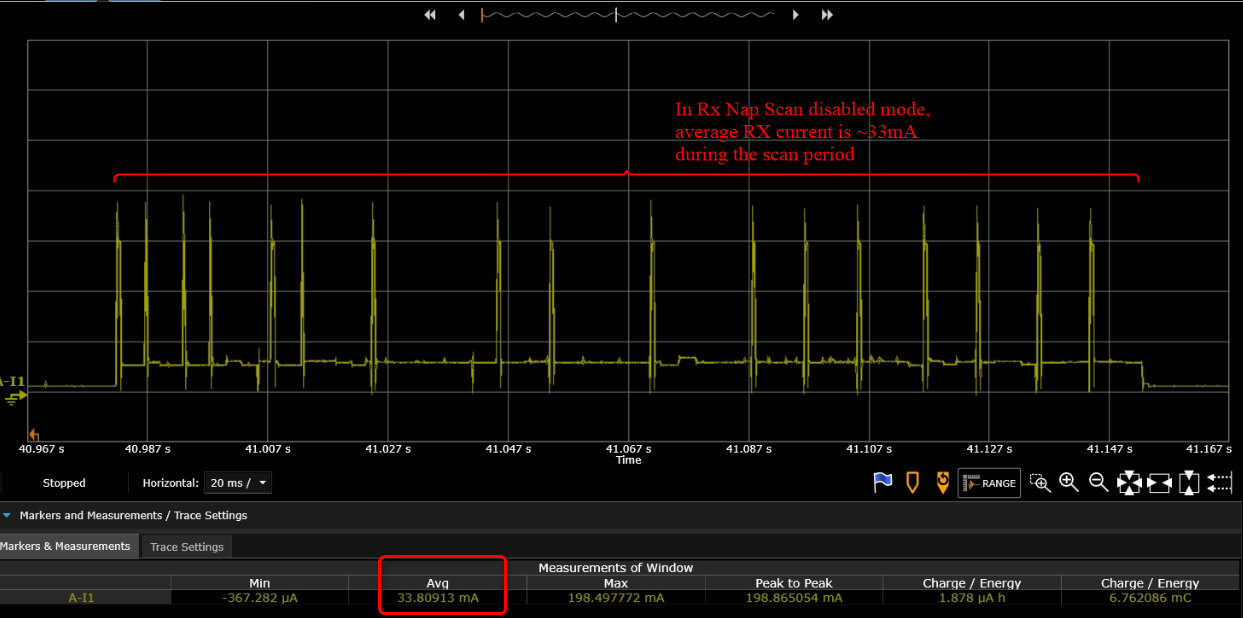


Figure 5: Rx nap scan mode disabled