Low-Power Scan

# Introduction

This application note describes the basics of the Low Power Wi-Fi scan feature in InnoOS™ with an example illustrating Talaria TWO implementation, using the provided data structs and API calls in the SDK release.

# System / API Functions

## SDK Provided Data Structures and Objects

|  |  |
| --- | --- |
| **Data Structure/Object** | **Description** |
| *struct wcm\_handle* | Handle to the Wi-Fi connection manager. It is a pointer return from the API call wcm\_create(). |
| *struct wifi\_netinfo* | Used to hold the properties of a Wi-Fi network |
| *struct wifi\_scan\_param* | Used to hold the parameters for Wi-Fi scan operation. |
| *struct os\_workqueue* | For the OS queue data |

Table : SDK Provided Data Structures/Objects

**Note**:

1. Wi-Fi data structures are declared in: SDK\include\wifi\scan.h
2. struct os\_workqueue is declared in: \include\kernel\workqueue.h

## API Functions

### wcm\_create()

Create a Wi-Fi network interface and return a struct wcm\_handle pointer

|  |
| --- |
| static inline struct wcm\_handle \*  wcm\_create(const uint8\_t \*hwaddr)  {  ……  } |

### wifi\_init\_scan\_default()

Get the current Wi-Fi scan parameters.

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

wifi\_init\_scan\_default() is called to get the current Wi-Fi scan parameters in the param of struct wifi\_scan\_param.

### os\_init\_workqueue ()

Initialize a work queue with an os\_workqueue object.

|  |
| --- |
| void os\_init\_workqueue(struct os\_workqueue \*wq); |

### os\_init\_delayed\_work()

Initialize a work queue with an os\_workqueue object with the delayed occurrence for the next callout.

|  |
| --- |
| int os\_queue\_delayed\_work(struct os\_delayed\_work \*dw, struct os\_workqueue \*wq, uint32\_t expire); |

### os\_run\_work()

Start the queue work with os\_workqueue object.

|  |
| --- |
| void os\_run\_work(struct os\_workqueue \*wq); |

### wcm\_scan()

Perform the Wi-Fi scan with the parameter block. Results are contained in the buffer pointed with the parameter wifi\_netinfo \*\*result.

|  |
| --- |
| int wcm\_scan(struct wcm\_handle \*h, const struct wifi\_scan\_param \*param, struct wifi\_netinfo \*\*result, size\_t max); |

### A Set of Wi-Fi Scan Related Data Retrieval

A set of Wi-Fi scan related functions for retrieval of data from wifi\_netinfo.

|  |
| --- |
| int wifi\_netinfo\_get\_ssid(const struct wifi\_netinfo\* ni, struct wifi\_ssid\* ssid);  int wifi\_netinfo\_get\_chan(const struct wifi\_netinfo\* ni, uint8\_t\* chan);  int wifi\_netinfo\_get\_authmode(const struct wifi\_netinfo\* ni,struct wifi\_authmode\* authmode);  size\_t wifi\_netinfo\_authmode\_tostr(uint32\_t authmask, char\* mode\_name, size\_t size); |

# Code Walkthrough

## lpscan.c

### Overview

The sample code is the path: examples/lpscan/src/lpscan.c is a simple application which demonstrates the low power scan feature.

### Sample Code Walkthrough

1. User-defined Data Struct for the Scan Work Task:

Declare a data struct scan\_workq\_task for the scan work task:

|  |
| --- |
| struct scan\_workq\_task {  struct os\_delayed\_work dwork;  struct os\_workqueue wqueue;  struct {  uint32\_t counter;  size\_t max\_nets;  int sum\_total;  struct wcm\_handle \*h;  const struct wifi\_scan\_param \*param;  struct wifi\_netinfo \*\*scan\_result;  }conf;  }; |

1. Static Variables for the Scan Work Task

Define the static variables for the scan feature:

|  |
| --- |
| #define TIME\_BETWEEN\_SCAN\_ITERS 10  static bool ap\_logging;  static uint32\_t num\_iterations, dt\_iterations; |

1. Creation of Scan Work Control Object

Use the SDK provided data struct scan\_workq\_task to create the scan work control:

|  |
| --- |
| struct scan\_workq\_task \*sworkt;  if (!sworkt)  return -1; |

1. Initialization of the Static Variables for the Scan Work Task:

The app initializes the static variables from the boot arguments:

|  |
| --- |
| ap\_logging = os\_get\_boot\_arg\_int("ap\_logging", 0);  //0 means infinity  num\_iterations = os\_get\_boot\_arg\_int("num\_iterations", 0);  //time between iterations (in seconds)  dt\_iterations = os\_get\_boot\_arg\_int("dt\_iterations", TIME\_BETWEEN\_SCAN\_ITERS);  assert(dt\_iterations >= 5); |

1. Creation of WCM Object for the Wi-Fi Interface

In the following code, a Wi-Fi handle is created. The NULL parameter in the wcm\_create() specifies that no user-specified MAC address is used, and the MAC address in the flash will be used to create the Wi-Fi interface.

|  |
| --- |
| h = wcm\_create(NULL);  assert(h != NULL} |

1. Creation of Local Data for Wi-Fi Scan Results

Use the SDK provided data struct wifi\_netinfo as the data object for the scan results. This is the scan result data container that will be populated by the scan\_workq\_task.

|  |
| --- |
| struct wifi\_netinfo \*\*scan\_result;  scan\_result = os\_alloc(max\_nets \* sizeof(void \*));  assert(scan\_result != NULL); |

1. Set-up of Wi-Fi Scan Parameter Block

Use the SDK provided data struct wifi\_scan\_param to retrieve the default values for the Wi-Fi scan task from SDK in the param, get the user-entry values from the bootargs, convert the values for format, and populate the param with the new values for the scan task.

|  |
| --- |
| struct wifi\_scan\_param param;  wifi\_init\_scan\_default(&param);  uint32\_t channel\_mask = os\_get\_boot\_arg\_int("wifi.scan\_channel\_mask", 0xffffffff);  memset(param.channel\_mask, 0, sizeof(param.channel\_mask));  memcpy(&param.channel\_mask, &channel\_mask, sizeof(channel\_mask));  tmp = os\_get\_boot\_arg\_str("addr");  if (tmp)  parse\_macaddr(tmp, param.bssid);  param.txrate = os\_get\_boot\_arg\_int("rate", RATE\_6);  tmp = os\_get\_boot\_arg\_str("scan\_ssid");  if (tmp)  wifi\_ssid\_from\_bytes(&param.ssid, tmp, strlen(tmp));  max\_listen\_time = os\_get\_boot\_arg\_int("wifi.scan\_max\_listen\_time", param.max\_listen\_time/SYSTIME\_MS(1));  min\_listen\_time = os\_get\_boot\_arg\_int("wifi.scan\_min\_listen\_time", param.min\_listen\_time/SYSTIME\_MS(1));  wait\_time = os\_get\_boot\_arg\_int("wifi.scan\_wait\_time", param.wait\_time/SYSTIME\_MS(1));  probe\_tx\_timeout= os\_get\_boot\_arg\_int("wifi.scan\_probe\_tx\_timeout", param.probe\_tx\_timeout/SYSTIME\_MS(1));  param.min\_listen\_time = SYSTIME\_MS(min\_listen\_time);  param.max\_listen\_time = SYSTIME\_MS(max\_listen\_time);  param.wait\_time = SYSTIME\_MS(wait\_time);  param.probe\_tx\_timeout= SYSTIME\_MS(probe\_tx\_timeout);  param.num\_probes = os\_get\_boot\_arg\_int("wifi.scan\_num\_probes", param.num\_probes);  param.idleslots = os\_get\_boot\_arg\_int("wifi.scan\_idleslots", param.idleslots);  param.max\_responses = os\_get\_boot\_arg\_int("wifi.scan\_max\_responses", param.max\_responses); |

1. Setup for the Suspend Mode Enable

|  |
| --- |
| /\* Enable device suspend (deep sleep) via boot argument \*/  if (os\_get\_boot\_arg\_int("suspend", 1) != 0)  os\_suspend\_enable(); |

1. Setup of the Scan Work Control Block

Populate the Scan work control block sworkt with the parameter block and the scan result object:

|  |
| --- |
| //set up confs  sworkt->conf.counter = 0;  sworkt->conf.max\_nets = max\_nets;  sworkt->conf.h = h;  sworkt->conf.param = &param;  sworkt->conf.scan\_result = scan\_result; |

1. Initialization of Work Queue and Start of Work

Set up and start the work queue for the scan work:

|  |
| --- |
| //initialize  os\_init\_workqueue(&sworkt->wqueue);  os\_init\_delayed\_work(&sworkt->dwork, scan\_work\_clbk);  os\_queue\_delayed\_work(&sworkt->dwork, &sworkt->wqueue, os\_systime()+SYSTIME\_MS(10));  os\_run\_work(&sworkt->wqueue); |

1. Definition of Scan Callback Function scan\_work\_clbk()

The callback function scan\_work\_clbk() is called every timeout of the interval that is specified with dt\_iterations.

|  |
| --- |
| static void  scan\_work\_clbk(struct os\_work \*w)  {  struct scan\_workq\_task \*twork = container\_of(w, struct scan\_workq\_task, dwork.work);  int num\_ap\_found;  uint32\_t next\_tmo = os\_systime() + SYSTIME\_SEC(dt\_iterations);  twork->conf.counter++;  num\_ap\_found = wcm\_scan(twork->conf.h,  twork->conf.param,  twork->conf.scan\_result,  twork->conf.max\_nets);  pr\_always("Round:%u Found %d nets:\n",  twork->conf.counter, num\_ap\_found);  /\*  More code …  \*/  //reschedule  os\_queue\_delayed\_work(&twork->dwork, &twork->wqueue, next\_tmo);  } |

The callback function is registered with the system by the statement:

|  |
| --- |
| //initialize  ……;  os\_init\_delayed\_work(&sworkt->dwork, scan\_work\_clbk);  ……; |

1. Display of the Scan Results

Under the set of the ap\_logging flag, the data list from the scan work is populated with API calls.

|  |
| --- |
| if (ap\_logging) {  for (int i = 0; i < num\_ap\_found; i++) {  uint8\_t chan;  char mode\_name[64];  struct wifi\_ssid ssid;  struct wifi\_authmode authmode;  wifi\_netinfo\_get\_ssid(twork->conf.scan\_result[i], &ssid);  wifi\_netinfo\_get\_chan(twork->conf.scan\_result[i], &chan);  wifi\_netinfo\_get\_authmode(twork->conf.scan\_result[i], &authmode);  wifi\_netinfo\_authmode\_tostr(authmode.authmask, mode\_name, 64);  pr\_always("%6pM on channel %2d @ %3d '%s' '%s'\n",  twork->conf.scan\_result[i]->ni\_bssid,  chan,  twork->conf.scan\_result[i]->ni\_rssi,  ssid.ws\_ssid,  mode\_name );  }  }  wcm\_free\_scanresult(twork->conf.scan\_result, num\_ap\_found); |

### Running the Application

Program lpscan.elf (sdk\_x.y\examples\lp\_scan\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 lpscan.elf by clicking on Select ELF File.
   3. Boot arguments: Pass the following boot arguments:

|  |
| --- |
| wifi.scan\_min\_listen\_time=8, wifi.scan\_max\_listen\_time=24, wifi.scan\_num\_probes=1, wifi.scan\_idleslots=3, wifi.nap\_scan=1, dt\_iterations=10, ap\_logging=1, suspend=1 |

* 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

lpscan.elf is created when compiling the code which provides the following console output when programmed to Talaria TWO:

**Note**: The following console output is from SDK 2.3 release and is applicable to the current release as well.

|  |
| --- |
| UART:NWWWWAEBuild $Patch: git-5e70acd25 $ $Id: git-c74d301bc $  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:PNWWWWWWAEBuild $Id: git-f92bee540 $  wifi.scan\_min\_listen\_time=8 wifi.scan\_max\_listen\_time=24 wifi.scan\_num\_probes=1 wifi.scan\_idleslots=3 wifi.nap\_scan=1 dt\_iterations=10 ap\_logging=1 suspend=1  addr 02:03:04:66:f4:63  [0.566,075] Round:1 Found 12 nets:  -------------------------  [0.566,503] ac:9e:17:45:fc:28 on channel 8 @ -36 'Asus\_AC87U' 'WPA2-PSK+PMF'  [0.566,726] 10:c3:7b:52:6b:c8 on channel 11 @ -39 'dtim100' 'WPA2-PSK+PMF'  [0.566,946] f6:c3:2a:92:3a:fa on channel 6 @ -40 '' 'WPA2-PSK'  [0.567,083] 00:24:a5:f1:8c:9e on channel 10 @ -45 'IP006' 'WPA-PSK/WPA2-PSK Mixed Mode'  [0.567,199] 74:da:88:a9:02:dd on channel 4 @ -46 'TPLinkC3' 'WPA2-PSK'  [0.567,310] ec:41:18:1a:d5:05 on channel 13 @ -46 'IP014' 'WPA2-PSK'  [0.567,423] 28:80:88:28:4c:d2 on channel 1 @ -48 'NGR8000\_1' 'WPA2-PSK+PMF'  [0.567,533] 8c:fe:74:35:40:a8 on channel 6 @ -54 'RuckusR500' 'WPA2-PSK'  [0.567,670] bc:54:fc:c7:1c:8c on channel 8 @ -57 'IP011' 'WPA-PSK/WPA2-PSK Mixed Mode'  [0.567,789] 14:d6:4d:25:67:b2 on channel 2 @ -57 'dlink' 'WPA2-PSK'  [0.567,896] d0:ae:ec:99:97:fc on channel 12 @ -57 '360cn' 'WPA2-PSK'  [0.568,339] 08:9b:b9:3d:1d:14 on channel 1 @ -65 'ATTsyYY4SQ' 'WPA2-PSK+PMF'  [10.600,973] Round:2 Found 10 nets:  -------------------------  [10.601,061] 04:d4:c4:3e:74:88 on channel 6 @ -36 'AsusC3' 'WPA2-PSK+PMF'  [10.601,174] 10:c3:7b:52:6b:c8 on channel 11 @ -38 'dtim100' 'WPA2-PSK+PMF'  [10.601,290] 3c:7c:3f:62:5f:f8 on channel 6 @ -39 'AsusC4' 'WPA2-PSK+PMF'  [10.601,403] e4:c3:2a:92:3a:fa on channel 6 @ -40 'TPLinkC4' 'WPA2-PSK'  [10.601,843] 86:da:88:a9:02:dd on channel 4 @ -43 '' 'WPA2-PSKip' 'WPA2-PSK+PMF'  [10.603,544] d0:ae:ec:99:97:fc on channel 12 @ -58 '360cn' 'WPA2-PSK'  [10.603,681] bc:54:fc:c7:1c:8c on channel 8 @ -59 'IP011' 'WPA-PSK/WPA2-PSK Mixed Mode'  [10.603,795] 84:82:f4:35:f5:58 on channel 13 @ -61 'IP021' 'WPA2-PSK'  [10.604,019] 08:9b:b9:3d:1d:14 on channel 1 @ -63 'ATTsyYY4SQ' 'WPA2-PSK+PMF'  [10.604,243] 14:1f:ba:7a:30:04 on channel 11 @ -66 'NVR-2.4G\_2M024D2PAZ00916' 'WPA2-PSK' |