Firmware Over the Air

# Introduction

Firmware-Over-the-Air (FOTA) allows wireless delivery of firmware updates and/or configurations to embedded devices.

This document describes the FOTA process for the Talaria TWO EVB using the Talaria TWO SDK with details on how to implement or trigger FOTA in a customer provided application.

# Topology

FOTA

Talaria TWO

Amazon S3

Figure 1: Topology

# Overview

This implementation of FOTA provides the following features:

1. Check for the availability of new upgrades.
2. Securely download the image into flash.
3. Check the validity of the downloaded image.
4. Set the new image as the boot image.

In conjunction with SSBL, it enables booting the latest image downloaded. The firmware is downloaded into the application image partition in the Flash.

# List of APIs

* 1. fota\_init(): Initializes the FOTA module.
  2. fota\_perform(): Performs the FOTA update.
  3. fota\_commit(): After the FOTA update is done, this function is called to set the newly updated firmware as the default.
  4. fota\_deinit(): Releases all the resources allocated during fota\_init() and other FOTA API calls.
  5. os\_free(): Returns allocated memory to the heap. If the memory has more than one reference, the count is simply dropped by one.

# Features & Limitations

Following are the FOTA application features:

1. FOTA over HTTP/HTTPS.
2. Image download from Cloud or any HTTP/web server.
3. Two copy solution. Backup copy of the correct firmware always exists.
4. Image integrity check using sha256 hash.
5. Error handling and recovery
   1. If any error occurs during downloading the image or updating the configuration files (*part.json/boot.json/fota\_config.json*), the device will remain in the current image.
   2. If a reboot occurs (due to issues like power failure) during image download or configuration files upgrade, the device will boot with the current image.
6. JSON based configuration.

# Dependent Talaria TWO Module Information

This section provides a list of modules in Talaria TWO on which FOTA is dependent. It is important to understand these concepts before proceeding with the design aspects of the FOTA.

## Flash Layout

About Talaria TWO Flash:

1. Size: 2MB
2. 512 sectors
3. 4096 bytes/sector
4. 256-byte page

Flash is divided into eight partitions. Partition table information is stored in the Boot sector. Each partition has a starting sector and a sector count, along with a type, and some control bits. No two partitions overlap. The reason for using sector addressing is so that partitions can be independently erased.

Figure 2 provides the proposed layout of Flash memory when using SSBL. To use SSBL, Flash must at least contain SSBL, filesystem, and one application.

Graphical user interface, application

Description automatically generated

Figure 2: Flash layout when using the SSBL

The Boot Image is the default application that Talaria TWO’s boot ROM would look for when a Talaria TWO device is powered ON. To support FOTA, SSBL shall run as Boot image. SSBL is a special application that determines the final application to load. In a nutshell, on power cycle, the boot ROM boots the SSBL application which in turn loads the final application

For detailed documentation on Flash layout, refer: Application\_for\_using\_SSBL.pdf *(sdk\_x.y/apps/ssbl/doc/)*.

**Note**: x and y in sdk\_x.y refers to the SDK package release version.

## Partition Table File (part.json)

This is a json file that provides the partition information of the application images in the Flash. The file is stored in root/user FS (*sdk\_x.y/apps/fota/fs*). This file mainly contains an array of image information (represented by the name **image:**).

Each of the image information entry in the array gives image name, version, starting sector and other information about the application. Following is the basic content:

|  |
| --- |
| {  "image" : [  {  "name" : "fota",  "version" : "1.0",  "start\_sector" : 32,  "bootargs\_start": 1,  "ssid" : "inno\_test",  "passphrase" : "1234567890",  "bootargs\_end" : 1  },  {  "name" : "test\_app",  "version" : "1.0",  "start\_sector" : 154,  "bootargs\_start": 1,  "ssid" : "inno\_test",  "passphrase" : "1234567890",  "bootargs\_end" : 1  },  {  "name" : "test\_app",  "version" : "0.0",  "start\_sector" : 230,  "bootargs\_start": 1,  "ssid" : "inno\_test",  "passphrase" : "1234567890",  "bootargs\_end" : 1  }  ],  "baudrate" : 2560000,  "timeout" : 0,  "verbose" : 1  } |

In the part.json file from the above example, the start sector computation for applications can be done as:

The start sector for the first application (FOTA) in the partition table is 32. For the next consecutive applications (test\_app), start sector can be calculated based on the size of ELF i.e.,

Size of FOTA ELF = 519356.

Start sector of FOTA application = 32.

Total number of sectors needed for FOTA application: 519356/4096=122 sectors, where 4096 is the size of one sector.

The next application, “test\_app” start sector can be flashed on or after 32+122 sectors = 154th sector.

## Boot Index File (boot.json)

This is a json file stored in root/user FS. It contains the image index. This is the index in the image information array present in part.json file. SSBL gets the index of the image to be loaded from this file.

Following is the content:

|  |
| --- |
| {  image : 0  } |

## FOTA Configuration File (fota\_config.json)

The FOTA configuration file fota\_config.json is a json file. This file is stored in the root/user FS in Flash. The FOTA module gets all the information required to download the Firmware or a file.

Each object in this file shall give information about the file to be downloaded. Each object will have the following tokens:

1. type: Type of the file. It can be firmware or file
2. name: Name of the firmware image/ file
3. hostname: Fully Qualified domain name of the server
4. port: Server port
5. uri: This is the location of the firmware/file in the cloud
6. secured: Value for this token will be 2 if the connection is secure with server authentication, else 1
7. ca\_cert: Certificate file name
8. hash: Hash used for checking the integrity of the firmware/file

Following is the basic content of the file:

|  |
| --- |
| {  "package\_version" : "1.0",  "files" : [  {  "type" : "configuration",  "name" : "fota.config",  "hostname" : "innotestota.s3.us-east-2.amazonaws.com",  "port" : 443,  "secured" : 2,  "uri" : "/fota\_config.json",  "ca\_cert" : "/data/fota\_ca\_cert.pem"  },  {  "type" : "firmware",  "name" : "test\_app",  "hostname" : "innotestota.s3.us-east-2.amazonaws.com",  "port" : 443,  "secured" : 2,  "uri" : "/test\_app.elf",  "ca\_cert" : "/data/fota\_ca\_cert.pem"  }  ]  } |

The group of Firmware and files and its information present in this file is considered as a package. Each fota\_config.json file will have a package version at the top. The array of objects will provide information about firmware and files considered as one package.

The package\_version provides the version of the package. There will be a fota\_config.json file in the Cloud. If the package\_version of the fota\_config.json file present in the Cloud is greater than that of the file currently present in the device, FOTA needs to be done.

The first object shall give the information about the fota\_config.json file available on Cloud. Device can fetch the file and see if a package with a higher version is available. The Firmware will be downloaded in the application partition and files will be stored in root/user FS.

## Secure Secondary Boot Loader (SSBL)

SSBL is an application that facilitates booting a specific image from the flash. On boot, the boot-ROM loads & starts SSBL. SSBL reads the image index from the boot.json file. It parses the part.json file and picks the image info in the image info array at the index read from boot.json file. The SSBL then loads and runs the image at the sector provided by this image information.

For detailed information about the SSBL design, refer: Application\_for\_using\_SSBL.pdf (*sdk\_x.y/apps/ssbl/doc/*).

# Design

FOTA process involves the following components:

1. Parsing the FOTA configuration file
2. Checking for the new updates
3. Selecting image area
4. Secured connection
5. Downloading the Firmware
6. Error handling

## Checking for New Updates

For checking new updates, module fetches the fota\_config.json file from the cloud. The package version of the downloaded file is compared against the fota\_config.json file already present in the device. If the version is higher, FOTA needs to be done.

This functionality is optional, and the step can be skipped if an external application like Mobile Application does the check and provisions the device to trigger the FOTA. The functionality is provided through API for the applications to be used for polling.

## Selecting Image Area

This logic will parse the part.json file and selects the image area in flash for downloading the image.

Each application that can be upgraded using FOTA will have a unique name in the image information table. Multiple image information entries for the same application will have the same name. That is, each such application will have at-least two slots in the table.

For example, if there is an application called app\_image, there will be two entries in the image information table with the same name. There will be a minimum of two entries for an application which can be upgraded using FOTA.

The version field in the image information shall represent the FOTA version and not the application release version. The selection logic will go through all the entries for a given application and selects area (image information) with least version number.

For example, if one entry for app\_image has version 1 and its starting sector is 66 and other entry for the same application has the version 0 and its starting sector is 166, the first entry will be selected for FOTA image download. The new image will be downloaded at sector 66.

Each time after FOTA succeeds, the version number for the selected image information is changed to one more than the highest currently available version, so that the newer version will always have the highest version number.

## Secured Connection

The fota\_config.json file provides the following information for connection and download:

1. Server IP/ DNS
2. Port number
3. Firmware location on the server (URI)
4. Root CA certificate to authenticate the server at the time of SSL connection

If the DNS name is provided, DNS will be resolved. The root CA certificate as indicated in the fota\_config.json file will be present in the root/user FS. HTTPS connection will be established with the server. The connection will be secured using Transport Layer Security (TLS1.2).

## Downloading the Firmware

Once the HTTPS connection is successfully established, the image is downloaded using HTTP GET. The URI of the Firmware as provided in the fota\_config.json file is used during the GET. The image is downloaded into flash at the location selected as detailed in section 9.2.

After successful download, image is authenticated using the certificate indicated by ca\_cert field in fota\_config.json file. This will also ensure that the integrity of the image is intact. This certificate will be present in the root/user FS.

## Setting the new image for boot and reload

If the image integrity of the downloaded image is found to be intact, version number of the selected image information in part.json file will be increased by one more than the highest version currently in use. Finally, image index in boot.json file will be updated with the index of the selected image information and the device is reset. After reboot, SSBL will automatically load the newly downloaded image.

## Error handling

The FOTA alternates the image download between two application image area in flash. At any point of time there will at-least one proper application image (currently running). This acts as a backup/fallback image in case FOTA fails. The boot image index in boot.json file is changed to point to the new image only at the last step of FOTA after the integrity of the downloaded image is found to be intact.

At any point of time if the error occurs, the procedures can be retried. The procedure will be retried for FOTA\_MAX\_RETRIES multiple times before giving up. If FOTA is not successful, the currently available stable image will run.

## Flow Diagram

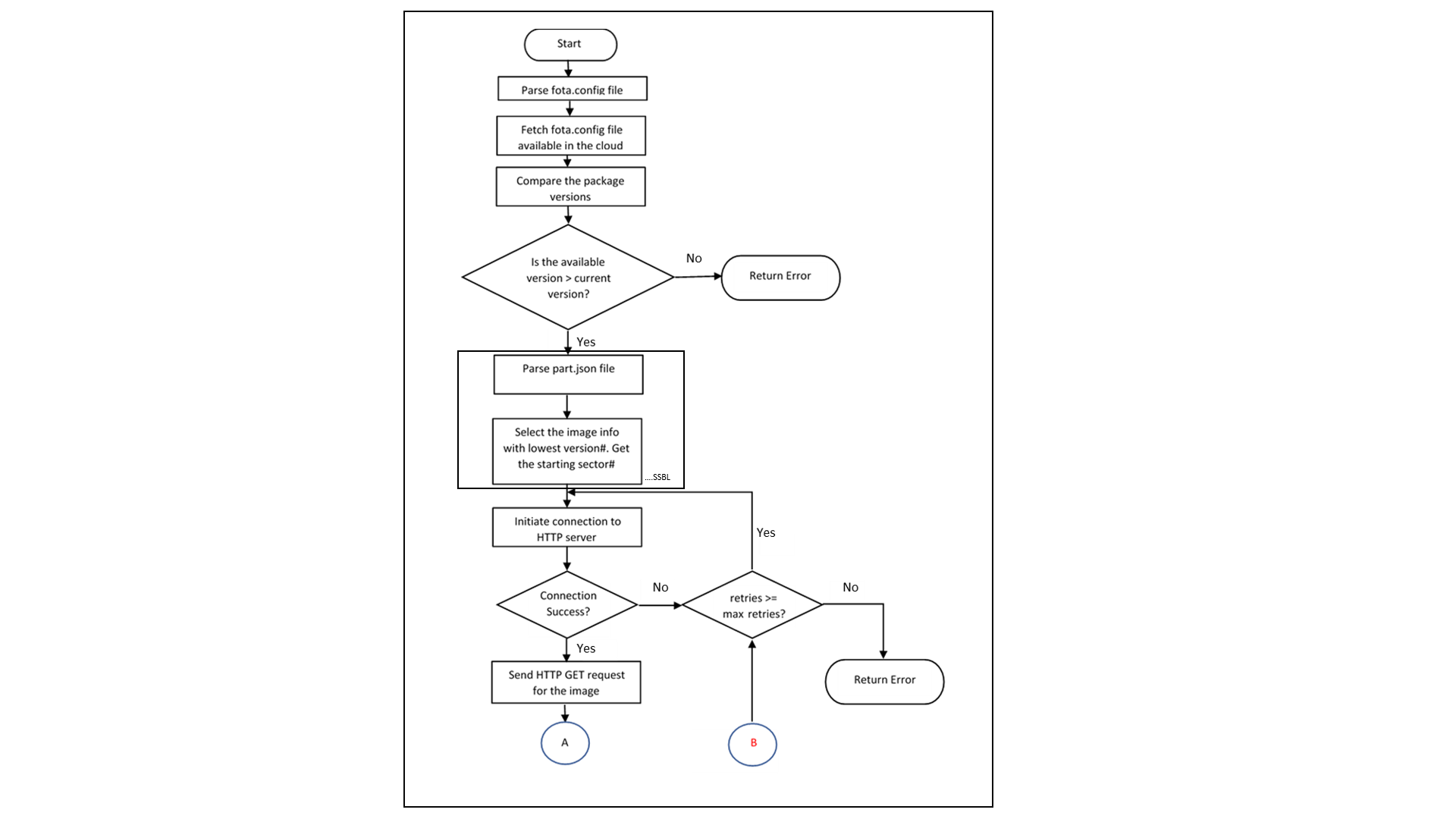


Figure 3: Flow Diagram

Continued from the previous flow diagram:

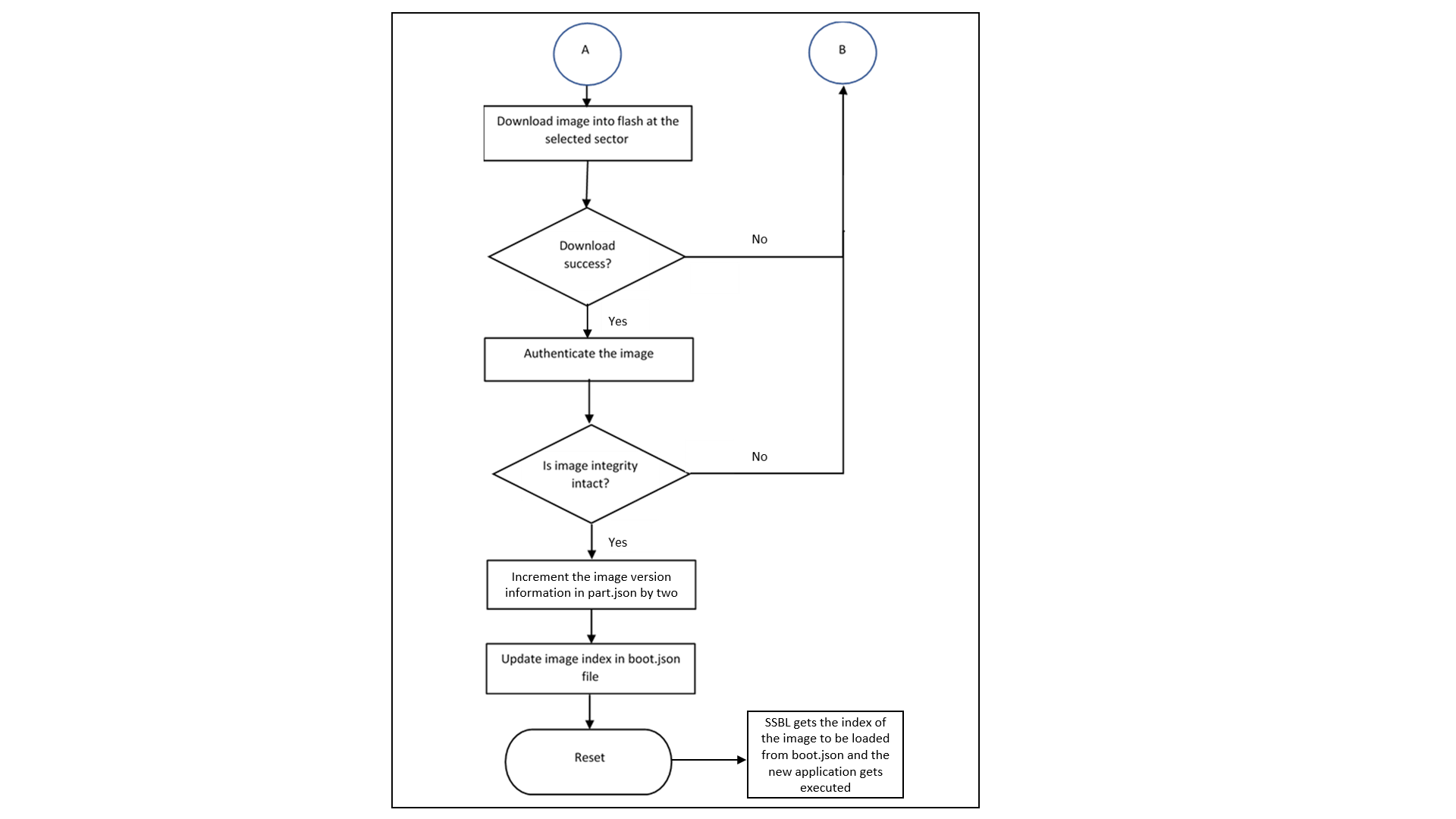


Figure 4: Flow Diagram - continued

## Code Walkthrough

### Initializing FOTA

The fota\_init() API initializes the FOTA module. This will be called before any other FOTA APIs.

fota\_init\_param needs to be initialized appropriately before passing it onto the fota\_init() function. Following is the definition for fota\_init\_param:

|  |
| --- |
| typedef struct {      uint32\_t \*cipher\_key; /\*\*cipher key used with secureboot\*/  }fota\_init\_param\_t; |

cipher\_key in fota\_init\_param will be NULL in case of non-secureboot. In case of secureboot, this should be initialized appropriately.

|  |
| --- |
| fota\_handle\_t \*handle;  fota\_init\_param\_t \*fota\_init\_param;  handle=fota\_init(&fota\_init\_param); |

### FOTA Perform

The fota\_perform() API parameter check\_for\_update==1 downloads the remote fota\_config.json file, compares the package version with the local fota\_config.json file, and only perform FOTA if the package version on the cloud is higher than the one present on device.

|  |
| --- |
| int  fota\_perform(fota\_handle\_t \*f\_handle, int check\_for\_update,  int flags)  {  fota\_files\_info\_t \*files\_info;  int ret = FOTA\_ERROR\_NONE;  int update\_available;  os\_printf("\n%s check\_for\_update = %d",  \_\_FUNCTION\_\_, check\_for\_update);  if(check\_for\_update == 1)  {  ret = fota\_update\_check(f\_handle, &update\_available);  if(ret){  os\_printf("\nError: fota\_update\_check");  return ret;  }  if(!update\_available){  os\_printf("\nError: No new update available");  return FOTA\_ERROR\_NO\_NEW\_UPDATE;  } |

The fota\_perform() API parameter check\_for\_update==2 downloads the remote fota\_config.json file and no check is performed.

|  |
| --- |
| else if(check\_for\_update == 2){  ret = fota\_config\_file\_download(f\_handle);  if(!ret){  return ret;  } |

Based on the type of the file files\_info->type, FOTA is performed as needed.

|  |
| --- |
| /\* Loop through the files list\*/  while(files\_info){  os\_printf(“\n type = %s”, files\_info->type);  if(!strcmp (files\_info->type, “configuration")){  files\_info = files\_info->next;  continue;  }else if(!strcmp(files\_info->type, "firmware")){  ret = fota\_firmware\_download(f\_handle, files\_info);  if(ret){  break;  }  }else if(!strcmp(files\_info->type, "file")){  ret = fota\_file\_download(f\_handle, files\_info);  if(!ret){  break;  }  }  files\_info = files\_info->next;  }  return ret;  } |

### FOTA Commit

After successful FOTA update, fota\_commit() is called to set the new firmware as the default.

This function will check if the fota\_config.json, part.json and boot.json file is updated successfully and resets the device.

|  |
| --- |
| fota\_commit(fota\_handle\_t \*f\_handle, int do\_reset)  {  int rval;  /\*Mark that fota was in progress\*/  os\_printf("\n%s", \_\_FUNCTION\_\_);  if(utils\_file\_touch(FOTA\_IN\_PROGRESS\_FILE\_PATH) < 0){  return -1;  }  if(FOTA\_ERROR\_NONE != (rval = fota\_update\_config\_file(f\_handle))){  os\_printf("\nError: updating config file failed");  return rval;  }  if(FOTA\_ERROR\_NONE != (rval = fota\_update\_part\_file(f\_handle))){  os\_printf("\nError: updating part file failed");  return rval;  }  if(FOTA\_ERROR\_NONE != (rval = fota\_set\_boot\_index(f\_handle))){  os\_printf("\nError: updating boot.json failed");  return rval;  }  /\*Fota is success\*/  unlink(FOTA\_IN\_PROGRESS\_FILE\_PATH);  os\_printf("\n\n\n\n");  /\*Reboot the device\*/  if(do\_reset == 1) {  reset\_device();  } |

### Deinitialize FOTA

This API releases all the resources allocated during fota\_init() and other FOTA API calls.

It frees up memory allocated for the new configuration file (f\_handle->recv\_buff).

Frees up application partition information used during the image download. (f\_handle->image\_info\_list).

|  |
| --- |
| void fota\_deinit(fota\_handle\_t \* f\_handle)  {  fota\_image\_info\_t \*img\_p, \*prev\_img;  fota\_files\_info\_t \*p, \*prev;  if(NULL == f\_handle)  return;  sector\_cache\_deinit();  os\_free(f\_handle->recv\_buff);  img\_p = f\_handle->image\_info\_list;  while(img\_p){  prev\_img = img\_p;  img\_p = img\_p->next;  os\_free(prev\_img);  } |

Frees up memory used to store the local fota\_config.json (f\_handle->cfg).

|  |
| --- |
| if(f\_handle->cfg){  p = f\_handle->cfg->files\_info\_list;  while(p){  prev = p;  p = p->next;  os\_free(prev);  }  os\_free(f\_handle->cfg); } |

Frees up memory used to store remote fota\_config.json (f\_handle->cfg\_remote).

|  |
| --- |
| if(f\_handle->cfg\_remote){  p = f\_handle->cfg\_remote->files\_info\_list;  while(p){  prev = p;  p = p->next;  os\_free(prev);  }  os\_free(f\_handle->cfg\_remote);  }  } |

Decrement the JSON reference count of part.json, fota\_cofig.json and remote fota\_config.json and free up FOTA handle.

|  |
| --- |
| json\_decref(f\_handle->json\_part);  json\_decref(f\_handle->json\_cfg);  json\_decref(f\_handle->json\_cfg\_remote);  /\* free f\_handle\*/  os\_free(f\_handle); |

# Block Diagram

Figure 5 block diagram represents memory layout before SSBL executes.

Figure 5: Flash layout

Boot Sector

SSBL

fota

Version:”1.0”

starting sector=32

starting sector=154

index=0

index=1

Partition sector for apps

starting sector=230

index=2

test\_app Version:”1.0”

test\_app Version:”0.0”

System FS

Root FS

SSBL will initially load FOTA application present at image index=0 of boot.json file as shown in Figure 6.

Figure 6: Before FOTA

SSBL

Boot Sector

fota

Version:”1.0”

starting sector=32

starting sector=154

index=0

index=1

starting sector=230

index=2

test\_app Version:”1.0”

test\_app Version:”0.0”

System FS

Root FS

FOTA app will download the test\_app.elf from the cloud based on the fota\_config.json package version comparison. The test\_app.elf on the cloud replaces test\_app.elf (version="0.0") at index=2, sector 230.

The boot.json gets updated to index=2. When the module gets reset, SSBL will boot the application at index=2.

Figure 7: After FOTA

SSBL

Boot Sector

fota

Version:”1.0”

starting sector=32

starting sector=154

index=0

index=1

starting sector=230

index=2

test\_app Version:”1.0”

test\_app Version:”2.1”

Root FS

System FS

# AWS Set-up

Amazon S3 bucket must be created to upload the objects such as ELF or fota\_config.json.

Refer user guide to create bucket: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/create-bucket-overview.html>

# Build and Run FOTA Application

Building and running of FOTA applications can be achieved in two ways:

## Using Script

With SDK directory as the current directory, execute the following command:

|  |
| --- |
| sdk$ python3 ./script/program\_flash.py -i apps/fota/bin/fota.elf -spt tools/partition\_files/ssbl\_part\_table.json |

where,

1. Mandatory arguments:

-i <elf\_path or elf folder>

(For example: *apps/fota*) in SDK or complete elf path (For example: *apps/fota/bin/fota.elf*)

1. Optional arguments:
   1. -spt <ssbl ptable>: provide the input path for ssbl\_partition\_table along with -spt in case the SSBL partition table is being considered.
   2. --no\_reset: provide the --no\_reset flag if there is no need to reset at the end. Please reset using the below command in case of this option

|  |
| --- |
| ./script/boot.py --device /dev/ttyUSB2 --reset=evk42 |

After successful programming, open miniterm at baud rate of 2457600 and reset the EVB either by using the above command or by pressing the reset button on the EVB.



Figure 8: Miniterm console output

**Note**:

1. The mentioned script also takes care of generating the root.img in the FOTA folder considering changes in fota/fs contents
2. Edit the part.json file and fota\_config.json file present in *sdk\_x.y/apps/fota/fs* if any configuration needs to be changed before issuing this command.

The SSID and passphrase of the Wi-Fi network needs to be updated in the part.json. Each time the above command is issued, it creates a new root fs image (root.img).

1. Ensure only one EVB is connected to the PC.

## Using Manual Programming Discrete Commands

### Build

|  |
| --- |
| sdk/apps/fota$  make |

Expected output:

Graphical user interface, text

Description automatically generated with medium confidence

Figure 9: Build fota.img file – Output

### Create Root Filesystem Image

|  |
| --- |
| sdk$ python3 ./script/build\_rootfs\_generic.py --folder\_path apps/fota/ |

### Flash Required Images

Execute the following instructions to flash the different components into Talaria TWO EVB under the SDK directory:

Load Flash Helper

|  |
| --- |
| ./script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl ./apps/gordon/bin/gordon.elf |

Invalidate the boot image

|  |
| --- |
| dd if=/dev/zero of=./empty.img bs=1K count=1  ./script/flash.py --device /dev/ttyUSB2 write 0x1000 ./empty.img |

Write Partition

|  |
| --- |
| ./script/flash.py --device /dev/ttyUSB2 from\_json ./tools/partition\_files/ssbl\_part\_table.json |

Download root fs image

|  |
| --- |
| ./script/flash.py --device /dev/ttyUSB2 write 0x180000 ./apps/fota/root.img |

Download SSBL

|  |
| --- |
| ./script/flash.py --device /dev/ttyUSB2 write 0x1000 ./apps/ssbl/fast\_ssbl.img |

Download fota.img

|  |
| --- |
| ./script/flash.py --device /dev/ttyUSB2 write 0x20000 ./apps/fota/out/fota.img |

Open a miniterm at baud rate of 2457600 and reset the EVB:



Figure 10: Miniterm console output

Reset the board either by giving the following command or by pressing the reset button on the EVB:

|  |
| --- |
| ./script/boot.py --device /dev/ttyUSB2 --reset=evk42 |

**Note**: Since GDB does not work with SSBL, it is not possible to use GDB for debugging as of now.

## Expected Output

On successful execution of the steps in section 12, reset the Talaria TWO EVB. The following observation is made:

1. Talaria TWO loads SSBL
2. SSBL loads FOTA test application
3. FOTA test application modifies files in the filesystem to trigger FOTA, then reboots
4. Talaria TWO reboots and loads SSBL, SSBL loads the FOTA application
5. FOTA application downloads and flashes application from server and reboots
6. Talaria TWO loads SSBL, SSBL loads the downloaded application

Console output:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PWAE  WWWWAE4 DWT comparators, range 0x8000  Build $Id: git-8bc43d639 $  4 DWT comparators, range 0x8000  Build $Id: git-8bc43d639 $  vm.flash\_location=0x00038700 sys.reset\_reason=1 passphrase=InnoQA2023$ ssid=Xiaomi\_Ax6000\_iop  Application Information:  ------------------------  Name : FOTA application  Version : 1.0  Build Date : Aug 15 2023  Build Time : 21:05:57  Heap Available: 314 KB (322448 Bytes)  [APP]data fs mounted  addr e0:69:3a:00:41:0c  network profile created for ssid: Xiaomi\_Ax6000\_iop  Connecting to added network : Xiaomi\_Ax6000\_iop  [1.450,212] CONNECT:d4:da:21:54:d3:c6 Channel:13 rssi:-41 dBm  wcm\_notify\_callback :WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_callback :CM\_NOTIFY\_MSG\_ADDRESS  [2.097,894] MYIP 192.168.31.211  [2.097,942] IPv6 [fe80::e269:3aff:fe00:410c]-link  wcm\_notify\_callback :\_NOTIFY\_MSG\_CONNECTED  Connected to added network : Xiaomi\_Ax6000\_iop  [APP]N/w Connection done..  fota\_json\_init: /data/fota\_config.json f = 0x000be2e0  Parsing rootfs FOTA config file\*\*\*  package\_version = 1.0  Package version = 1.0  type = configuration  name = fota.config  version, <null>  protocol, <null>  hostname = innotestota.s3.us-east-2.amazonaws.com  port = 443  secured = 2  uri = /fota\_config.json  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  type = firmware  name = test\_app  version, <null>  protocol, <null>  hostname = innotestota.s3.us-east-2.amazonaws.com  port = 443  secured = 2  uri = /test\_app.elf  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  Fota Init Success: bedf0  [APP]Perform Fota  fota\_perform check\_for\_update = 1  fota\_config\_file\_download 1078  fota\_http\_connect 688getting cert:/data/fota\_ca\_cert.pem  fota\_http\_connect:host=innotestota.s3.us-east-2.amazonaws.com port=443  Calling http\_client\_open()  . [SSL\_WRAP]Checking input configurations...  . [SSL\_WRAP]Seeding the random number generator...  . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1189  . [SSL\_WRAP]Connecting to tcp innotestota.s3.us-east-2.amazonaws.com:443...  . [SSL\_WRAP]Setting up the SSL/TLS structure...  . [SSL\_WRAP]setting configurations..  >auth mode = 2 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  fota\_config\_file\_download 1091  package\_version = 3.1  Package version = 3.1  type = configuration  name = fota.config  version, <null>  protocol, <null>  hostname = innotestota.s3.us-east-2.amazonaws.com  port = 443  secured = 2  uri = /fota\_config.json  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  type = firmware  name = test\_app  version, <null>  protocol, <null>  hostname = innotestota.s3.us-east-2.amazonaws.com  port = 443  secured = 2  uri = /test\_app.elf  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  utils\_num\_str\_cmp  3  1  1  0  deci1 = 3, fracn1 = 1, deci2 = 1, fracn2 = 0  Using the Remote config (Newly fetched) file  type = configuration  type = firmware  fota\_json\_init: /data/part.json f = 0x000bf778  Image array size = 3  name = fota  name = test\_app  version = 1.0  start\_sector = 154  name = test\_app  version = 0.0  start\_sector = 230  utils\_num\_str\_cmp  1  0  0  0  deci1 = 1, fracn1 = 0, deci2 = 0, fracn2 = 0  Selected index = 2  Download the new f/w @ sector = 230  fota\_http\_connect 688getting cert:/data/fota\_ca\_cert.pem  fota\_http\_connect:host=innotestota.s3.us-east-2.amazonaws.com port=443  Calling http\_client\_open()  . [SSL\_WRAP]Checking input configurations...  . [SSL\_WRAP]Seeding the random number generator...  . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1189  . [SSL\_WRAP]Connecting to tcp innotestota.s3.us-east-2.amazonaws.com:443...  . [SSL\_WRAP]Setting up the SSL/TLS structure...  . [SSL\_WRAP]setting configurations..  >auth mode = 2 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  All data received  Fw download complete  next index = 2  fota\_commit  utils\_num\_str\_add  0  0  2  0  deci1 = 0, fracn1 = 0, deci2 = 2, fracn2 = 0  utils\_num\_str\_add : out\_str = 2.0  fota\_update\_part\_file: !!!Updated new version = 2.0  fota\_json\_init: /data/boot.json f = 0x000a8908  Setting next boot index = 2  Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PWAE  WWWWAE4 DWT comparators, range 0x8000  Build $Id: git-8bc43d639 $  4 DWT comparators, range 0x8000  Build $Id: git-13f33b8d7 $  vm.flash\_location=0x000f0300 sys.reset\_reason=4 passphrase=InnoQA2023$ ssid=Xiaomi\_Ax6000\_iop  Hello World |

In case of any error, due to network failure or Wi-Fi disconnection, the program will exit. Upon resetting the EVB by pressing the reset button, FOTA application will be loaded again and the firmware upgrade will be tried again.

# Build and Run FOTA Application with Secureboot

FOTA application can be built with secureboot.

For more details on secureboot mode, refer Application\_for\_using\_SSBL.pdf (*sdk\_x.y\_alpha\apps\ssbl\doc*).

**Note**:

1. x and y in sdk\_x.y refer to the SDK package release version.
2. Enabling secureboot enables the use of encrypted files.

## Flashing and Testing

1. Enroll keys & flash SSBL components in secureboot mode (refer steps 1 to 4 of section: 7.2.2 in Application for\_using\_SSBL.pdf (*sdk\_x.y\_alpha\apps\ssbl\doc)*
2. Build the filesystem (root\_secure.img)

|  |
| --- |
| cd <sdk>  python ./script/build\_rootfs\_generic.py --folder\_path apps/fota --secure True --keyfile ./apps/ssbl/enroll.json |

1. Create signed and encrypted ELF (fota.elf.enc)

|  |
| --- |
| cd <sdk>/apps/fota  make clean  make SECUREBOOT=1 KEY=<sdk>/apps/ssbl/enroll.json |

1. Flash application at 0x20000

|  |
| --- |
| cd <sdk>$  ./script/flash.py write 0x20000 <sdk>/apps/fota/out/fota.elf.enc |

1. Flash the filesystem

|  |
| --- |
| cd <sdk>$  ./script/flash.py write 0x180000 <sdk>/apps/fota/root\_secure.img |

## Expected Output

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWAE  FIRST:SWWWWAHE  Si Build $Id: git-a042e9a42 $  Warning! Make sure to remove this code section once in production  secureboot\_secret:  6cd7d2c0c1f5820b83a69b0c1bb961a3a01502ae21b198236a3013b3456bc661ac000000  Warning! Make sure to remove this code section once in production  cipher key: 4e3b0b9792183c53ecc78a38c64a45c071b97bc40b0baba308ed76db8a46cef1  public key: 20b003d2f297be2c5e2c83a7e9f9a5b9eff49111acf4fddbcc0301480e359de6dc809c49652aeb6d63329abf5a52155c766345c28fed3024741c8ed01589d28b  Build $Id: git-a042e9a42 $  vm.flash\_location=0x00037700 sys.reset\_reason=1 passphrase=1234567890 ssid=innotest  Application Information:  Name       : FOTA application  Version    : 1.0  Build Date : Apr 20 2023  Build Time : 07:10:32  Heap Available: 240 KB (246680 Bytes)  [APP]root fs mounted  [1.716,504] rfdrv: unknown module type (0)  addr f8:e9:43:c6:08:ef  network profile created for ssid: innotest  Connecting to added network : innotest  [2.768,898] CONNECT:60:32:b1:33:b5:7b Channel:11 rssi:-38 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [7.508,564] MYIP 192.168.1.16  [7.508,612] IPv6 [fe80::fae9:43ff:fec6:8ef]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Connected to added network : innotest  [APP]N/w Connection done..  Parsing rootfs FOTA config file\*\*\*  package\_version = 1.0  Package version = 1.0  type = configuration  name = fota.config  version, <null>  protocol, <null>  hostname = innosecuredfota.s3.amazonaws.com  port = 443  secured = 2  uri = /fota\_config.json  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  type = firmware  name = wifi\_scan  version, <null>  protocol, <null>  hostname = innosecuredfota.s3.amazonaws.com  port = 443  secured = 2  uri = /wifi\_scan.elf.enc  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  Fota Init Success: b1728  [APP]Perform Fota  fota\_perform check\_for\_update = 1  fota\_config\_file\_download 1078  fota\_http\_connect 688getting cert:/data/fota\_ca\_cert.pem  fota\_http\_connect:host=innosecuredfota.s3.amazonaws.com port=443  Calling http\_client\_open()    . [SSL\_WRAP]Checking input configurations...    . [SSL\_WRAP]Seeding the random number generator...    . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1201    . [SSL\_WRAP]Connecting to tcp innosecuredfota.s3.amazonaws.com:443...    . [SSL\_WRAP]Setting up the SSL/TLS structure...    . [SSL\_WRAP]setting configurations..  >auth mode = 2 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  fota\_config\_file\_download 1091  package\_version = 2.0  Package version = 2.0  type = configuration  name = fota.config  version, <null>  protocol, <null>  hostname = innosecuredfota.s3.amazonaws.com  port = 443  secured = 2  uri = /fota\_config.json  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  type = firmware  name = wifi\_scan  version, <null>  protocol, <null>  hostname = innosecuredfota.s3.amazonaws.com  port = 443  secured = 2  uri = /wifi\_scan.elf.enc  url, <null>  hash, <null>  ca\_cert = /data/fota\_ca\_cert.pem  utils\_num\_str\_cmp  2  0  1  0  deci1 = 2, fracn1 = 0, deci2 = 1, fracn2 = 0  Using the Remote config (Newly fetched) file  type = configuration  type = firmware  Image array size = 3  name = fota  name = wifi\_scan  version = 1.0  start\_sector = 200  name = wifi\_scan  version = 2.0  start\_sector = 300  utils\_num\_str\_cmp  1  0  2  0  deci1 = 1, fracn1 = 0, deci2 = 2, fracn2 = 0  Selected index = 1  Download the new f/w @ sector = 200  fota\_http\_connect 688getting cert:/data/fota\_ca\_cert.pem  fota\_http\_connect:host=innosecuredfota.s3.amazonaws.com port=443  Calling http\_client\_open()    . [SSL\_WRAP]Checking input configurations...    . [SSL\_WRAP]Seeding the random number generator...    . [SSL\_WRAP]Loading the CA root certificate ...Cert Len = 1201    . [SSL\_WRAP]Connecting to tcp innosecuredfota.s3.amazonaws.com:443...    . [SSL\_WRAP]Setting up the SSL/TLS structure...    . [SSL\_WRAP]setting configurations..  >auth mode = 2 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  All data received  Fw download complete  next index = 1  fota\_commit  utils\_num\_str\_add  1  0  2  0  deci1 = 1, fracn1 = 0, deci2 = 2, fracn2 = 0  utils\_num\_str\_add : out\_str = 3.0  fota\_update\_part\_file: !!!Updated new version = 3.0  Setting next boot index = 1  Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWAE  FIRST:SWWWWAHE  Si Build $Id: git-a042e9a42 $  Warning! Make sure to remove this code section once in production  secureboot\_secret:  6cd7d2c0c1f5820b83a69b0c1bb961a3a01502ae21b198236a3013b3456bc661ac000000  Warning! Make sure to remove this code section once in production  cipher key: 4e3b0b9792183c53ecc78a38c64a45c071b97bc40b0baba308ed76db8a46cef1  public key: 20b003d2f297be2c5e2c83a7e9f9a5b9eff49111acf4fddbcc0301480e359de6dc809c49652aeb6d63329abf5a52155c766345c28fed3024741c8ed01589d28b  4 DWT comparators, range 0x8000  Build $Id: git-ccecb0473 $  vm.flash\_location=0x000dd400 sys.reset\_reason=4 passphrase=1234567890 ssid=innotest  SDK Ver: SDK\_2.6.3master  Wifi Scan Demo App  [1.814,917] rfdrv: unknown module type (0)  addr f8:e9:43:c6:08:ef  Scan parameters:      channel\_masks: 255 255 255 255 255 255 255 255      bssid: 0xFFFFFFFFFFFF      txrate: 0      waittime: 0      ie list: 0x  Found 9 nets:  04:42:1a:bd:6e:08 on channel  1 @ -38 'asusax55u\_iop' 'WPA2/WPA3-Enterprise+MFPR'  74:da:88:a6:9c:ea on channel  1 @ -39 'low\_rssi' 'WPA2-PSK'  58:11:22:71:ee:10 on channel  1 @ -40 'ASUS\_Outside' 'WPA2-PSK'  38:6b:1c:c0:da:38 on channel 11 @ -42 'connect\_Idle\_stability' 'WPA-PSK/WPA2-PSK Mixed Mode'  d4:5d:64:d9:5c:a0 on channel 1 @ -45 'test\_shetty' 'WPA2-PSK+MFPR'  24:4b:fe:5e:fd:d8 on channel 1 @ -50 'Asus\_86U\_2G\_iop' 'WPA2-PSK'  62:d4:f7:bf:fe:47 on channel 10 @ -52 '' 'WPA2-PSK'  98:da:c4:d5:89:9b on channel 11 @ -58 'TP-Link\_BLE' 'WPA2-PSK'  b0:39:56:93:83:31 on channel 6 @ -62 'innotest\_open123' 'OPEN' |