# Use Cases

## Linux Platform

### Reset Device

1. **Normal Device Reset**

Reset the device.

**Command syntax**:

|  |
| --- |
| ./script/boot.py [device] **--reset=evk42** |

A screenshot of a computer

Description automatically generated with medium confidence

Figure 16: Normal device reset - terminal output

**Console output**:

|  |
| --- |
| $ miniterm /dev/ttyUSB3 2457600 |

Text

Description automatically generated

Figure 17: Normal device reset - console output

In this example, the app HelloWorld is stored in device flash. Hence a flash-boot from reset will display the Hello World.

1. **Device Reset with Inhibiting-flash-boot (Boot loader mode)**

Reset the device with the inhibiting-flash-boot feature.

If the device has app code stored in flash from which the device will boot, this special reset command is required which will allow flashing of a new app code.

**Command syntax**:

|  |
| --- |
| ./script/boot.py [device] **--reset=evk42\_bl** |

Text

Description automatically generated

Figure 18: inhibiting-flash-boot - terminal output

This reset will trigger Talaria TWO to abort from device port link (expected).

Text

Description automatically generated

Figure 19: Abort from device port link

Launch miniterm to resume the device port connection.

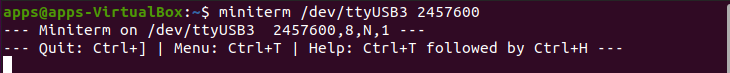


Figure 20: Resume device port connection

1. **Reset with Inhibiting-flash-boot with booting Gordon**

Reset the device with inhibiting-flash-boot and boot the gordon app.

This combination of commands is for the host to obtain access to the storage on Talaria TWO and operate read/write data.

Command syntax:

|  |
| --- |
| ./script/boot.py [device] **--reset=evk42\_bl** apps/gordon/bin/gordon.elf |

Text

Description automatically generated

Figure 21: Inhibiting-flash-boot with booting Gordon - terminal output

### Program App in Memory

Program app in Talaria TWO memory.

**Command syntax**:

|  |
| --- |
| ./script/boot.py [device] **--reset=evk42** <path>/app.elf [boot\_args list] |

Text

Description automatically generated

Figure 22: Programming app in Talaria TWO memory - terminal output

**Console output:**

Text

Description automatically generated

Figure 23: Programming app in Talaria TWO memory - console output

### Erase Image or Program New Image in Flash

1. Erase Image in Flash

The following command erases the app code in flash:

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Flash empty.img

**Command syntax**:

|  |
| --- |
| ./script/flash.py [device] **write 0x1000 <path>/empty.img** |



Figure 24: Flash empty.img

Button reset to confirm the erase operation is a success.

Text

Description automatically generated

Figure 25: Erase operation success

1. Program New Image in Flash

**Command syntax**:

|  |
| --- |
| ./script/boot.py [device] **--reset=evk42\_bl** **--flash=all** <path>/app.elf [boot\_args\_list] |

Text

Description automatically generated

Figure 26: Program new image to flash

Button reset to confirm the flash operation a success.

Text

Description automatically generated

Figure 27: Flash operation success

### Storage Data & File: Read/Write

1. **Read Partition Table from Device (to\_json)**

Reading file from device involves storage access. Hence, running of Gordon on the device is necessary.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the flash.py with the command: to\_json

**Command syntax**:

|  |
| --- |
| ./script/flash.py [device] **to\_json** <user\_specify\_path>/<user\_spefcify\_name>.json |

Text

Description automatically generated

Figure 28: Inhibiting-flash-boot and load gordon.elf

Display of the board\_356.json file

Text, timeline

Description automatically generated

Figure 29: Display of the board\_356.json file

1. **Write Partition Table in Device (from\_json)**

Writing file to device involves storage access. Hence, running of Gordon on the device is necessary.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the flash.py with the command: from\_json

**Command syntax**:

|  |
| --- |
| ./script/boot.py [device] **--reset=evk42\_bl** <path>/gordon.elf  ./script/flash.py [device] **from\_json** <user\_specify\_path>/<user\_spefcify\_name>.json |

Text

Description automatically generated

Figure 30: inhibiting-flash-boot and load gordon.elf - terminal output

**Console output**:

Text

Description automatically generated

Figure 31: flash.py - from\_json

1. **Show File System Contents**

Display the file system contents.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: tree

**Command syntax**:

|  |
| --- |
| ./script/storage.py **tree** |

Text

Description automatically generated

Figure 32: Show file system contents

1. **Read File and Store to Location on Host**

Read file from the file system and store it in host.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: read

**Command syntax**:

|  |
| --- |
| ./script/storage.py **read** <path\_read>/<file\_name\_read> **--output** <path\_write>/<file\_name\_write> |

Chart, text

Description automatically generated

Figure 33: Inhibiting-flash-boot and load gordon.elf – terminal output

The file is created:

Shape, rectangle

Description automatically generated with medium confidence

Figure 34: File creation

1. **Write File to Device**

Write file from host to the device in file system.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: write

**Command syntax**:

|  |
| --- |
| ./script/storage.py **write** <path\_read>/<file\_name\_read> <path\_write>/file\_name\_write> |



Figure 35: Running storage.py - write

Use ./script/storage.py tree command to check the new file on Talaria TWO file system:

Text

Description automatically generated

Figure 36: Check new file on Talaria TWO file system

Example: Load certificates

1. $ ./script/storage.py write rootCACert.pem /data/ca.pem
2. $ ./script/storage.py write client\_cert.pem /data/client.pem
3. $ ./script/storage.py write client\_key.pem /data/client.key

### Network Profile

The Wi-Fi network profile stores the configuration of the Wi-Fi network with parameters such as SSID, passphrase, BSSID, security type etc. Network profile can be created from the JSON data.

JSON data can be stored in either:

1. The filesystem: Build a network profile from a file in Talaria TWO filesystem given by the path in Talaria TWO filesystem

or

1. A string: JSON string which describes and parses the network profile

Characteristics of creating JSON network profile:

1. All network profiles MUST start with { and end with }.
2. The network profile contents use the form ("key": "value") pairs.
3. All the ("key": "value") pairs MUST be followed by a comma (,) EXCEPT the last row.
4. There is NO COMMA at the end of the network profile.
5. Mandatory fields MUST always be present, otherwise, parsing the network profile will fail.
6. If a passphrase contains double quotes, for example: "my\_secret", a backslash MUST preceded the double quotes, i.e., "passphrase": "\"my\_secret\"".

**Note**:

To avoid recomputing the PMK for each connection (WPA/WPA2-PSK), it can be stored in Talaria TWO’s flash. This can be done by a pmk\_path key and the location of PMK file in Talaria TWO flash. There are two new keys to make it easier in creating network profiles:

1. security\_type key which replaces the old np\_type in network profiles and can have the following values depending the type of the network:
   1. open
   2. wpa\_psk or wpa2\_psk
   3. wpa3\_psk
   4. wpa23\_psk
   5. enterprise
2. auth\_type key which can ONLY be used when configuring enterprise network profiles and describes the authentication protocol. Values for auth\_type can be:
   1. eap\_psk
   2. eap\_tls
   3. eap\_peap

Following are some examples of JSON data specifying different network profiles:

1. **Open security**:

|  |
| --- |
| {  “security\_type”: “open”  “ssid”: “my\_ap\_ssid”  } |

Profile fields:

* 1. Mandatory – security\_type, ssid
  2. Optional – bssid

1. **WPA/WPA2-PSK**:

|  |
| --- |
| {  "security\_type": "wpa23\_psk",  "ssid": "my\_wpa2\_ap\_ssid",  "passphrase": "my\_wpa2\_passphrase"  “pmk\_path”: “sys/pmkey\_fname.data”  } |

Profile fields:

* 1. Mandatory – security\_type, ssid, passphrase
  2. Optional – bssid, pmk\_path

1. **WPA/WPA2-PSK** specifying one specific AP among several with same SSID. BSSID is the APs mac address:

|  |
| --- |
| {  "security\_type": "wpa2\_psk",  "ssid": "my\_wpa2\_ap\_ssid",  "passphrase": "my\_wpa2\_passphrase",  "bssid": "my\_ap\_bssid"  “pmk\_path”: “/data/pmkey.data”  } |

Profile fields:

* 1. Mandatory – security\_type, ssid, passphrase
  2. Optional – bssid, pmk\_path

1. **WPA3-PSK (SAE)** only:

|  |
| --- |
| {  "security\_type": "wpa3\_psk",  "ssid": "my\_wpa3\_ap\_ssid",  "sae\_password": "my\_sae\_password"  } |

Profile fields

* 1. Mandatory: security\_type, ssid, sae\_password
  2. Optional : sae\_password\_id, bssid

1. **WPA/WPA2-PSK and WPA3-PSK.** If WPA3 is available, it will be given a higher priority:

|  |
| --- |
| {  "security\_type": "wpa23\_psk",  "ssid": "my\_wpa2\_wpa3\_ap\_ssid",  "sae\_password": "my\_sae\_password"  "passphrase": "my\_wpa2\_passphrase"  } |

Profile fields

* 1. Mandatory: security\_type, ssid, sae\_password, passphrase
  2. Optional : bssid

1. **EAP-PSK** Enterprise security:

|  |
| --- |
| {  "security\_type": "enterprise",  "auth\_type": "eap\_psk",  "ssid": "my\_eap\_psk\_ap\_ssid",  "passphrase": "my\_eap\_psk\_passphrase",  "identity": "my\_eap\_identity"  }  Profile fields  Mandatory: security\_type, auth\_type, ssid, passphrase, identity  Optional : bssid } |

1. **EAP-TLS** Enterprise security.

**Note**: Storing the certificate and key files on flash is not part of the network profile functionality.

|  |
| --- |
| {  "security\_type": "enterprise",  "auth\_type": "eap\_tls",  "ssid": "my\_eap\_tls\_ap\_ssid",  "identity": "my\_eap\_tls\_identity",  "ca\_path": "/data/ca.pem",  "cert\_path": "/data/client.pem",  "pkey\_path": "/data/client.key",  "pkey\_pwd": "my\_key\_password"  }  Profile fields  Mandatory: security\_type, auth\_type, ssid, identity, ca\_path, cert\_path, pkey\_path  Optional : bssid, pkey\_pwd, dh\_path, server\_hostname |

1. **EAP-PEAP/MSCHAPv2** Enterprise security:

|  |
| --- |
| {  "security\_type": "enterprise",  "auth\_type": "eap\_peap",  "ssid": "my\_eap\_peap-mschapv2\_ssid",  "identity": "my\_eap\_peap\_identity",  "ca\_path": "/data/ca.pem",  "identity2": "eap-peap",  "password": "password",  "phase2auth":"MSCHAPv2"  } |

Profile fields

* 1. Mandatory: security\_type, auth\_type, ssid, identity, ca\_path, identity2, password, phase2auth
  2. Optional : bssid, server\_hostname.

**Note**: For "phase2auth", the key MUST always be "phase2auth": while the VALUE can be, For example: "MSCHAPv2", "mschapv2" or "MsChAPv2" or any other similar lower/upper case letter combination.

**Prerequisites:**

To upload network profile into Talaria TWO file system, running of Gordon on the device is necessary.

1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
2. Write network profile JSON to Talaria TWO

**Command syntax:**

|  |
| --- |
| python ./script/storage.py [device] write <user\_specify\_path>/<user\_spefcify\_name>.json /data/<user\_specify\_name>.json |

**Note**: It is a single line command.

**Steps:**

1. Create a Network Profile File(JSON file).

Create a file and name anywhere in your PC and name it **nprofile.json** or **<file\_name>.json**, and copy the following contents (we assume we are connecting to a WPA2-PSK network):

|  |
| --- |
| {  "security\_type":"wpa23\_psk",  "ssid": "My-AP-Name",  "passphrase": "0CD7AC607B"  }  nprofile.json file: |

1. To upload our network profile into Talaria file system, start **gordon.elf** and reset in bootloader mode.

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

Text

Description automatically generated

Figure 37: Starting gordon.elf

1. Use **storage.py** to write the network config file (**nprofile.json**) into Talaria TWO file system in the partition /data.

|  |
| --- |
| python ./script/storage.py write nprofile.json /data/nprofile.json |

Command Log:

Text

Description automatically generated

Figure 38: Writing network config file

Console Log:

Text

Description automatically generated

Figure 39: Writing network config file – console output

1. Program(RAM/FLASH) app to connect to the Access Point configured in **nprofile.json** as:
   1. Program RAM:

|  |
| --- |
| python ./script/boot.py --reset=evk42 examples/using\_wifi/bin/wifi\_connect.elf np\_conf\_path=/data/nprofile.json |

Command Log:

Text

Description automatically generated

Figure 40: Program RAM to connect to AP

Console Log:

A screenshot of a computer

Description automatically generated

Figure 41: Program RAM - Console output

* 1. Program FLASH:

|  |
| --- |
| python ./script/boot.py --reset=evk42 –flash=all examples/using\_wifi/bin/wifi\_connect.elf np\_conf\_path=/data/nprofile.json |

Command Log:

A black background with white text

Description automatically generated with medium confidence

Figure 42: Program Flash to connect to AP

Console Log:

A screenshot of a computer program

Description automatically generated

Figure 43: Program Flash - Console output

## Windows Platform – libusbK Installed

### Reset Device

1. **Device Reset**

Reset the device.

**Command syntax**:

|  |
| --- |
| python script\boot.py [device] **--reset=evk42** |

Text

Description automatically generated

Figure 44: Device reset

**Console output**:

Text

Description automatically generated

Figure 45: Device reset - console output

1. **Device Reset with Inhibiting-flash-boot**

Reset the device with the inhibiting-flash-boot feature.

If the device has app code stored in flash from which the device will boot, this special reset command is required which will allow to flash a new app code.

**Command syntax**:

|  |
| --- |
| python script\boot.py [device] **--reset=evk42\_bl** |

Text

Description automatically generated

Figure 46: Device Reset with Inhibiting-flash-boot

**Console output**:

Table

Description automatically generated with medium confidence

Figure 47: Device Reset with Inhibiting-flash-boot - console output

The device has been put into the programming mode (inhibiting-flash-boot).

1. **Reset with Inhibiting-flash-boot and booting Gordon**

Reset the device with inhibiting-flash-boot and boot Gordon app.

This combination of commands is for the host to obtain access to the storage on Talaria TWO and operate read/write data.

**Command syntax:**

|  |
| --- |
| python script\boot.py [device] **--reset=evk42\_bl** apps\gordon\bin\gordon.elf |

Text

Description automatically generated

Figure 48: Reset with Inhibiting-flash-boot with booting Gordon

**Console output:**

A picture containing graphical user interface

Description automatically generated

Figure 49: Reset with Inhibiting-flash-boot with booting Gordon - console output

### Program App in Memory

Program app in Talaria TWO memory, assuming Talaria TWO has no code stored in flash to boot from.

**Command syntax**:

|  |
| --- |
| python script\boot.py [device] **--reset=evk42** <path>\app.elf [boot\_args\_list] |

Text

Description automatically generated

Figure 50: Program app in Talaria TWO memory

**Console output**:

Text

Description automatically generated

Figure 51: Program app in Talaria TWO memory - console output

### Erase Image or Program New Image in Flash

**Erase Image in Flash**

The following command erases the app code in flash:

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done so)
  2. Flash empty.img

**Command syntax**:

|  |
| --- |
| python script\flash.py [device] **write 0x1000 <path>\empty.img** |

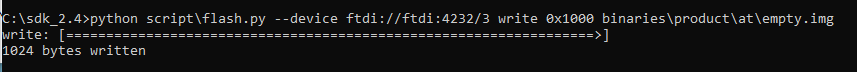


Figure 52: Erase image in flash

**Console output**:

Text

Description automatically generated

Figure 53: Erase image in flash - console output

**Program New Image in Flash**

**Command syntax**:

|  |
| --- |
| python script\boot.py [device] **--reset=evk42\_bl** **--flash=all** <path>\app.elf [boot\_args\_list] |

Text

Description automatically generated

Figure 54: Program new image in flash

**Console output**:

Text

Description automatically generated with medium confidence

Figure 55: Program new image in flash - console output

### Storage Data & File: Read/Write

1. **Read Partition Table from Device (to\_json)**

Reading file from device involves storage access. Hence, the running of Gordon on the device is necessary.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the flash.py with the command: to\_json

**Command syntax:**

|  |
| --- |
| python script\flash.py [device] **to\_json** <user\_specify\_path>\<user\_spefcify\_name>.json |

Text

Description automatically generated

Figure 56: Read Partition Table from device

Display of the board\_0183.json file

Text, timeline

Description automatically generated

Figure 57: Display of the board\_0183.json file

1. **Write Partition Table in Device (from\_json)**

Writing file to device involves storage access. Hence, running of Gordon on the device is necessary.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the flash.py with the command: from\_json

**Command syntax:**

|  |
| --- |
| python script\flash.py [device] **from\_json** <user\_specify\_path>/<user\_spefcify\_name>.json |

Text

Description automatically generated

Figure 58: Write Partition Table in device

1. **Show File System Contents**

Display the file system contents.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: tree

**Command syntax**:

|  |
| --- |
| python script\storage.py [device] **tree** |

Text

Description automatically generated

Figure 59: Show file system contents

1. **Read File and Store to Location on Host**

Read file from the file system and store it in host.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: read

**Command syntax**:

|  |
| --- |
| python script\storage.py [device] **read** <path\_read>\<file\_name\_read> **--output** <path\_write>/<file\_name\_write> |

**Note**: Single line command.

Text

Description automatically generated

Figure 60: Read file and store to location on host

The file is created as per user entry:



Figure 61: File creation

1. **Write File to Device**

Write file from host to the device in file system.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: write

**Command syntax:**

|  |
| --- |
| python script\storage.py [device] **write** <path\_read>\<file\_name\_read> <path\_write>\file\_name\_write> |

**Note**: A single line command.

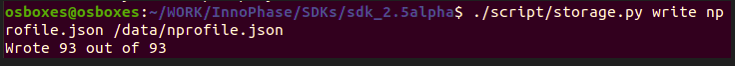


Figure 62: Write file to device

Use the ./script/storage.py tree command to check the new file on Talaria TWO file system:

Text

Description automatically generated

Figure 63: Check new file on Talaria TWO file system

## Windows Platform – With COM Port

To retrieve the COM ports on the device, the libusbK drivers need to be uninstalled.

For more details on updating driver from libusbK to COM port, refer section: Update Driver from libusbK to COM Port of document UG\_Download\_Tool.pdf (*sdk\_x.y\pc\_tools\Download\_Tool\doc*).

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

### Reset Device

All command resets are not supported under this setup.

Depending on the requirement to reset the device manually, execute the following steps:

1. Reset the device manually:
   1. Press the RESET button on Talaria TWO EVB
2. Inhibiting-flash-boot (to re-enable programming):
   1. Short GPIO17 to GND
   2. Press the RESET button on EVB
   3. Un-short GPIO17 and GND
   4. Load gordon.elf from host, using boot.py

### Load Gordon

Load Gordon app to run the flash.py from host, assuming the image in device flash has been erased or has been placed in the inhibiting-flash-boot state.

**Command syntax:**

|  |
| --- |
| python script\boot.py [device] apps\gordon\bin\gordon.elf |

Text

Description automatically generated with medium confidence

Figure 64: Load gordon.elf

Console output shows the inhibiting-flash-boot in effect and Gordon being ready.

Text

Description automatically generated

Figure 65: inhibiting-flash-boot in effect

### Erase Image

The following command erases the app code in flash.

1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done so)
2. Flash empty.img

**Command syntax**:

|  |
| --- |
| python script\flash.py [device] **write 0x1000 <path>\empty.img** |

A picture containing timeline

Description automatically generated

Figure 66: Erase image

Button reset to confirm the erase operation is a success.

Text

Description automatically generated

Figure 67: Erase operation – success

### Storage Data & File: Read/Write

1. **Read Partition Table from Device (to\_json)**

Reading file from device involves storage access. Hence, running of Gordon on the device is necessary.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the flash.py with the command: to\_json

**Command syntax**:

|  |
| --- |
| python script\flash.py [device] **to\_json** <user\_specify\_path>\<user\_spefcify\_name>.json |

Text

Description automatically generated

Figure 68: Read Partition Table from device

Display of the board\_0407.json file

Text, timeline

Description automatically generated

Figure 69: Display of the board\_0407.json file

1. **Write Partition Table in Device (from\_json)**

Writing file to device involves storage access. Hence, running of Gordon on the device is necessary.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the flash.py with the command: from\_json

**Command syntax**:

|  |
| --- |
| python script\flash.py [device] **from\_json** <user\_specify\_path>/<user\_spefcify\_name>.json |

Text

Description automatically generated

Figure 70: Write Partition Table in device

**Console output**:

Text

Description automatically generated

Figure 71: Write Partition Table in device - console output

1. **Show File System Contents**

Display the file system contents.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: tree

**Command syntax**:

|  |
| --- |
| python script\storage.py [device] **tree** |

Text

Description automatically generated

Figure 72: Show file system contents

1. **Read File and Store to Location on Host**

Read file from the file system and store it in host.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: read

**Command syntax:**

|  |
| --- |
| python script\storage.py [device] **read** <path\_read>\<file\_name\_read> **--output** <path\_write>/<file\_name\_write> |

**Note**: A single line command.

Text

Description automatically generated

Figure 73: Read file and store to location on host

The file is created per user entry:



Figure 74: File creation

1. **Write File to Device**

Write file from the host to the device in file system.

* 1. Reset device with inhibiting-flash-boot and load gordon.elf (if not done already)
  2. Run the storage.py with the command: write

**Command syntax**:

|  |
| --- |
| python script\storage.py [device] **write** <path\_read>\<file\_name\_read> <path\_write>\file\_name\_write> |

**Note**: A single line command.

Text

Description automatically generated

Figure 75: Write file to device

Use the ./script/storage.py tree command to check the new file on Talaria TWO file system:



Figure 76: Check new file on Talaria TWO filesystem

## Write New App to Device with Factory Loaded Image

Assume a Talaria TWO module is preloaded loaded with factory-loaded app like the helloworld app as shown in Figure 77.

Text

Description automatically generated

Figure 77: Talaria TWO preloaded with factory-loaded app

### Sequence of CLI Commands

1. **Reset with inhibiting-flash-boot and load Gordon**

**Command Syntax:**

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

**Console Output**:

Text

Description automatically generated

Figure 78: Reset with inhibiting-flash-boot and load gordon

1. **Write the Partition Table**

Write the Partition Table which is a standard configuration, standard\_part\_table.json, to Talaria TWO device:

**Command Syntax**:

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

**Note**: A single line command.

**Console Output**:



Figure 79: Write partition table - console output

1. **Flash New App to Device**

Write a new application, iperf3 with bootargs, to the device.

**Command Syntax**:

|  |
| --- |
| ./script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl --flash=all ./apps/iperf3/bin/iperf3.elf ssid=TP-LINK\_Guest\_864F passphrase=test13579 |

Text

Description automatically generated

Figure 80: Flash new app to device

Re-launch the console terminal app and reset Talaria TWO device.

**Console output**:

Text

Description automatically generated

Figure 81: Flash new app to device - console output

Figure 81 shows that the new app has been written to Talaria TWO’s flash and runs as expected.

## Flash New App to Device with an Existing App

Write the new app, stw.elf, to flash.

Command Syntax:

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

Text

Description automatically generated

Figure 82: Write new app to flash

Re-launch the console terminal app and reset Talaria TWO device.

**Console Output**:

Text

Description automatically generated

Figure 83: Write new app to flash - console output

Figure 83 shows that the new app, STW, has been written to Talaria TWO’s flash and runs as expected.