This section serves as a reference guide for Talaria TWO CLI commands, its usage, use case or examples in different scenarios supported by Talaria TWO modules.

# Introduction

The subsequent sections introduce the use of commands in the CLI environment to access the Talaria TWO (T2) module and configure the device to perform various development, automation and testing tasks.

These commands run Python scripts from the host machine (Windows or Linux), in conjunction with the application software that are loaded to and are running in the Talaria TWO device. The script files and the Talaria TWO reference/example applications are provided in the Talaria TWO SDK release package in the following folders:

1. freertos\_sdk/apps: Talaria TWO applications that are the core and basic release applications.
2. freertos\_sdk/examples: Collection of Talaria TWO software apps along with components for Talaria TWO software development. They are organized in various technology areas and implementation examples, together with the projects and make files.
3. freertos\_sdk/script: Python scripts/utilities to configure and program the device.
4. Other folders.

# Primary Scripts for CLI Operations

Written in the Python language, three script files for executing CLI operations are provided in the SDK for the user’s convenience that can be used for evaluation, testing, and development of Talaria TWO based systems.

1. sdk/script/boot.py
2. sdk/script/flash.py
3. sdk/script/storage.py

Based on the arguments provided command line, the scripts performs various functions.

## boot.py

Following are the basic functionalities of boot.py:

1. Load app to memory (RAM) on Talaria TWO module:

This script reads the application (in ELF format) provided in command line, along with the arguments to load the app to Talaria TWO in its memory, via the host and device interface. USB is used only in the evaluation, development or testing phase.

1. Write app to flash on Talaria TWO module:

This script reads the application (in ELF format) provided in command line, along with the arguments to write the app to Talaria TWO in its flash, with the default handling by the script, via the host and device interface. USB is used only in the evaluation, development or testing phase.

1. Create .img file from ELF file

This script reads the name of the app file (in ELF format) provided in command line to generate the .img file for flash download.

**Note**: Refer section [Using boot.py](#_Using_boot.py) for more details.

## flash.py

Script to read and write to the flash, which requires gordon.elf application.

This section provides details on script usage and prerequisites.

**Note**: Refer section [Using flash.py](#_Using_flash.py) for more details.

**Prerequisite/precondition to run flash.py**:

1. Program Talaria TWO with the Gordon app (gordon.elf)
2. gordon.elf application is available from the following folder: *freertos\_sdk\_x.y/apps/gordon.elf*.
3. Gordon is an application that enables access to Talaria TWO's flash over Host I/O, including partitioning and filesystem access and raw reads/writes to flash.
4. Boot.py script is used to program the gordon.elf onto Talaria TWO. Execute the following command to accomplish the same:

|  |
| --- |
| ./script/boot.py [device] <path>/gordon.elf |

**Load app .img file to flash on Talaria TWO module:**

This script reads the name of the app file (in .img format) provided in command line, along with the arguments to write the app to Talaria TWO in its flash with the write command and user-specified location (in hex), via the host and device interface (USB/Serial).

Create image file (.img) from the ELF file of an application:

|  |
| --- |
| ./script/boot.py –output <output\_path>/app.img <elf\_path>/app.elf |

**Read/Write Partition Table on module:**

This script reads from/writes the partition table on the device (in the json file).

## storage.py

Storage file access utility. Following are the basic file system access functionalities of storage.py:

1. Display the file system contents
2. Read file from the device to store in the location on host
3. Write file from the host to the device
4. Write the network config file (**.json file**) into Talaria TWO file system to configure L2 parameters such as SSID, passphrase, BSSID, security type and certificates for EAP authentication

Refer section [Storage Data & File: Read/Write](#_Storage_Data_&) for the details.

# Device Interface on Host Machine

CLI commands are supported on both Linux and Windows OS. However, there are some limitations on the same functionality in the Windows environment, which are highlighted in the respective sections.

## Device Port Identification on Linux OS

Generally, the host interface connected to the device takes the form of the device USB ports on Linux OS, /dev/ttyUSBn. Once connected to Talaria TWO EVB, the host detects 4 USB ports for the connected device.

Type in the Ls command in Linux to list the USB ports:

|  |
| --- |
| $ ls /dev/ttyUSB\* |

Text

Description automatically generated with medium confidence

Figure : USB ports

From the ports listed, identify the following ports:

1. /dev/ttyUSB2: Connected to the peripheral UART of Talaria TWO module
2. /dev/ttyUSB3: Connected to the console port (GPIO17) which is unidirectional UART from the Talaria TWO module. It operates at a high baud rate 2457600 (default), used for debug prints.

With this information available, all CLI commands should use the following argument for the peripheral port to the device in the command line:

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

This loads the helloworld app provided in SDK to Talaria TWO memory.

Launch miniterm, a small terminal application, to open the /dev/ttyUSB3 device port.

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

When an application runs on the device, it prints out debug messages through the console port.

Text

Description automatically generated

Figure : Debug messages displayed through console port

If there are multiple Talaria TWO devices where each of them has their own console port connected to the host via the USB interface, it will be sufficient to use the respective /dev/ttyUSBn with the command.

For instance, two Talaria TWO devices with USB connected.

Text

Description automatically generated

Figure : 2 Talaria TWO devices with USB connected

Figure 4 and Figure 5 show that the miniterm app is run on two separate terminals using respective device ports to communicate with the respective device.

Text

Description automatically generated

Figure : /dev/ttyUSB3

Text

Description automatically generated

Figure : /dev/ttyUSB7

## Device Port Identification on Windows OS

On a Windows platform, some control functionalities associated with the CLI device-reset feature requires additional support provided by installation of the libusbK library.

There are two ways for a Windows host to interface with the device EVB connected to the host via USB interface:

1. A Windows system with the libusbK installed (recommend)

The library can be installed with the Zadig tool: <https://zadig.akeo.ie/>

1. A Windows system without the libusbK installed

This is the default configuration when the EVB with USB is detected by Windows. It finds a generic, default USB/Serial driver to handle the device interface. In this setup, not all functionalities offered by the script tools are available.

Depending on the environment where the libusbK is installed or not, CLI use can be described as follows:

**On Windows Platform with libusbK installed (recommended)**

On this serial communication setup, Windows host is communicating with the device using the libusbK driver.

The libusbK’s installation and use by the host can be checked from the Windows Device Manager display.

Graphical user interface, text

Description automatically generated

Figure : libusbK driver installation

If there are multiple Talaria TWO devices connected, the Windows Device Manager appears:

Graphical user interface, text, application

Description automatically generated

Figure : Multiple Talaria TWO devices connected

With this setup in place, all CLI commands should use the following argument for the input port to the device in command line:

|  |
| --- |
| ftdi://ftdi:4232/3 |

where,

1. 4232: Device ID for the FTDI 4232H device on EVK
2. 3: FTDI UART interface ID for the input/programming on EVK

With this information available, all CLI commands should use the following argument for the peripheral port to the device in the command line:

|  |
| --- |
| python script\boot.py --device ftdi://ftdi:4232/3 --reset=evk42\_bl apps/hello\_world/bin/hello\_world.elf |

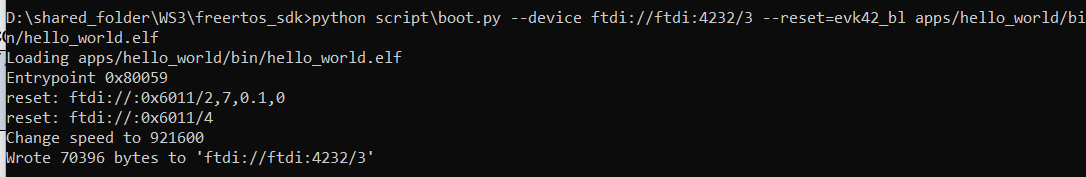


Figure : Flashing to Talaria TWO

This command loads the helloworld app provided in SDK to Talaria TWO memory.

If there are multiple Talaria TWO devices where each of them has their own peripheral port connected to the host via the USB interface, the device specifier must be expanded with the serial ID of the device in the command line. This is accomplished with the keyword --SN. Execute the following steps:

1. Append to the device port with the following string:

|  |
| --- |
| ftdi://ftdi:4232:**<device\_serial\_id>**/3, which is followed by: --SN **<device\_serial\_id>** |

1. Peripheral port to the device in the command line:

|  |
| --- |
| python script\boot.py --device ftdi://ftdi:4232:**10110001**/3 --reset=evk42\_bl --SN **10110001** apps/hello\_world/bin/hello\_world.elf |

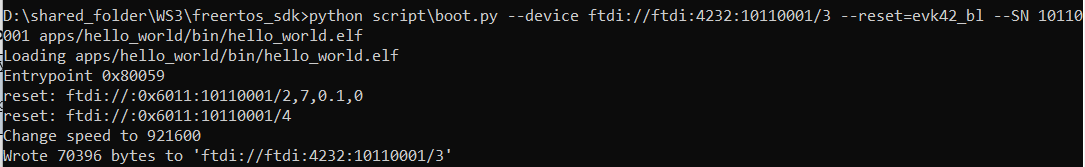


Figure 9: Expanding the device specifier

<device\_serial\_id>:

The hyphen ‘-‘ is a valid character in a serial ID of a device, if present. In other words, a serial ID of 11010391 is considered different than 1101-0391.

Example of a device with serial ID 00166d4:

|  |
| --- |
| python script\boot.py --device ftdi://ftdi:4232:**0016d4**/3 --reset=evk42\_bl --SN **0016d4** apps/hello\_world/bin/hello\_world.elf |

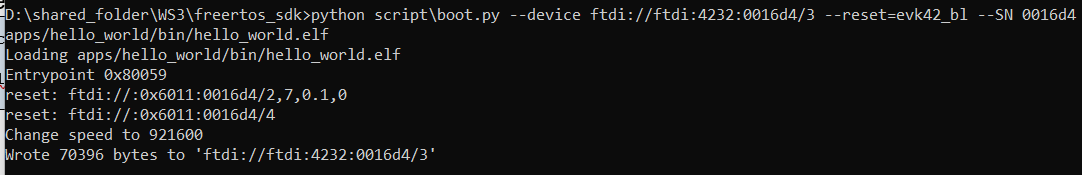


Figure : Device with serial ID 0016d4

**On Windows Platform using COM port (without the libusbK installed)**

On this serial communication environment, Windows host is communicating with the device using the standard (default) USB/Serial driver.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure : Standard (default) USB/serial driver

In the listed ports, identify the following ports:

1. COM6: Connected to the peripheral UART of Talaria TWO module
2. COM7: Connected to the console port (GPIO17) which is unidirectional UART from Talaria TWO module and operates at a high baud rate 2457600 (default), used for debug print

With this setup in place, all CLI commands can use the following argument for the input port to the device in the command line:

|  |
| --- |
| python script\boot.py --device COM6 apps\gordon.elf |

This command loads the gordon app provided in SDK to Talaria TWO memory.

# CLI Command Syntax and List Parameters

All commands with arguments are entered in a space-separated fashion.

Positional and Optional Arguments

Arguments should be passed on to the app to be loaded into the Talaria TWO device with the script. There are two different types of arguments:

1. Positional Arguments

Positional arguments must be included in the right order. Their positions in the sequence should be correct.

1. Optional Arguments

Optional arguments are optional. Defaults will be used when the arguments are not specified in the command line.

## Using boot.py

1. Command syntax to run boot.py

|  |
| --- |
| <path\_script>/boot.py [device] <path\_app\_elf>/app.elf [boot\_args] |

1. where,
   1. <path\_script>: Path to the script file. Default: Current folder
   2. [device]: Communication device ID with path.
      1. If there is only one Talaria TWO device with USB interface connected:
      2. Examples: COMn(Windows), /dev/ttyUSBn(Linux), ftdi://ftdi:xxxxxx(Windows libusbK installed).
      3. If there are multiple Talaria TWO devices where each of them has their own peripheral port connected to the host via the USB interface:
      4. The device specifier must be expanded with the serial ID of the device in the command line. This is accomplished with the keyword --SN. Execute the following:
      5. Command for device ID to the device port as shown above with the string is as follows:

|  |
| --- |
| ./script/boot.py --device /dev/ttyUSB2 --SN 15100010 --reset=evk42\_bl bins/iperf3.elf ssid=InnoPhase passphrase=Inno@9070 |

Refer section [Device Interface on Host Machine](#_Device_Interface_on) for information on multiple device connection on the host machine (Linux and Windows).

Default: Windows-none, Linux-/dev/ttyUSB2

1. <path\_app\_elf>: Path to the app ELF file
2. [boot\_args]: Bootargs. Examples: ssid=InnoPhase, passphrase=Inno@9070

Examples:

1. If there is only one Talaria TWO device with USB interface connected:

|  |
| --- |
| Note: A single line command  ./script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl bins/iperf3.elf ssid=InnoPhase passphrase=Inno@9070 |

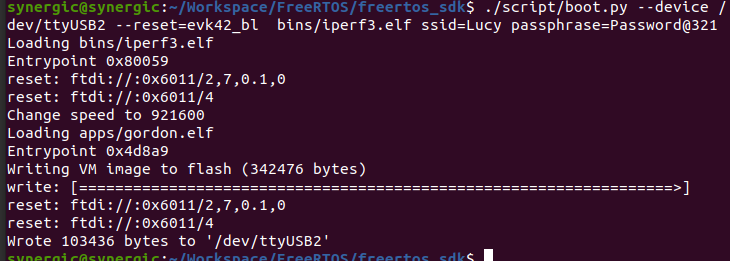


Figure : Single Talaria TWO device with USB interface

1. If there are multiple Talaria TWO devices connected:

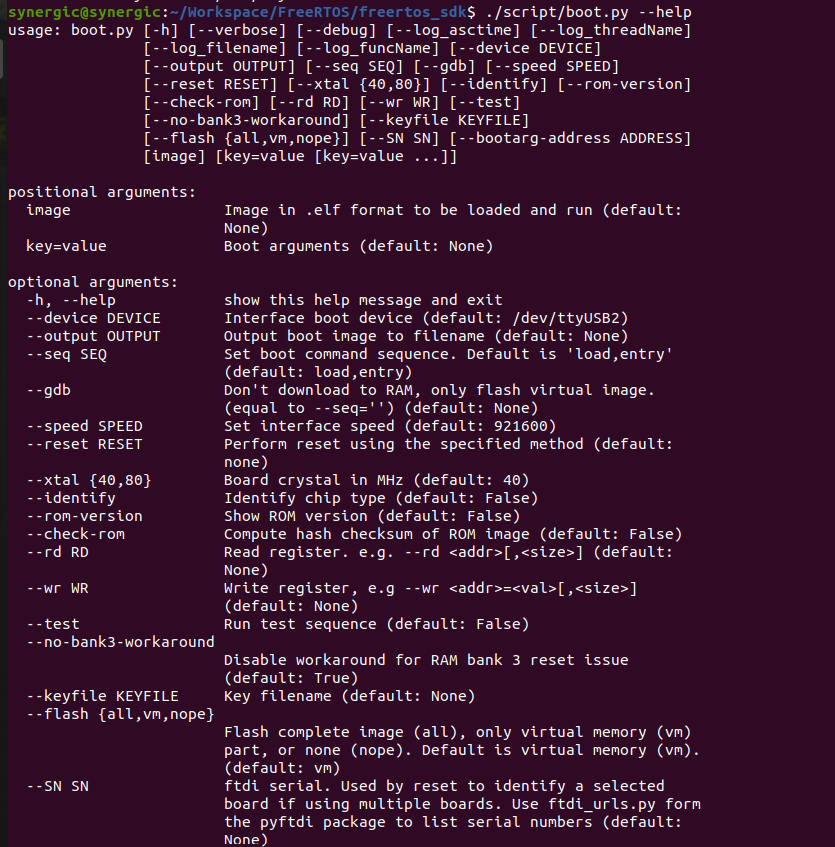
|  |
| --- |
| Note: A single line command  ./script/boot.py --device /dev/ttyUSB2 --SN 0016d4 --reset=evk42\_bl bins/iperf3.elf ssid=Lucy passphrase=Password@321 |



Figure : Multiple Talaria TWO devices connected

List Parameters (Manual in Script)

Type boot.py --help to bring up the manual of the boot.py command list:



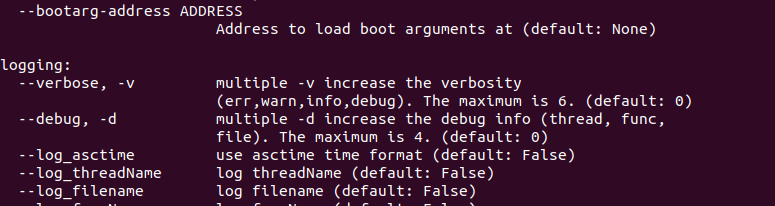


Figure : boot.py - command list

## Using flash.py

1. Command syntax to run flash.py

|  |
| --- |
| <path\_script>/flash.py [device] <pos\_arg> [boot\_args] |

where:

* 1. <path\_script>: Path to the script file. Default: Current folder
  2. [device]: Communication device ID with path.

For example: COMn(Windows), /dev/ttyUSBn(Linux), ftdi://ftdi:xxxxxx(Windows libusbK installed).

Default: Windows-none, Linux-/dev/ttyUSB2

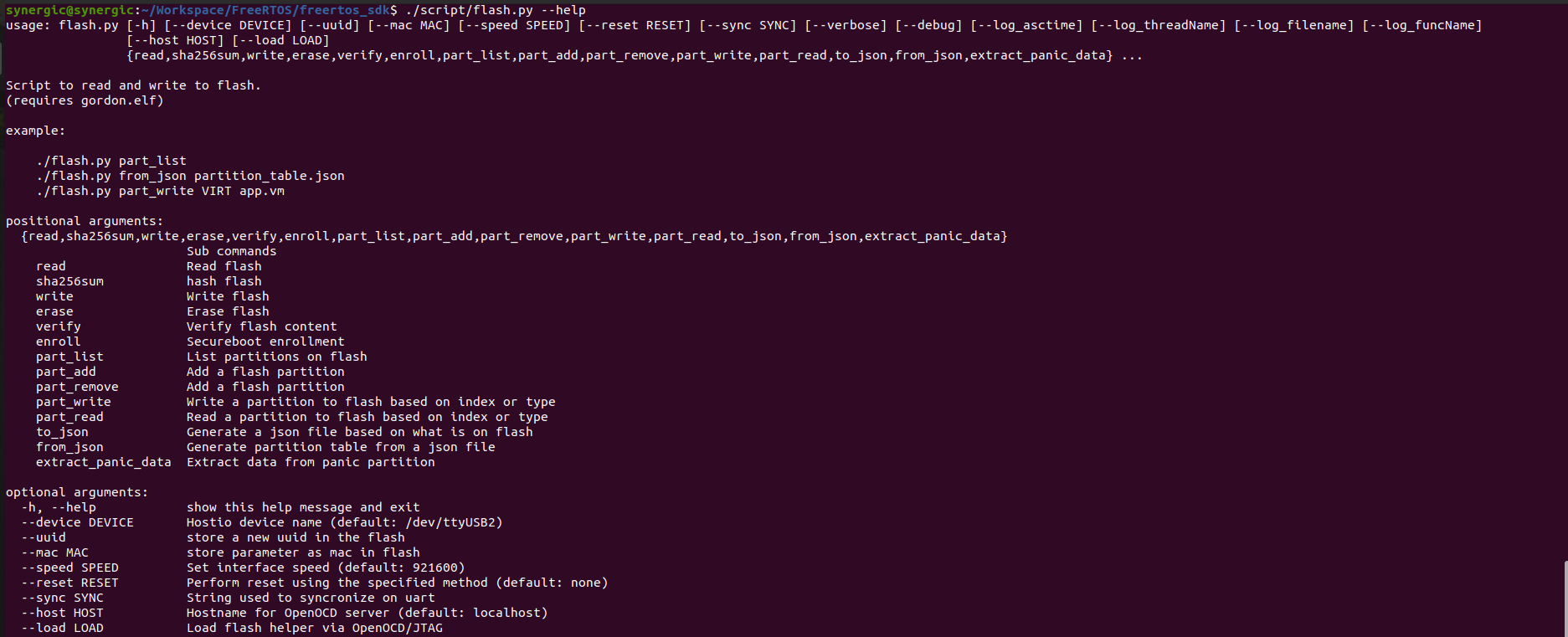
* 1. <pos\_arg>: Positional argument
  2. [boot\_args]: Bootargs

Example:

|  |
| --- |
| ./script/flash.py --device /dev/ttyUSB2 to\_json ./brd\_0183\_part.json |

1. List Parameters (Manual in Script)

Type flash.py --help to bring up the manual of the flash.py command list:



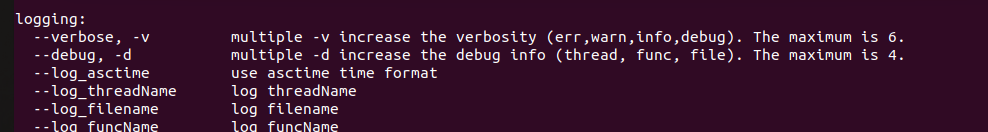


Figure : flash.py - command list

## Using storage.py

1. Command syntax to run storage.py

|  |
| --- |
| <path\_script>/storage.py [device] <pos\_arg> [boot\_args] |

where:

* 1. <path\_script>: Path to the script file. Default: Current folder
  2. [device]: Communication device ID with path.

For example: COMn, /dev/ttyUSBn, ftdi://ftdi:xxxxxx. Default: Windows-none, Linux-/dev/ttyUSB2

* 1. <pos\_arg>: Positional argument
  2. [boot\_args]: Bootargs

Example:

Connect to the Access Point configured in your **nprofile.json** as

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

1. List Parameters (Manual in Script)

Type storage.py --help to bring up the manual of the storage.py command list:

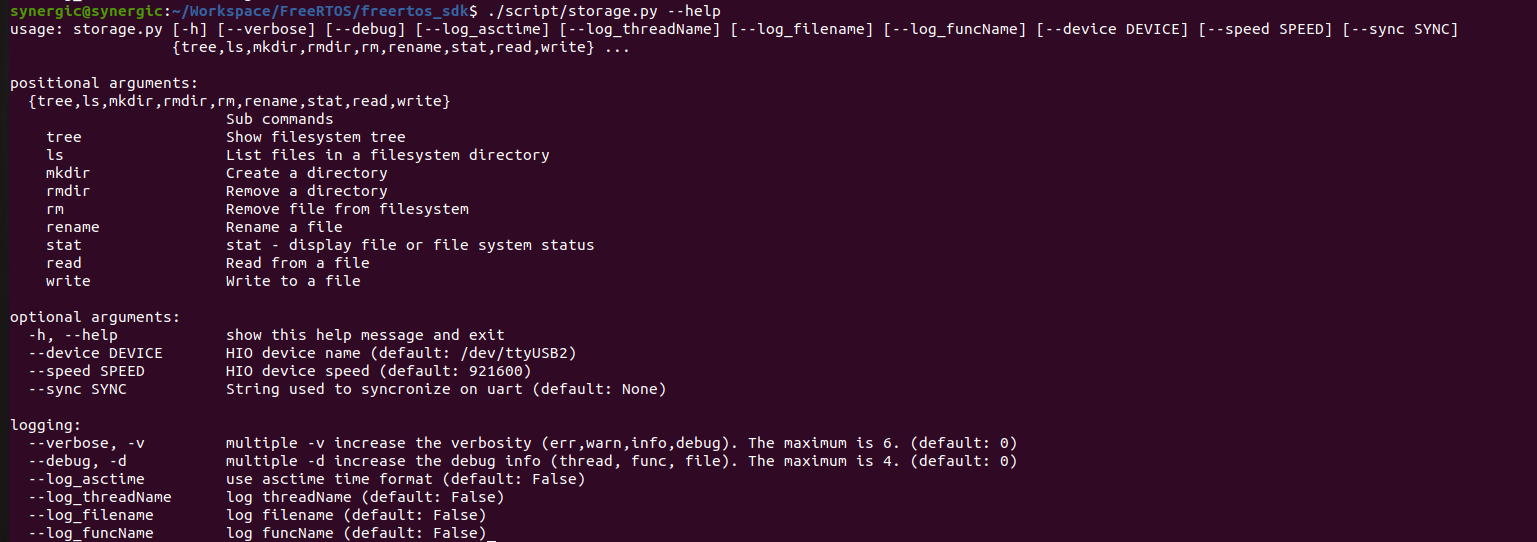


Figure : storage.py command script

# Use Cases

## Linux Platform

### Reset Device

1. **Normal Device Reset**

Reset the device.

**Command syntax**:

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

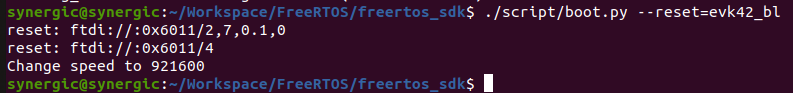


Figure : Normal device reset - terminal output

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** |

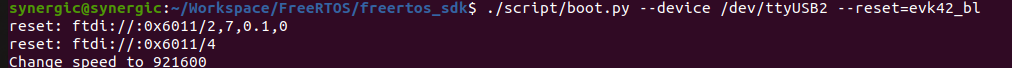


Figure : inhibiting-flash-boot - terminal output

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

Text

Description automatically generated

Figure : Abort from device port link

Launch miniterm to resume the device port connection.

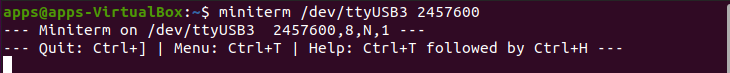


Figure : Resume device port connection

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.elf |

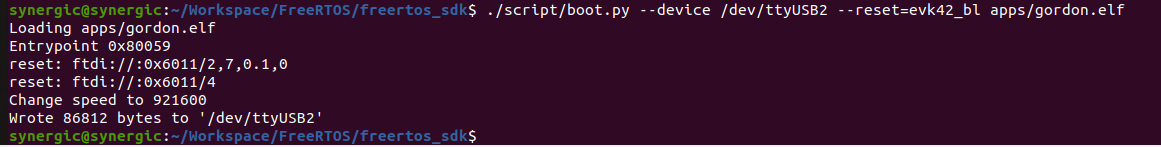


Figure : Inhibiting-flash-boot with booting Gordon - terminal output

### Program App in Memory

Program app in Talaria TWO memory.

**Command syntax**:

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

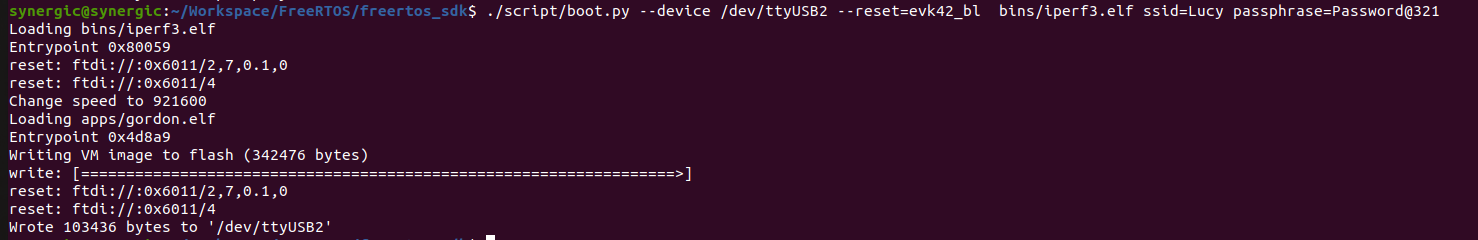
 

Figure : Programming app in Talaria TWO memory - terminal output

### 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** |

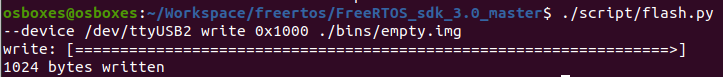


Figure : Flash empty.img

Button reset to confirm the erase operation is a success.

Text

Description automatically generated

Figure : 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] |

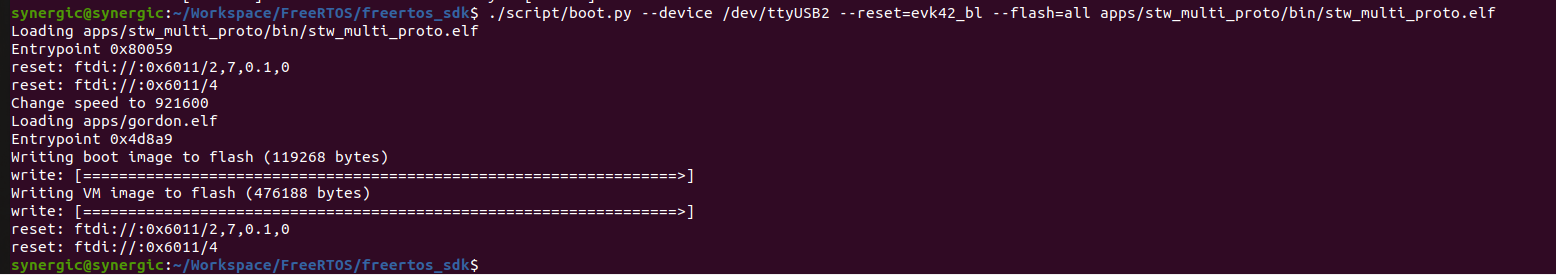


Figure : Program new image to flash

Button reset to confirm the flash operation a success.

Text

Description automatically generated

Figure : 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 |

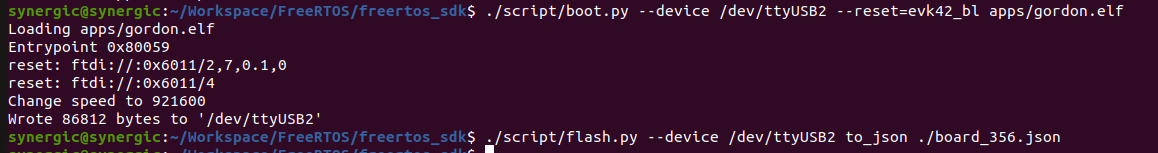


Figure : Inhibiting-flash-boot and load gordon.elf

Display of the board\_356.json file

Text, timeline

Description automatically generated

Figure : 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 |

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** |

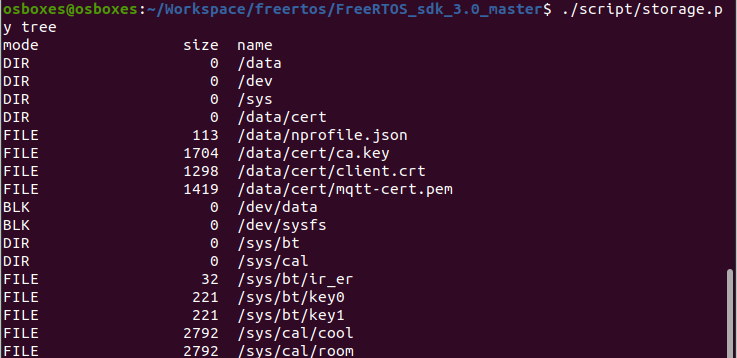


Figure : Show file system contents

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> |

The file is created:

Shape, rectangle

Description automatically generated with medium confidence

Figure : 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 : Running storage.py - write

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

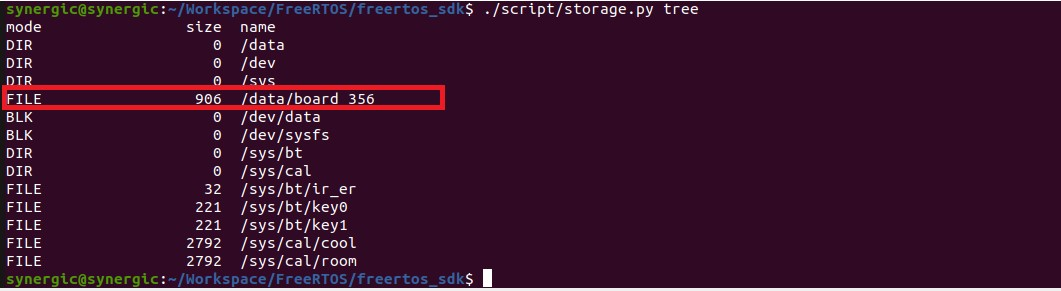


Figure : Check new file on Talaria TWO file system

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:**

|  |
| --- |
| ./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.

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

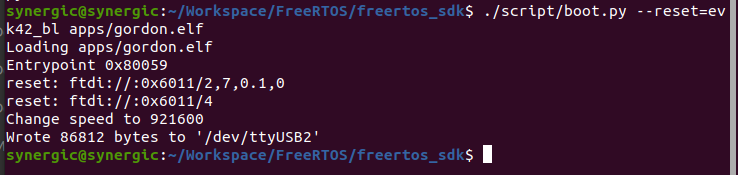


Figure : Starting gordon.elf

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

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

Command Log:

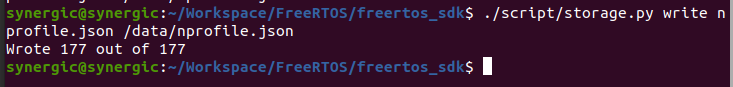


Figure : Writing network config file

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

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

Command Log:

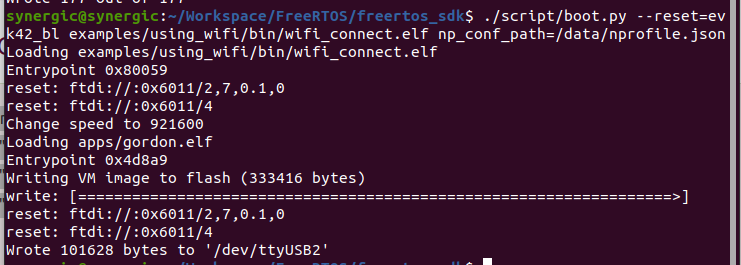


Figure : Program RAM to connect to AP

* 1. Program FLASH:

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

Command Log:

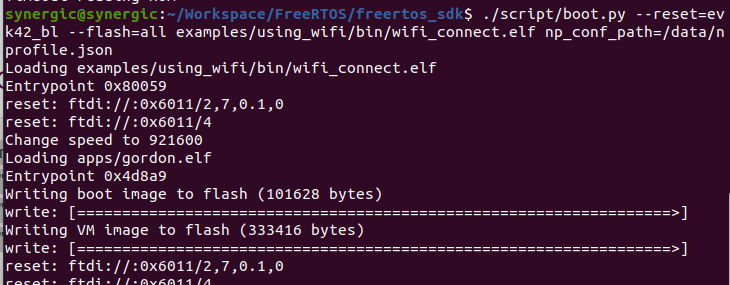


Figure : Program Flash to connect to AP

## Windows Platform – libusbK Installed

### Reset Device

1. **Device Reset**

Reset the device.

**Command syntax**:

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

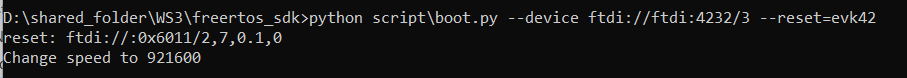


Figure : Device reset

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** |

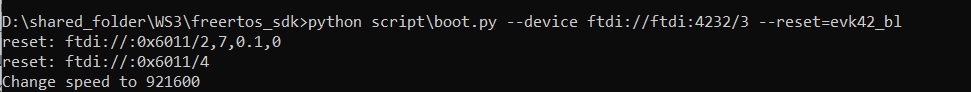


Figure : Device Reset with Inhibiting-flash-boot

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.elf |

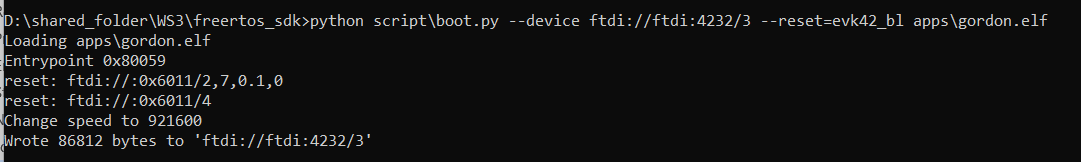


Figure : Reset with Inhibiting-flash-boot with booting Gordon

### 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] |

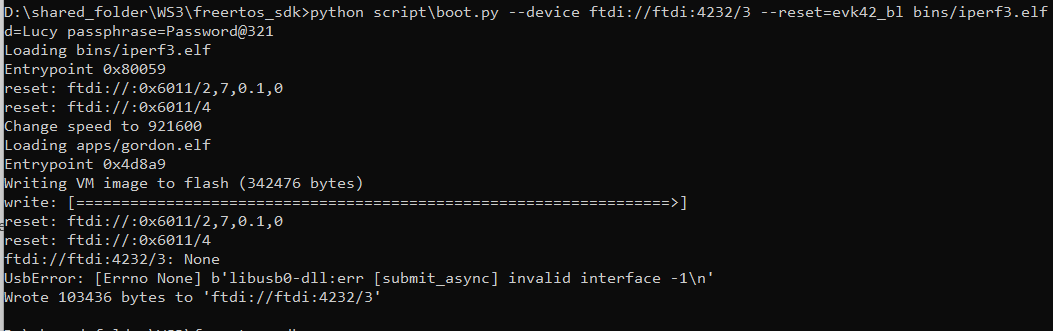


Figure : Program app in Talaria TWO memory

### 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** |

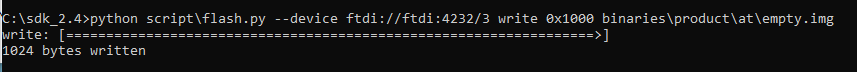


Figure : Erase image in flash

**Program New Image in Flash**

**Command syntax**:

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

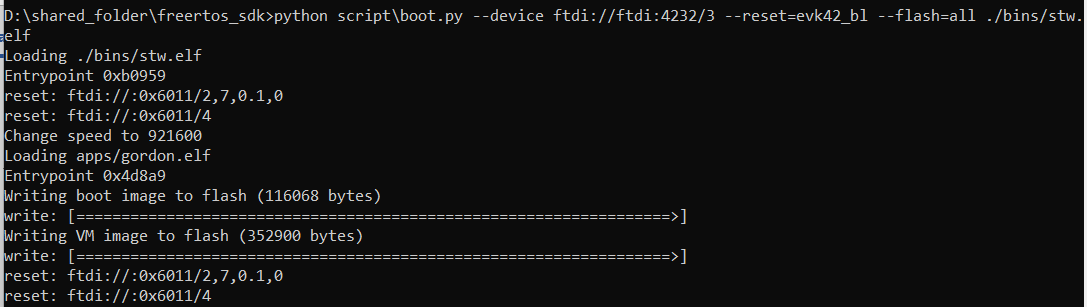


Figure : Program new image in flash

### 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 |



Figure : Read Partition Table from device

Display of the board\_0183.json file

Text, timeline

Description automatically generated

Figure : 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 |



Figure : 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** |

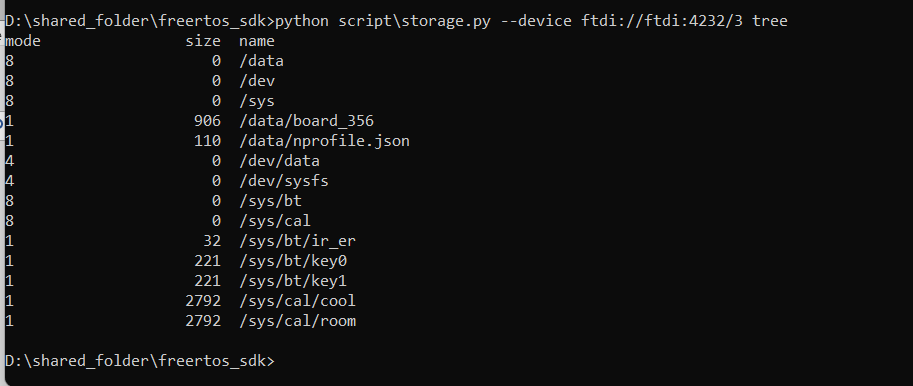


Figure : Show file system contents

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.

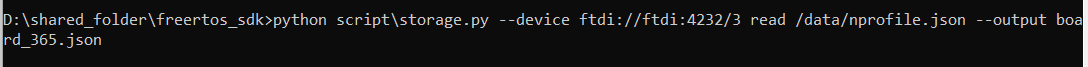


Figure : Read file and store to location on host

The file is created as per user entry:



Figure : 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.

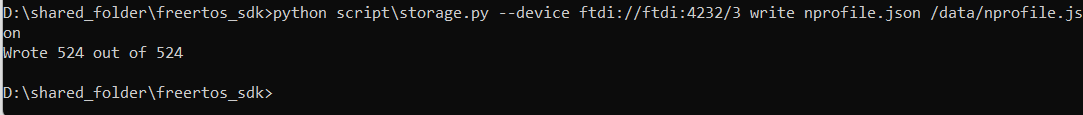


Figure : Write file to device

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

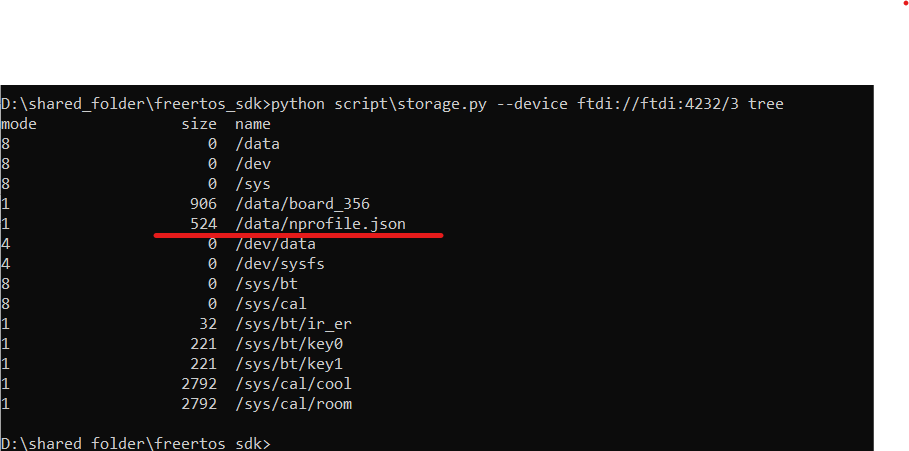


Figure : Check new file on Talaria TWO file system

## 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 the libusbK drivers to COM port, refer section: *Update Driver from libusbK to COM Port* of document UG\_Download\_Tool.pdf (*freertos\_sdk\_x.y\pc\_tools\Download\_Tool\doc*).

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

### Reset Device

All command reset is 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.elf |

### 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** |

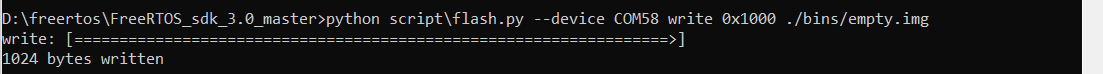


Figure : Erase image

Button reset to confirm the erase operation is a 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 |

Display of the board\_0407.json file

Text, timeline

Description automatically generated

Figure : 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 |

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** |

A screen shot of a computer

Description automatically generated



Figure : 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 : Read file and store to location on host

The file is created per user entry:



Figure : 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.





Figure : Write file to device

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

A screen shot of a computer

Description automatically generated

Figure : Check new file on Talaria TWO filesystem

### 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.elf |

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.

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 ./bins/iperf3.elf ssid=TP-LINK\_Guest\_864F passphrase=test13579 |

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

## 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 ./bins/stw.elf |

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