

# UM11442

NXP Wi-Fi and Bluetooth Demo Applications User Guide  
for i.MX RT Platforms

Rev. 6 – 14 Jan 2022

User manual

## Document information

Information	Content
Keywords	i.MX RT crossover MCU, i.MX RT products, MX RT1060 EVK board, MCUXpresso SDK, 88W8801-based wireless module, 88W8977-based wireless module, IW416- based wireless module, 88W8987-based wireless module, RTOS image,
Abstract	Provides step-by-step guidance to configure, compile, debug, flash and run the Wi-Fi and Bluetooth sample applications available in the MCUXpresso SDK. It also covers IDE configurations and required tool set up



## Revision history

Rev	Date	Description
v.1	20200717	Initial version
v.2	20210110	<p><b>Modifications:</b></p> <ul style="list-style-type: none"> <li>Extended the scope to IW416-based modules</li> <li><a href="#">Table 3</a>: updated</li> <li><a href="#">Section 1 "About this Document"</a>: updated</li> <li><a href="#">Section 2 "Tool Setup"</a>: updated</li> <li><a href="#">Section 3 "Wi-Fi Sample Applications"</a>: updated</li> <li><a href="#">Section 4 "Useful Wi-Fi APIs"</a>: updated</li> <li><a href="#">Section 3.5 "wifi_test_mode Sample Application"</a>: added</li> <li><a href="#">Section 3.6 "wifi_cert Sample Application"</a>: added</li> <li><a href="#">Section 5 "Bluetooth Classic/Low Energy Application"</a>: added</li> <li><a href="#">Section 6 "Acronyms and abbreviations"</a>: added</li> </ul>
v.3	20210331	<p><b>Modifications:</b></p> <ul style="list-style-type: none"> <li><a href="#">Section 1.3 "References"</a>: updated</li> <li><a href="#">Section 3.1.4 "Run a Demo using ARM GCC"</a>: updated</li> <li><a href="#">Section 3.5 "wifi_test_mode Sample Application"</a>: updated</li> <li><a href="#">Section 3.6 "wifi_cert Sample Application"</a>: updated</li> <li><a href="#">Section 3.6.1 "wifi_cert Application Execution"</a>: updated</li> <li><a href="#">Section 5 "Bluetooth Classic/Low Energy Application"</a>: updated</li> <li><a href="#">Table 15</a>: added</li> </ul>
v.4	20210602	<p><b>Modifications:</b></p> <ul style="list-style-type: none"> <li>Document Format modifications</li> <li><a href="#">Section 1.3 "References"</a>: updated</li> <li><a href="#">Section 1 "About this Document"</a>: updated</li> <li><a href="#">Section 2 "Tool Setup"</a>: updated</li> <li><a href="#">Table 3</a>: updated</li> <li><a href="#">Section 3.1 "wifi_iperf Sample Application"</a>: updated</li> <li><a href="#">Section 3.1.3.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.4.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.5.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.6.3 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.7.1 "Start-up logs"</a>: updated</li> <li><a href="#">Section 3.2 "wifi_setup Sample Application"</a>: added</li> <li><a href="#">Table 12</a>: updated</li> <li><a href="#">Section 3.3.1.1 "Run the application"</a>: updated</li> <li><a href="#">Section 3.3.1.3 "Wi-Fi Power Save"</a>: added</li> <li><a href="#">Section 3.3.1.4 "Other useful CLI commands"</a>: updated</li> <li><a href="#">Figure 35</a>: updated</li> <li><a href="#">Section 3.4.2.1 "Start-up logs"</a>: updated</li> <li><a href="#">Section 3.4.2.5 "Device reboot with configuration stored in mflash"</a>: updated</li> <li><a href="#">Section 3.5.1.3 "Wi-Fi Packet count"</a>: updated</li> <li><a href="#">Table 15</a>: updated</li> <li><a href="#">Section 3.5.1.8 "Other useful CLI commands"</a>: updated</li> <li><a href="#">Table 16</a>: updated</li> <li><a href="#">Table 17</a>: updated</li> <li><a href="#">Section 3.6.1.1 "Run the application"</a>: updated</li> <li><a href="#">Section 3.6.1.6 "Set/Get Tx Rate Configuration"</a>: updated</li> <li><a href="#">Table 21</a>: updated</li> </ul>

		<ul style="list-style-type: none"> <li><a href="#">Section 2.2 "Wireshark Tool Setup"</a>: added</li> <li><a href="#">Section 5 "Bluetooth Classic/Low Energy Applications"</a>: updated</li> <li><a href="#">Section 5.14.2 "audio_profile Application Execution"</a>: updated</li> <li><a href="#">Section 5.15.2 "wifi_provisioning Application Execution"</a>: updated</li> </ul>
v.5	20210823	<p><b>Modifications:</b></p> <ul style="list-style-type: none"> <li><a href="#">Section 1.3 "References"</a>: updated</li> <li><a href="#">Table 2</a>: updated</li> <li><a href="#">Section 3.1.3.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.4.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.5.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.6.3 "Project Settings"</a>: updated</li> <li><a href="#">Section 5.14 "Wireless UART Sample Application"</a>: added</li> <li><a href="#">Section 5.15 "Shell Sample Application"</a>: added</li> <li><a href="#">Table 23</a>: updated</li> </ul>
v.6	20220114	<p><b>Modifications:</b></p> <ul style="list-style-type: none"> <li><a href="#">Table 2</a>: updated</li> <li><a href="#">Section 3.1.3.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.4.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.5.2 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.6.3 "Project Settings"</a>: updated</li> <li><a href="#">Section 3.1.7.1 "Start-up logs"</a>: updated</li> <li><a href="#">Section 3.2.1.1 "Run the application"</a>: updated</li> <li><a href="#">Section 3.3.1.4 "Other useful CLI commands"</a>: updated</li> <li><a href="#">Section 3.4.2.1 "Start-up logs"</a>: updated</li> <li><a href="#">Section 3.4.2.5 "Device reboot with the configurations stored in mflash"</a>: updated</li> <li><a href="#">Section 3.5.1.8 "Other useful CLI commands"</a>: updated</li> <li><a href="#">Section 3.6.1.3 "Set/Get Tx Power Limit"</a>: updated</li> <li><a href="#">Table 23</a>: updated</li> <li><a href="#">Section 5.1.1 "a2dp_sink Application Execution"</a>: updated</li> <li><a href="#">Section 5.15.1.1 "Shell Run the application"</a>: updated</li> <li><a href="#">Table 24</a>: updated</li> <li><a href="#">Section 5.16.2.3 "Create IoT thing, private key, and certificate for device"</a>: updated</li> <li><a href="#">Section 5.16.2.5 "Configure the AWS IoT endpoint"</a>: updated</li> <li><a href="#">Section 5.16.2.4 "Configure the AWS IoT Certificate and Private Keys"</a>: updated</li> <li><a href="#">Section 5.17.2 "wifi_provisioning Application Execution"</a>: updated</li> </ul>

## 1 About this Document

---

### 1.1 Purpose and Scope

This document provides the steps to configure, compile, debug, flash and run the Wi-Fi and Bluetooth sample applications available in the MCUXpresso SDK. It also covers IDE configurations and required tool set up.

### 1.2 Considerations

The i.MX RT is powered by FreeRTOS and the RTOS drivers are added to support the 88W8801, 88W8977, IW416, and 88W8987 NXP-based wireless modules. This document does not include NXP-based wireless modules information, i.MX RT product information, hardware interconnection, board settings, bring-up, IDE setup, SDK download, as these are covered in the [UM11441](#). The user must have i.MX RT platform related IDE and tools installed before going through the given demo process.

## 1.3 References

Table 1: Reference Documents

Reference Type	Description
User manual	NXP – MCUXSDKGSUG - Getting Started with MCUXpresso SDK ( <a href="#">link</a> )
Web page	NXP - Getting Started with Wi-Fi on i.MX RT platforms ( <a href="#">link</a> )
User manual	NXP – UM11441 - Getting Started with NXP-based Wireless Modules and i.MX RT Platform Running on RTOS <i>SDK Documents available at SDK_&lt;version&gt;_EVK-&lt;RT-Platform&gt;_doc\docs\wireless</i>
User manual	NXP - MCUXpresso_SDK_WLAN_Driver_Reference_Manual.pdf SDK Documents available at <i>SDK_&lt;version&gt;_EVK-&lt;RT-Platform&gt;_doc\docs\wireless\Wi-Fi</i>
User manual	NXP - Hardware Rework Guide for MIMXRT1060-EVK and AW-AM457-uSD.pdf SDK Documents available at <i>SDK_&lt;version&gt;_EVK-&lt;RT-Platform&gt;_doc\docs\wireless.</i>
User manual	SIG - Core Specification ( <a href="#">link</a> )
Android Application	NXP – AwsMusicControl.apk SDK Source: <i>SDK_&lt;PATH&gt;\boards\evkmimxrt1060\edgefast_bluetooth_examples\audio_profile\android_app.</i>
Configuration file	NXP - aws_clientcredential.h SDK Source: <i>SDK_&lt;PATH&gt;\rtos\freertos\demos\include.</i>
Configuration file	NXP - CertificateConfigurator.html SDK Source: <i>SDK_&lt;PATH&gt;\rtos\freertos\tools\certificate_configuration.</i>
Android Project	NXP – amazon-freertos-ble-android-sdk SDK Source: <i>SDK_&lt;PATH&gt;\boards\evkmimxrt1060\edgefast_bluetooth_examples\wifi_provisioning.</i>
Mobile application	NXP - IoT Toolbox Android ( <a href="#">IoT Toolbox on Google Play</a> ) ( <a href="#">IoT Toolbox on the APP Store</a> )

## 2 Tool Setup

### 2.1 Serial Console Tool Setup

The serial console tool is used to read out the demo application's logs on the computer connected to i.MX RT EVK board.

- Download and install the terminal emulator software such as minicom (Linux or Mac OS) or Tera Term (Windows)
- Use a micro USB to USB cable to connect i.MX RT1060 EVK board to the host computer running on Linux, Mac OS or Windows.
- Open a terminal emulator program like minicom or Tera term.
- For minicom use following command and configure the below settings for serial console access:

```
# minicom -s

Serial Port Setup:
- /dev/ttyACMX serial port
- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control
```

Prior to running the Bluetooth demo application, update the serial console configuration so there is no extra spacing.

For Tera Term:

- Go to Setup > Terminal
- Look for the New line section
- Set the **Receive to Auto**

For minicom:

Press **Ctrl + A** and then press **Z** key to open the Help menu

Press the **U** key to add a carriage return

### 2.2 Wireshark Tool Setup

The Wireshark tool is required to analyze the Wi-Fi sniffer logs. Download and install Wireshark tool for Windows and Mac OS from [here](#).

Steps to install Wireshark tool on a computer running Linux Ubuntu:

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt update
sudo apt install wireshark
```

### 3 Wi-Fi Sample Applications

This chapter describes the Wi-Fi example applications that are available in the SDK, and the steps to configure, compile, debug, flash, and execute these examples.

These Wi-Fi examples can be configured based on the Wi-Fi modules used with the help of Wi-Fi module-specific macros.

Table 2 lists the Wi-Fi module specific macros that are common to all Wi-Fi examples. Macros are available in the file `evk<RT-Platform>\wifi_iperf\source\app_config.h`

Table 2: Macros for Wi-Fi Modules

Module	Chipset	Macro
Panasonic PAN9026	88W8977	WIFI_88W8977_BOARD_PAN9026_SDIO
AzureWave AW-AM281	88W8977	WIFI_88W8977_BOARD_AW_AM281_USD
AzureWave AW-NM191NF	88W8801	WIFI_88W8801_BOARD_AW_NM191_USD <sup>[1]</sup> WIFI_88W8801_BOARD_AW_NM191MA
AzureWave AW-AM457	IW416	WIFI_IW416_BOARD_AW_AM457_USD WIFI_IW416_BOARD_AW_AM457MA
AzureWave AW-AM510	IW416	WIFI_IW416_BOARD_AW_AM510_USD <sup>[1]</sup> WIFI_IW416_BOARD_AW_AM510MA <sup>[1]</sup>
AzureWave AW-CM358	88W8987	WIFI_88W8987_BOARD_AW_CM358_USD <sup>[1]</sup> WIFI_88W8987_BOARD_AW_CM358MA <sup>[1]</sup>
Murata Type 2DS	88W8801	WIFI_88W8801_BOARD_MURATA_2DS_USD <sup>[1]</sup> WIFI_88W8801_BOARD_MURATA_2DS_M2
Murata Type 1XK	IW416	WIFI_IW416_BOARD_MURATA_1XK_USD <sup>[1]</sup> WIFI_IW416_BOARD_MURATA_1XK_M2
Murata 1ZM	88W8987	WIFI_88W8987_BOARD_MURATA_1ZM_USD <sup>[1]</sup> WIFI_88W8987_BOARD_MURATA_1ZM_M2

[1] The module operation was tested during 2.11 release process.

USD=microSD interface

M2=M.2 interface

### 3.1 wifi\_iperf Sample Application

The sample application implements the protocol used by iPerf performance measurement tool. The performance is measured between a single i.MX RT+NXP-based Wireless module and a computer running the iPerf tool. The instructions in this guide use an i.MX RT1060 EVK board. Yet the same steps apply to other i.MX RT products. The following figures show the setup overview to run the wifi\_iperf sample applications with Soft AP mode or with Station mode

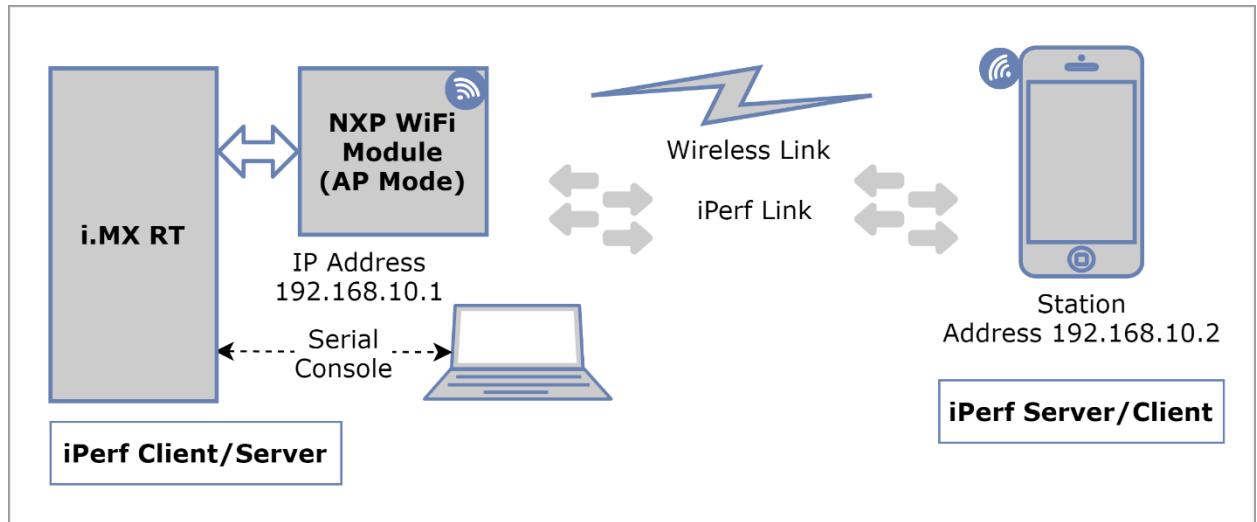


Figure 1: Hardware Setup for wifi\_iperf Sample Application with Soft AP Mode

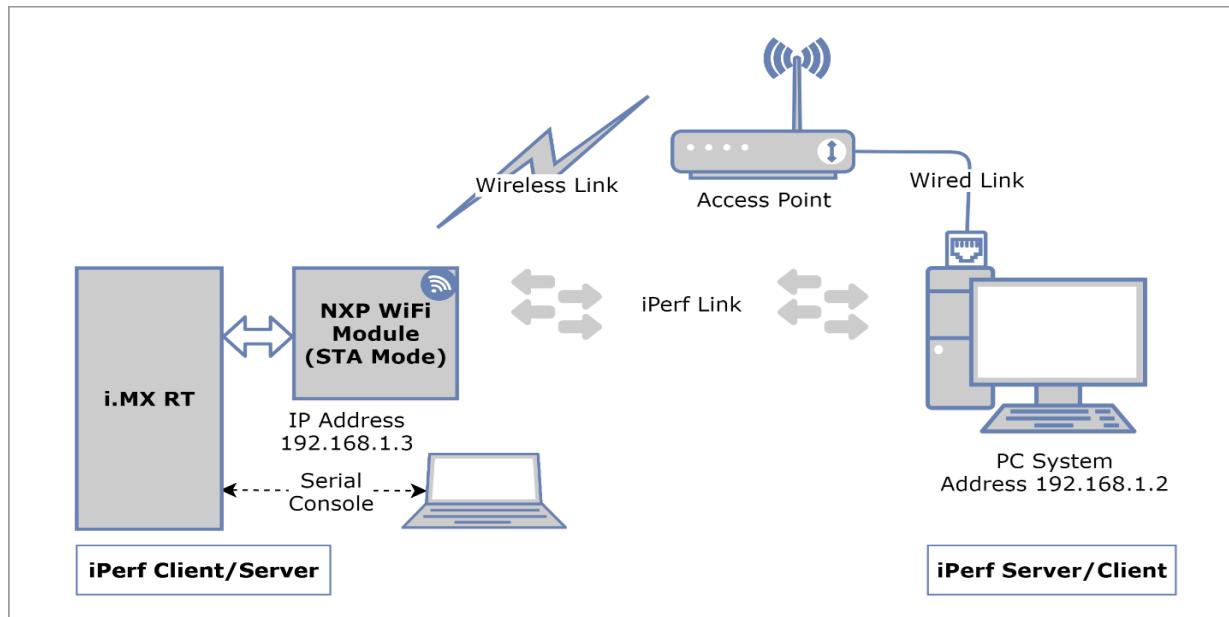


Figure 2: Hardware Setup for wifi\_iperf Sample Application with Station Mode

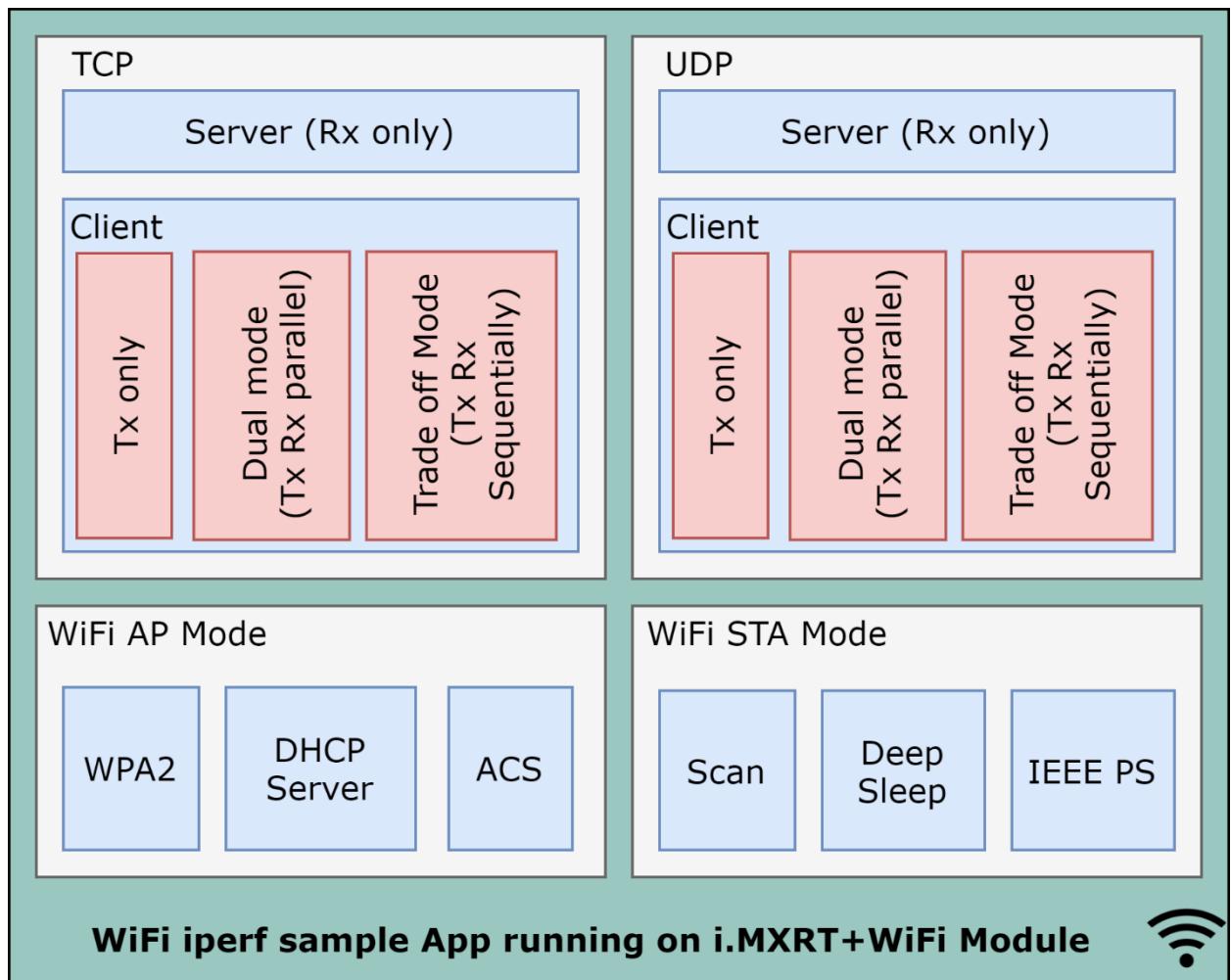


Figure 3: wifi\_iperf Sample Application Components

**Wi-Fi and iPerf Features:**

Table 3: Sample Application Features

Features	Details
Wi-Fi	Wi-Fi Soft AP mode Wi-Fi Station mode Wi-Fi Scan Wi-Fi IEEEPS (power saving) mode Wi-Fi deep-sleep power saving mode
iPerf	TCP Client and Server TCP Client dual mode (Tx and Rx in parallel) TCP Client trade-off mode (Tx and Rx sequentially) UDP Client and Server UDP Client dual mode (Tx and Rx in parallel) UDP Client trade-off mode (Tx and Rx sequentially)

**3.1.1 User Configurations**

This section describes the demo-related configurations that users can modify according to their needs.

**3.1.1.1 Wi-Fi Configurations**

Some of the Wi-Fi features and feature related macros that user can configure based on requirement are listed in below table along with source file name.

Table 4: Wi-Fi Configurations

Feature	Macro definition	Default value	File name	Details
Wi-Fi Soft AP	AP_SSID AP_PASSPHRASE	“NXP_Soft_AP” “12345678”	main.c	Default SSID and passphrase to start soft AP using the given sample application. It can be modified by changing the macro value. Default wpa2 security is used.
Wi-Fi STA	EXT_AP_SSID EXT_AP_PASSPHRASE	“nxp_wifi_demo” NONE	main.c	Default configuration (of targeted AP) to start station mode using given sample application. It can be modified by changing the macro value.

### 3.1.1.2 IPerf Configurations

Some of the IPerf feature related macros that user can configure based on requirement are listed in below table along with source file name.

Table 5: IPerf Configurations

Features	Macro definition	Default value	File name	Details
IPerf server address	IPERF_SERVER_ADDRESS	"192.168.1.2"	main.c	The IP address of the remote host (computer) IPerf server.
UDP data transmission rate	IPERF_UDP_CLIENT_RATE	100 (100 Mbps)	main.c	Changing this value impacts the bandwidth and bit rate. This rate can be controlled by remote host when the sample app runs in UDP server mode. If the remote host is in trade-off mode, there is no need to update this macro.
TCP Window size	TCP_WND	(15 * TCP_MSS)	lwipopts.h	Used for TCP window scaling 1 TCP_MSS (Maximum Segment Size) = 1460 bytes Ensure the PBUF_POOL_SIZE is greater than (TCP_WND / TCP_MSS).
IPerf Client Session Timeout	IPERF_CLIENT_AMOUNT	-1000 (10 seconds)	main.c	To make IPerf Client session time-limited.

**NOTE:** For RTOS applications, DEFAULT\_THREAD\_PRIO and TCPIP\_THREAD\_PRIO values can also affect the maximum throughput. UDP data transmission rate depends on the system time, which has a resolution of 1 ms in LwIP. Therefore, the actual data transmission rate could be a little lower or higher due to sending a calculated (rounded) number of frames each one or more milliseconds. It can also result in higher jitter.

### 3.1.2 IPerf Remote Host Setup

#### Remote host setup for OS-Windows:

Perform the following step to complete the setup:

- Download IPerf version 2.0.5 from [here](#).

Perform the following steps to run the iPerf:

- Traverse to the path using cmd prompt where the iperf is downloaded.

```
> cd C:\Users\XXXX\Downloads
```

- Run the suitable command from the following table.

Table 6: iPerf Commands for Windows Remote Host

Functionality	Command
TCP server	iperf.exe -s -i 1
UDP server	iperf.exe -s -u -i 1
TCP client	iperf.exe -c <server_ip> -i 1 -t 60
UDP client	iperf.exe -c <server_ip> -u -i 1 -t 60

#### Remote host setup for OS-Linux:

Perform the following steps to complete the setup:

- Download Debian package of IPerf 2.0.5 for Ubuntu 16.04 from [here](#)

```
$ sudo wget https://iperf.fr/download/ubuntu/iperf_2.0.5+dfsg1-2_amd64.deb
```

- Install the package using one of the commands below.

```
$ sudo dpkg -i iperf_2.0.5+dfsg1-2_amd64.deb
```

OR

```
$ sudo apt install /path/to/package/iperf_2.0.5+dfsg1-2_amd64.deb
```

**NOTE:** Iperf 2.0.5 is used for the demonstration.

- Run the suitable command from the following table.

Table 7: iPerf Commands for Linux Remote Host

Functionality	Command
TCP server	iperf -s -i 1
UDP server	iperf -s -u -i 1
TCP client	iperf -c <server_ip> -i 1 -t 60
UDP client	iperf -c <server_ip> -u -i 1 -t 60

#### Remote host setup for mobile phone:

Perform the following steps to run the iPerf:

- Download the iPerf application like Magic iPerf, HE.NET Network Tools etc.
- Open the application and select the iperf2. Run the suitable from the following table.

Table 8: iPerf Commands for Mobile Phone Remote Host

Functionality	Command
TCP server	-s -i 1
UDP server	-s -u -i 1
TCP client	-c <server_ip> -i 1 -t 60
UDP client	-c <server_ip> -u -i 1 -t 60

### 3.1.3 Run a Demo with MCUXpresso IDE

This section describes the steps to import, configure, build, debug and run the demo example through MCUXpresso IDE. MCUXpresso IDE version v11.4.0 is used for the following demo steps.

#### 3.1.3.1 Project Import

##### Step 1: SDK Installation

- Open MCUXpresso IDE.
- Locate the **Installed SDKs** tab at the bottom of the following image.
- Drag and drop the SDK into the **Installed SDKs** tab. Once done click “OK” on the pop up window.

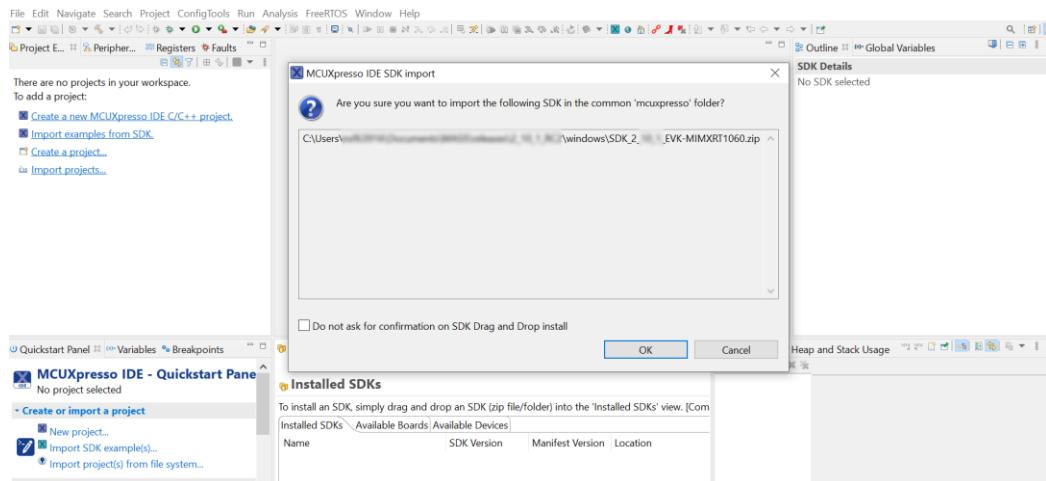


Figure 4: SDK Drag and Drop in MCUXpresso

##### Step 2: Import an example

- Go to the **Quickstart panel** and select the option **Import SDK example(s)**.



Figure 3: SDK Import Example in MCUXpresso

**Step 3:** Select EVK board.

- Select the evaluation board.

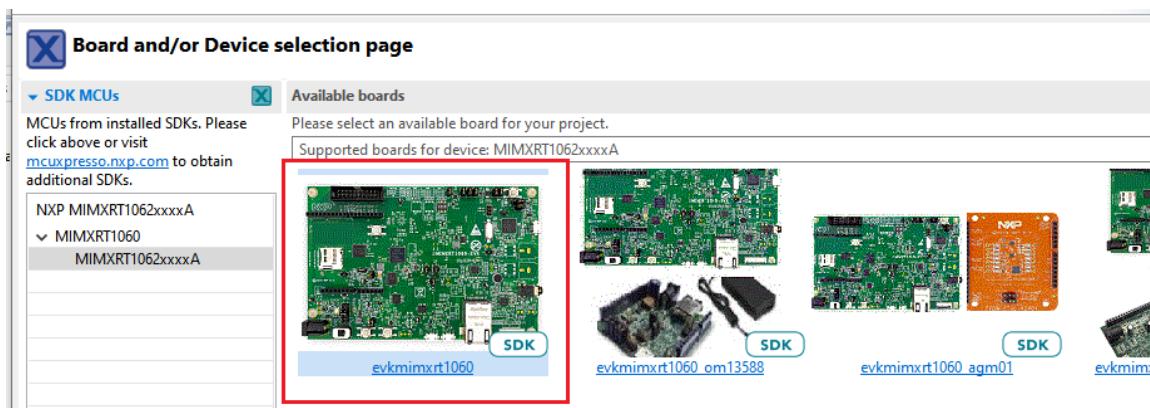


Figure 5: Device/EVK Selection in MCUXpresso

**Step 4:** Select any Wi-Fi or Bluetooth example and verify default Project Options.

- For example select wifi\_examples > wifi\_iperf and press **Finish** button to import the selected example into the workspace.

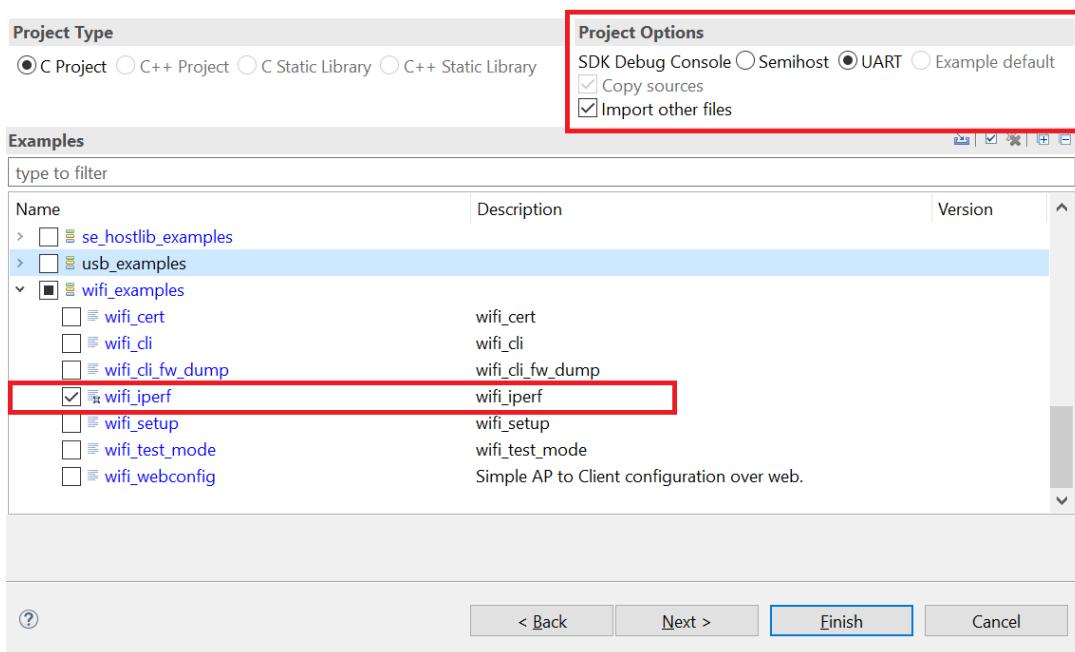
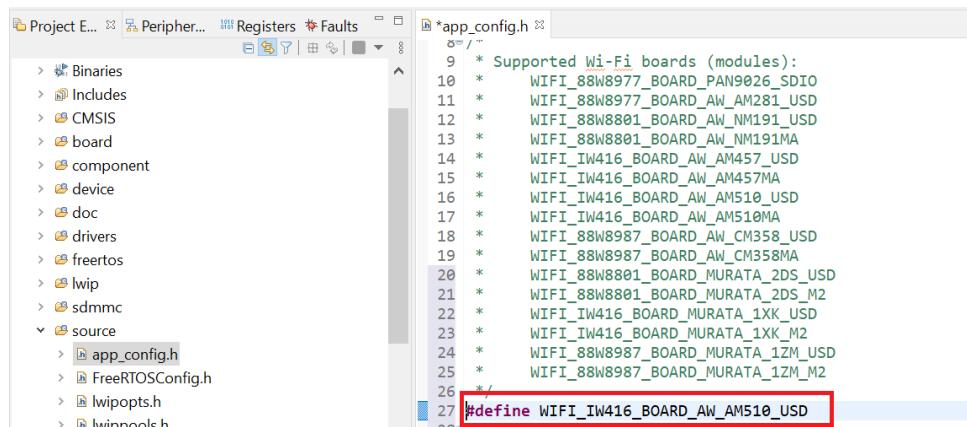


Figure 6: Sample App Selection in MCUXpresso

### 3.1.3.2 Project Settings

- By default, the project is configured to use the **WIFI\_IW416\_BOARD\_AW\_AM510\_USD** Wi-Fi module based on IW416 chipset. Modify the value to match the module on your setup to include and compile the desired driver, components and application(s).
- The file "**app\_config.h**" from the source folder is used for the macro definitions.
- Refer to Table 2 for the list of macros for Wi-Fi modules.



```
* Supported Wi-Fi boards (modules):
WIFI_88W8977_BOARD_PAN9826_SDIO
WIFI_88W8977_BOARD_AW_AM281_USD
WIFI_88W8801_BOARD_AW_NM191_USD
WIFI_88W8801_BOARD_AW_NM191MA
WIFI_IW416_BOARD_AW_AM457_USD
WIFI_IW416_BOARD_AW_AM457MA
WIFI_IW416_BOARD_AW_AM510_USD
WIFI_IW416_BOARD_AW_AM510MA
WIFI_88W8987_BOARD_AW_CM358_USD
WIFI_88W8987_BOARD_AW_CM358MA
WIFI_88W8801_BOARD_MURATA_2DS_USD
WIFI_88W8801_BOARD_MURATA_2DS_M2
WIFI_IW416_BOARD_MURATA_1XK_USD
WIFI_IW416_BOARD_MURATA_1XK_M2
WIFI_88W8987_BOARD_MURATA_1ZM_USD
WIFI_88W8987_BOARD_MURATA_1ZM_M2

#define WIFI_IW416_BOARD_AW_AM510_USD
```

Figure 7: Wi-Fi Module Selection in MCUXpresso

### 3.1.3.3 Build the Application

- To build the application, go to the **Quickstart** panel and select **Build**, or select the **Build** icon in the main toolbar.

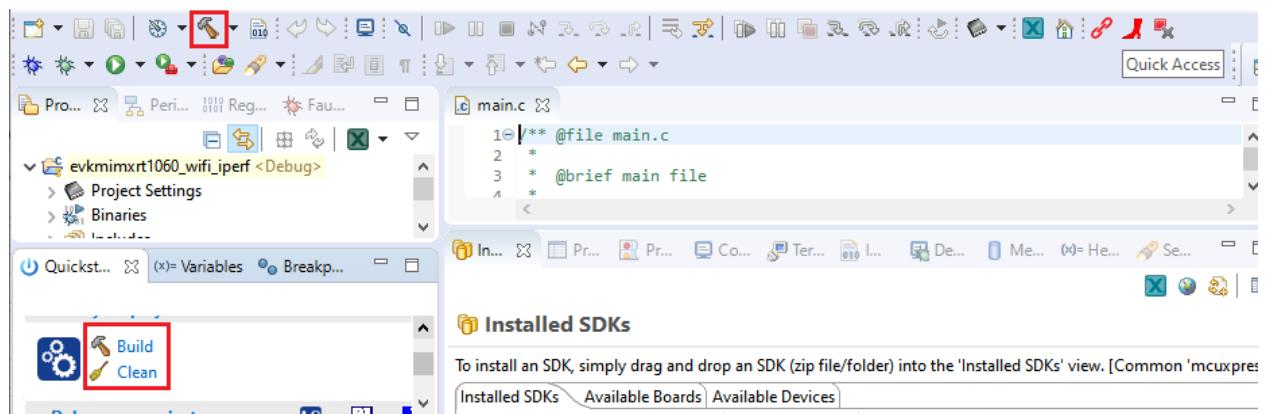


Figure 8: Application Build in MCUXpresso

- Verify the build result (success or fail) on the console window.

A screenshot of the MCUXpresso IDE showing the CDT Build Console window. The console output is as follows:

```
CDT Build Console [evkmimxrt1060 wifi_iperf]
Finished building target: evkmimxrt1060_WIFI_iperf.axt

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "evkmimxrt1060_wifi_iperf.axf"; # arm-none-eabi-objcopy -v
    text      data      bss      dec      hex filename
 722088     68631   133828   924547  e1b83 evkmimxrt1060_wifi_iperf.axf

12:39:48 Build Finished. 0 errors, 0 warnings. (took 49s.482ms)
```

The entire console window is highlighted with a red border.

Figure 9: Build Messages in MCUXpresso

### 3.1.3.4 Run the Application in Debug Mode

Please follow these steps to run the application in debug mode.

- Initiate the application debug using the debug icon in the toolbar or go to the **Quickstart** panel and select **Debug**.

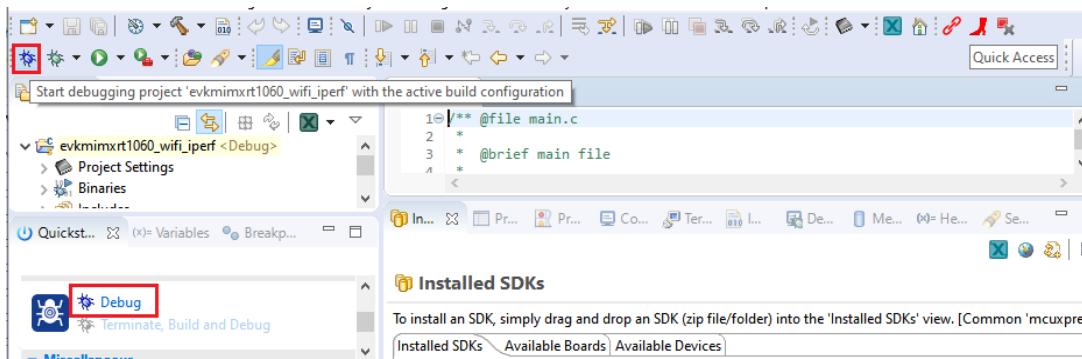


Figure 10: Initiate Debug in MCUXpresso

- Select the associated emulator probe for the first time as illustrated below and press **OK**.

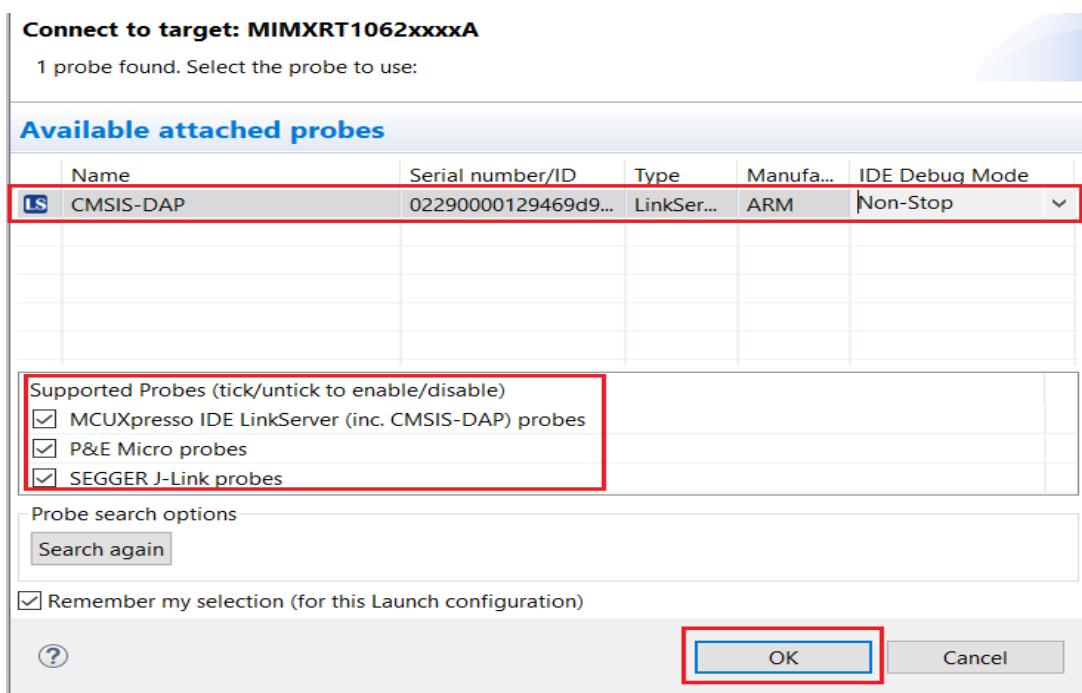


Figure 11: Emulator Probe Selection in MCUXpresso

- Upon selecting the probe, the application is downloaded on the board and the program execution starts with the program counter set at the main() function. Press **Resume** to start the application. To debug the application, use the **step into**, **step over** and **step return** buttons. To end the debugging session, use the **Terminate** button.

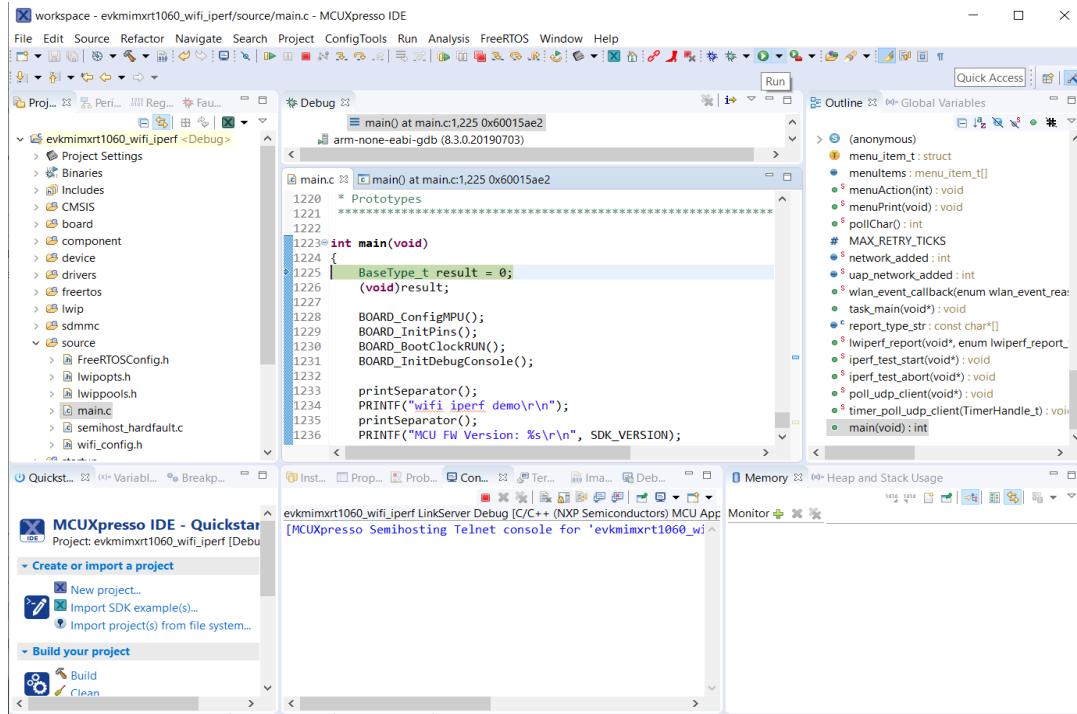


Figure 12: Application Debugging in MCUXpresso

### 3.1.3.5 Flash the Application Program (no debugging)

Please use the following steps to flash the application program.

To flash the required binaries, select the **GUI Flash Tool icon** in the toolbar as shown in the figure below. The GUI Flash Tool can be used to flash pre-build binary or locally compiled binary with \*.axf or \*.bin format. The path to the locally compiled binary is the following.

`$(workspace_loc)\evkmimxrt1060_wifi_iperf\Debug\evkmimxrt1060_wifi_iperf.axf`

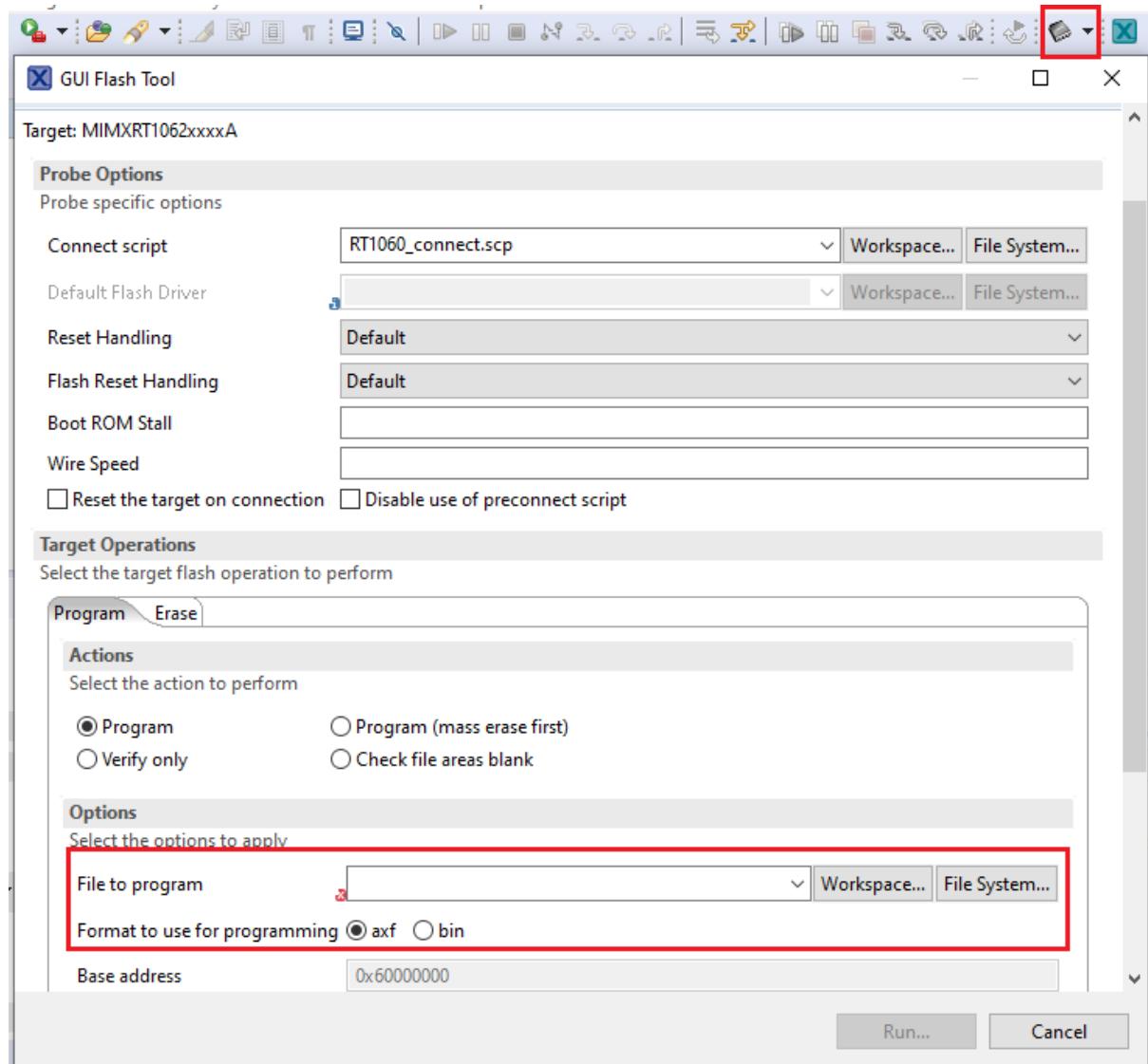


Figure 13: Binary Flashing in MCUXpresso

**NOTE:** Please refer to section 3.1.7 to view the output on the console once the application is executed.

### 3.1.4 Run a demo using ARM® GCC

This section describes the steps to configure the command line ARM® GCC tools to build and run demo applications. The wifi\_iperf application is used as an example, yet the same steps apply to any other example application available with the MCUXpresso SDK. The example uses Linux, one of the operating systems that ARM GCC tools support. Please refer to [MCUXSDKGSUG](#) for more details on ARM GCC toolchain setup.

#### 3.1.4.1 Install ARM® GCC toolchain

In this section, the following steps are given to install toolchain:

- Download the toolchain for Linux x86\_64 system from the [Link](#) (package *Linux x86\_64 tarball*).
- Create a directory at the location of your choice:

```
$ mkdir toolchain-dir
```

- Copy the downloaded toolchain package to the created directory and extract the downloaded toolchain.

```
$ cp <download_path>/gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2  
toolchain-dir/
```

```
$ cd toolchain-dir/
```

```
$ tar -xf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2
```

- Export the ARMGCC\_DIR variable using the following command:

```
$ export ARMGCC_DIR=<absolute-path>/toolchain-dir/gcc-arm-none-eabi-10-2020-q4-  
major/
```

- Add the toolchain path to the PATH environment variable using the command:

```
$ export PATH=$PATH:<absolute-path>/toolchain-dir/ gcc-arm-none-eabi-10-2020-  
q4-major/bin/
```

- Download and install *cmake* (source and binary distribution) using the [Link](#) for Linux system.
- Extract the source distribution and copy it to the /usr/share/ directory

```
$ tar -zxf cmake-3.19.1.tar.gz
```

```
$ sudo cp -rf cmake-3.19.1 /usr/share/cmake-3.19
```

- Extract the binary distribution and copy the binaries to the /usr/bin/ directory

```
$ tar -zxf cmake-3.19.1-Linux-x86_64.tar.gz
```

```
$ sudo cp cmake-3.19.1-Linux-x86_64/bin/* /usr/bin/
```

### 3.1.4.2 Build the application

This section provides the steps to build the application using the ARM GCC toolchain:

- Go to the `armgcc` directory of the application

```
$ cd <SDK-top-dir>/boards/evkmimxrt1060/wifi_examples/wifi_iperf/armgcc/
```

Modify the configuration for a wireless module

- By default, the project is configured to use the ***WIFI\_IW416\_BOARD\_AW\_AM510\_USD*** Wi-Fi module based on IW416 chipset. Modify the value in the file "***<SDK\_EXAMPLE\_PATH>/source/app\_config.h***" to match the module on your setup to include and compile the desired driver, components and application(s).
- Build the binary

```
$ sh build_flexspi_nor_debug.sh
[100%] Linking C executable flexspi_nor_debug/wifi_iperf.elf
[100%] Built target wifi_iperf.elf
```

- Generate `wifi_iperf.bin` using following command

```
arm-none-eabi-objcopy flexspi_nor_debug/wifi_iperf.elf -O binary
flexspi_nor_debug/wifi_iperf.bin
```

**NOTE:** Please refer to [MCUXSDKGSUG](#) for more details to debug the application using GDB.

### 3.1.4.3 Flash the application program (no debugging)

This section provides the steps to flash the binary on the i.MX RT board:

- Connect the board to the Linux host system. The board shows as a Mass storage device in the Linux host system.
- Copy the application binary (`wifi_iperf.bin`) to the Mass storage device and wait for the start of the binary download on the board.

```
$ sudo cp flexspi_nor_debug/wifi_iperf.bin /media/<user>/RT1060-EVK/
```

- The board stops showing as Mass storage device and appears again once the flash process has completed. If any error occurs during the flashing, the `FAIL.txt` file is generated and stored in the Mass storage device.
- To access the device using the serial console please refer to section 2.1.

```
OPTIONS: I18n
Compiled on Nov 15 2020, 20:18:47.
Port /dev/ttyACM0, 17:45:05
Press CTRL-A Z for help on special keys
=====
wifi iperf demo
=====
Initialize WLAN Driver
MAC Address: 70:66:55:7B:AC:84
```

**NOTE:** Please refer to section 3.1.7 to view the actual output on the console once the application is executed.

### 3.1.5 Run a demo with IAR IDE

This section provides the steps to open, configure, build, debug and run the demo example using IAR Embedded Workbench IDE. The instructions and illustrations refer to IAR version 9.10.2.

#### 3.1.5.1 Open the project workspace

To open the wifi\_iperf project available in the SDK, double-click the project workspace file named `wifi_iperf.eww` stored at the following location.

`<install_dir>\boards\evkmimxrt1060\wifi_examples\wifi_iperf\iar\wifi_iperf.eww`

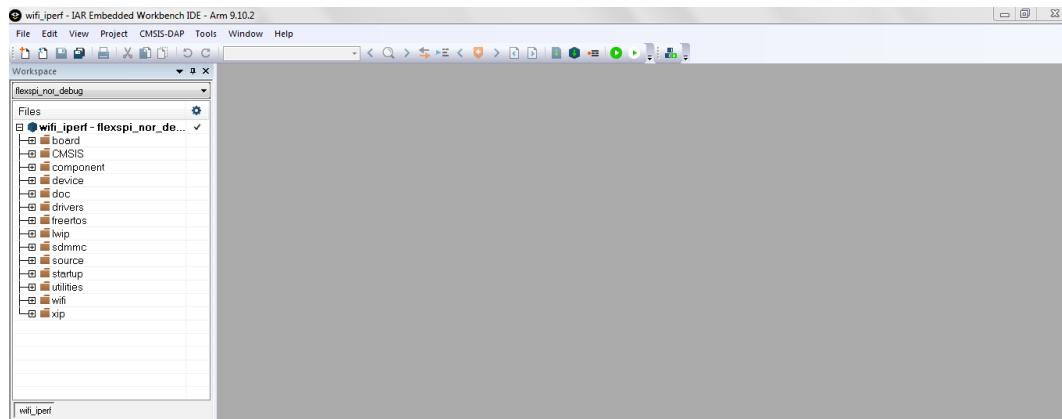
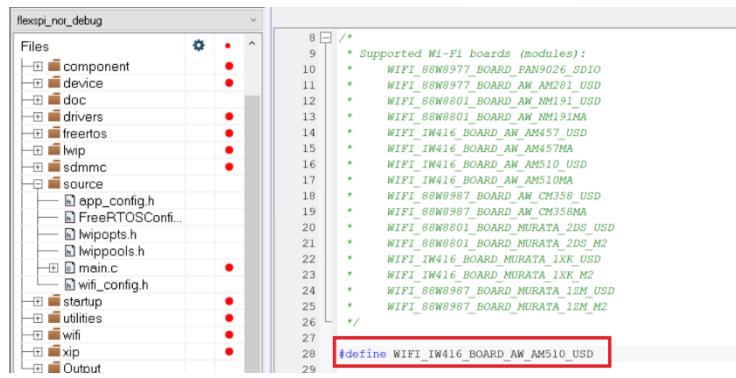


Figure 14: Open Project in IAR

**Note:** Please refer to section 3.1.1 for the application configurations.

### 3.1.5.2 Project Settings

- By default, the project is configured to use the **WIFI\_IW416\_BOARD\_AW\_AM510\_USD** Wi-Fi module based on IW416 chipset. Modify the value to match the module on your setup to include and compile the desired driver, components and application(s).
- The file "**app\_config.h**" from the source folder is used for the macro definitions
- Refer to Table 2 for the list of macros for Wi-Fi modules.



The screenshot shows the IAR Embedded Workbench interface. On the left, the file tree for the project "flexspi\_nor\_debug" is visible, containing directories like component, device, doc, drivers, freertos, lwip, sdmmc, and source, along with files such as app\_config.h, FreeRTOSConfig.h, main.c, and wifi\_config.h. On the right, the code editor displays a portion of the "app\_config.h" file. The code includes a multi-line comment listing supported Wi-Fi boards and modules, followed by a specific macro definition:  
/\*  
 \* Supported Wi-Fi boards (modules):  
 \* WIFI\_88W8977\_BOARD\_PAN9026\_SDIO  
 \* WIFI\_88W8977\_BOARD\_AW\_AM281\_USD  
 \* WIFI\_88W8801\_BOARD\_AW\_NM191\_USD  
 \* WIFI\_88W8801\_BOARD\_AW\_NM191MA  
 \* WIFI\_IW416\_BOARD\_AW\_AM457\_USD  
 \* WIFI\_IW416\_BOARD\_AW\_AM457MA  
 \* WIFI\_IW416\_BOARD\_AW\_AM510\_USD  
 \* WIFI\_IW416\_BOARD\_AW\_AM510MA  
 \* WIFI\_88W8987\_BOARD\_AW\_CM358\_USD  
 \* WIFI\_88W8987\_BOARD\_AW\_CM358MA  
 \* WIFI\_88W8801\_BOARD\_MURATA\_2DS\_USD  
 \* WIFI\_88W8801\_BOARD\_MURATA\_2DS\_M2  
 \* WIFI\_IW416\_BOARD\_MURATA\_1XK\_USD  
 \* WIFI\_IW416\_BOARD\_MURATA\_1XK\_M2  
 \* WIFI\_88W8987\_BOARD\_MURATA\_12M\_USD  
 \* WIFI\_88W8987\_BOARD\_MURATA\_12M\_M2  
 \*/  
**#define WIFI\_IW416\_BOARD\_AW\_AM510\_USD**

Figure 15: Wi-Fi Module Selection in IAR

### 3.1.5.3 Build the application

- To build the *wifi\_iperf* application, press the **Make** icon as illustrated below.

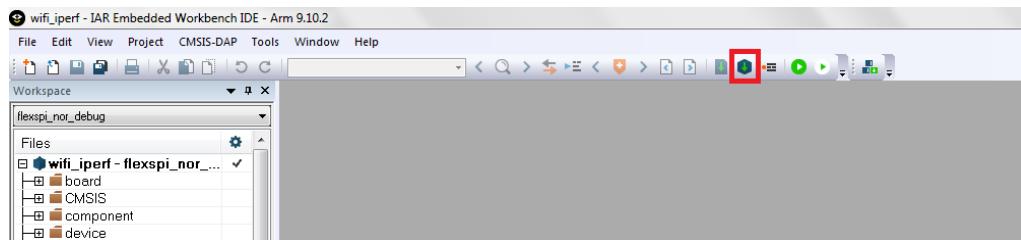


Figure 16: Application Build in IAR

- The details of the Build procedure are displayed in the **Messages** window of the **Build** tab.

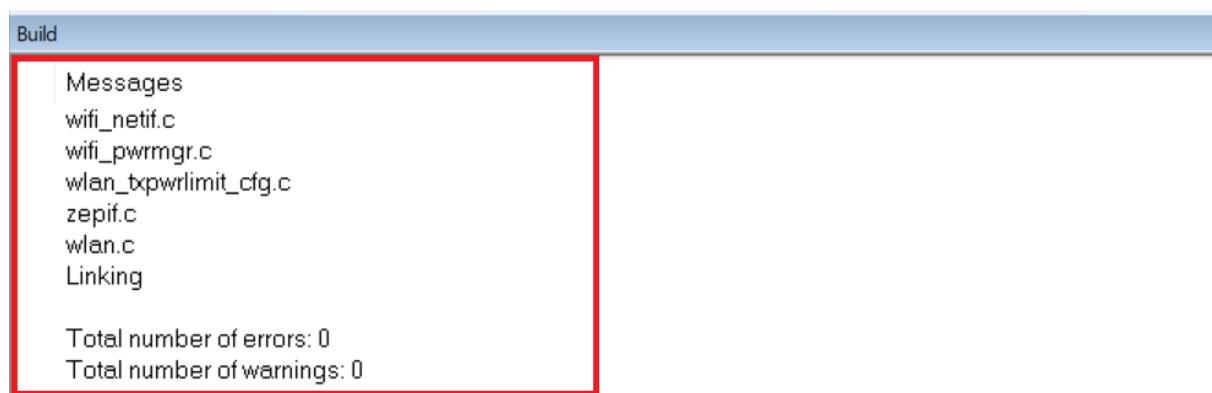


Figure 17: Build Message in IAR

### 3.1.5.4 Run the application in debug mode

The following steps describe how to run the application in debug mode.

The default debugger is **CMSIS-DAP**. However, if **CMSIS-DAP** is not selected, use the drop-down list to select it and press **OK**.

The selection of the debugger is a one-time configuration step that is not required for incremental debug.

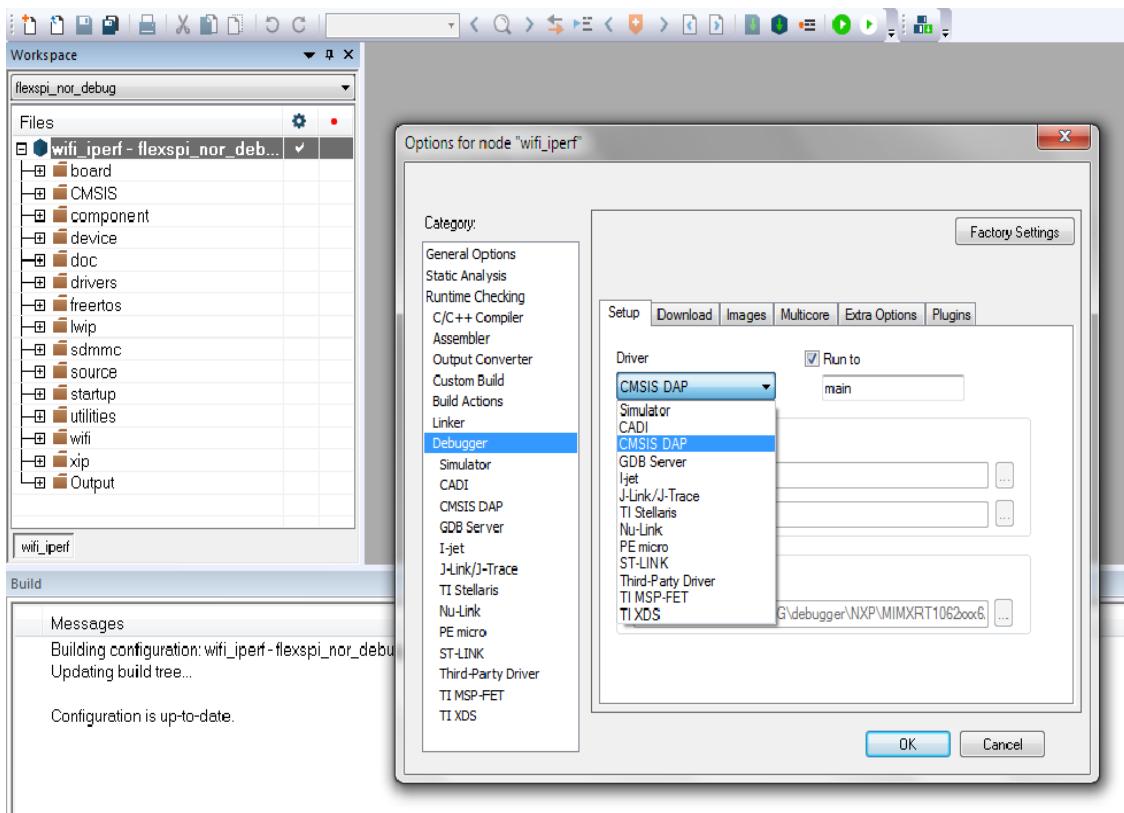


Figure 18: Debugger Selection in IAR

- To initiate the application debug, press the **Download and Debug** icon on the toolbar.



Figure 19: Initiate Debug in IAR

- The **Download and Debug** button is used to download the application to the target and set the program counter to the `main()` function of the application. Press **Go** to start the application. To debug the application, use the **Step Into**, **Step over** and **Step return** icons. To stop the debugging session, press the **Stop Debugging** icon.



Figure 20: Application Debugging in IAR

**Note:** Please refer to section 3.1.2 for iperf remote host setup.

### 3.1.5.5 Flash the application program (no debugging)

Please use the following steps to flash the application program.

- Go to **Project > Download** to flash the binary file. The **Download** menu provides the commands to flash the pre-built binary file and to erase the memory.

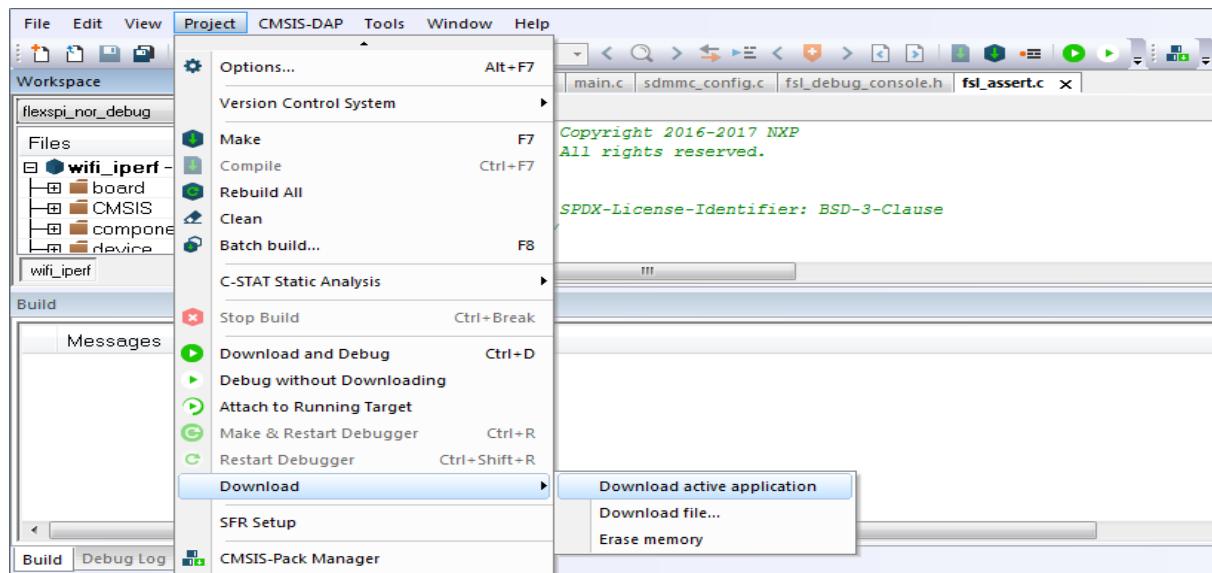


Figure 21: Binary Flashing in IAR

**NOTE:** Refer to section 3.1.7 to view the output on the console once the application is executed.

### 3.1.6 Run a demo using Keil MDK/μVision

This section details the steps to open, configure, build, debug and run demo example through Keil IDE. The Keil version used in the following instructions is V5.34.0.0.

**NOTE:** For Bluetooth demo applications Keil MDK/ μVision IDE is not supported.

#### 3.1.6.1 Install CMSIS device pack

Following the installation of the MDK tools, install the CMSIS device packs so you can use the debug functionality on your device. The CMSIS device packs include the memory map information, register definitions and flash programming algorithms. The following steps install the MIMXRT106x CMSIS pack.

- Click on the **Pack Installer** icon in the toolbar, look for **iMXRT1060\_MWP** in the **Packs** tab. Press **Install** in the **Action** column.

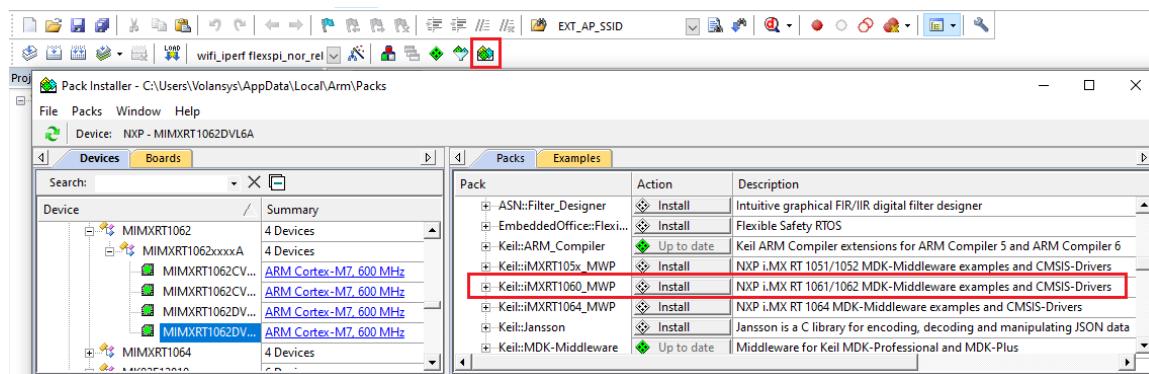


Figure 22: Install Packages using Pack Installer in Keil

- When the installation is complete, **Up to date** is displayed in the **Action** column. Verify that the **Board Support Pack (BSP)** and **Device Family Pack (DFP)** are both listed in the **Device > Packs** tab.

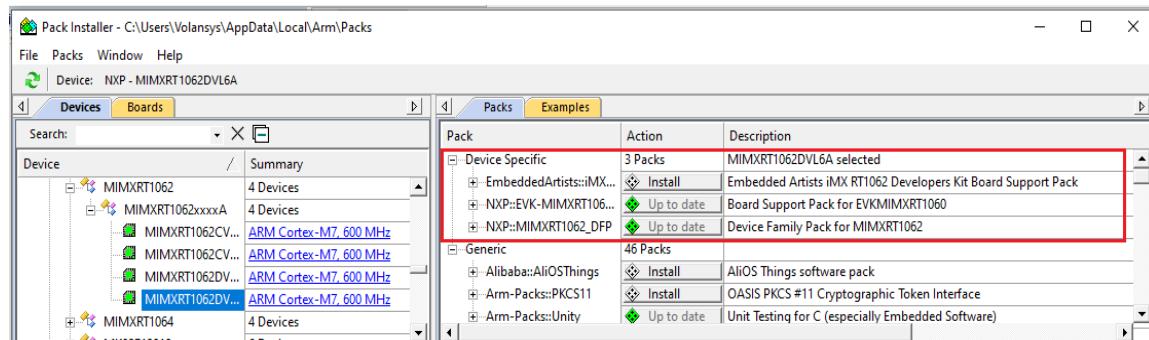


Figure 23: DFP Verification in Pack Installer in Keil

### 3.1.6.2 Open the project workspace

To open the `wifi_iperf` project: double-click the project workspace file `wifi_iperf.uvprojx` located at the following path: `<install_dir>\boards\evkmimxrt1060\wifi_examples\wifi_iperf\mdk\wifi_iperf.uvprojx`

**NOTE:** For a multi-project, use `wifi_iper.uvmpw` instead of `wifi_iperf.uvprojx`.

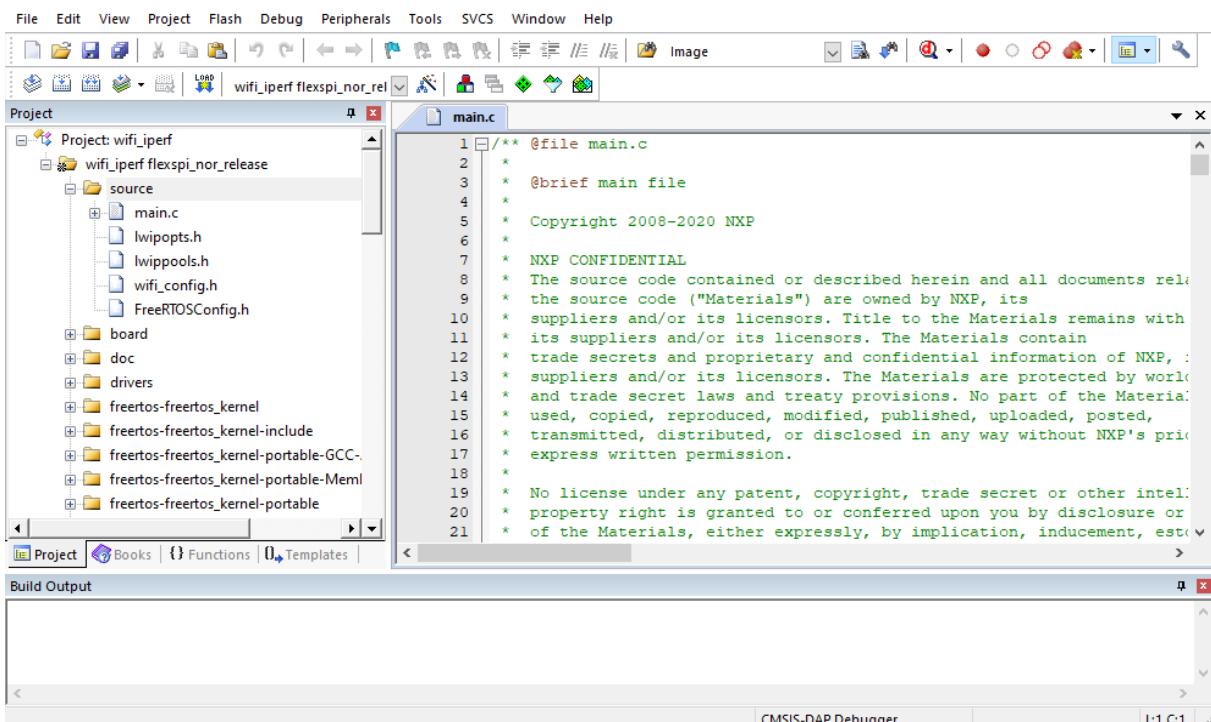


Figure 24: Open Project in Keil

**NOTE:** Please refer to section 3.1.2 for the application configuration.

### 3.1.6.3 Project Settings

- By default, the project is configured to use the **WIFI\_IW416\_BOARD\_AW\_AM510\_USD** Wi-Fi module based on IW416 chipset. Modify the value to match the module on your setup to include and compile the desired driver, components and application(s).
- The file "**app\_config.h**" from the source folder is used for the macro definitions
- Refer to Table 2 for the list of macros for Wi-Fi modules.

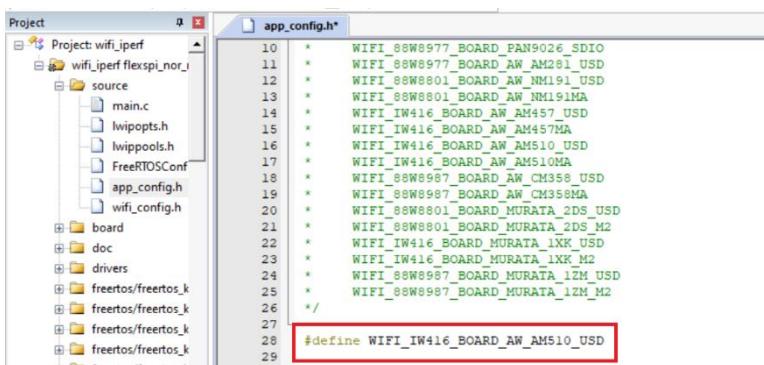


Figure 25: Wi-Fi Module Selection in Keil

### 3.1.6.4 Build the application

- To build the *wifi\_iperf* application, press the **Build** or **Rebuild** icons.

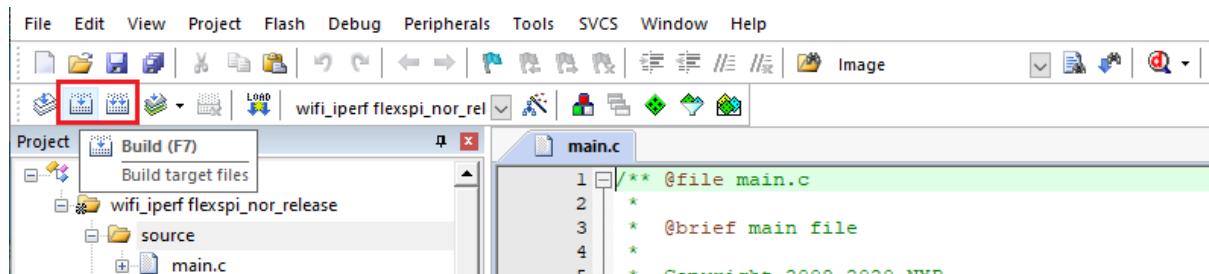


Figure 26: Application Build in Keil

- Verify the build progress in the **Build Output** window.

```
Build Output
assembling startup_MIMXRT1062.s...
compiling generic_list.c...
compiling evkmimxrt1060_flexspi_nor_config.c...
compiling fsl_flexspi_nor_boot.c...
compiling fsl_sdmmc_osa.c...
compiling fsl_os_abstraction_free_rtos.c...
linking...
Program Size: Code=152266 RO-data=535678 RW-data=79056 ZI-data=709556
"flexspi_nor_release\wifi_iperf.out" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:49
```

Figure 27: Build Message in Keil

### 3.1.6.5 Run the application in debug mode

Please refer to following steps to run the application in debug mode.

The default debugger is **CMSIS-DAP**. However, if **CMSIS-DAP** is not selected, use the **Options** icon in the toolbar and open the **Debug** tab, select the debugger in the drop-down list and press **OK**.

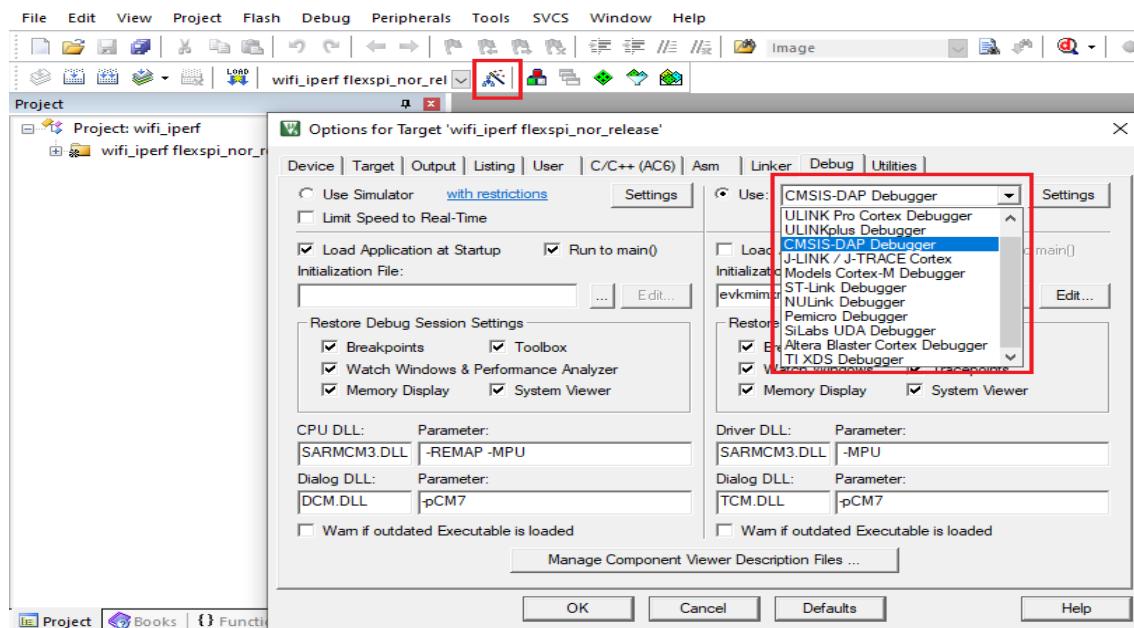


Figure 28: Debugger Selection in Keil

- To start the application debug, click on the **LOAD** icon to download the application on the board then click on the **Start/Stop Debug Session** icon in the toolbar.

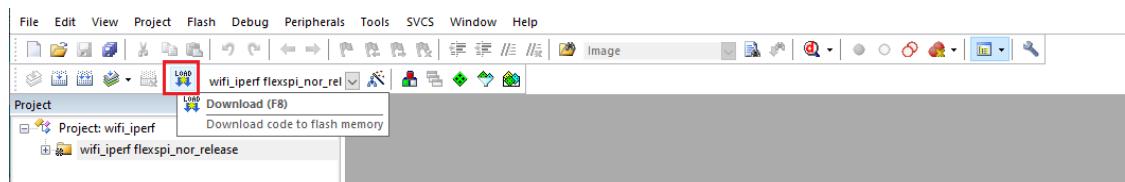


Figure 29: Load the application



Figure 30: Initiate Debug in Keil

- Click on the Start/Stop Debug Session icon to set the program counter to the main() function of the application.

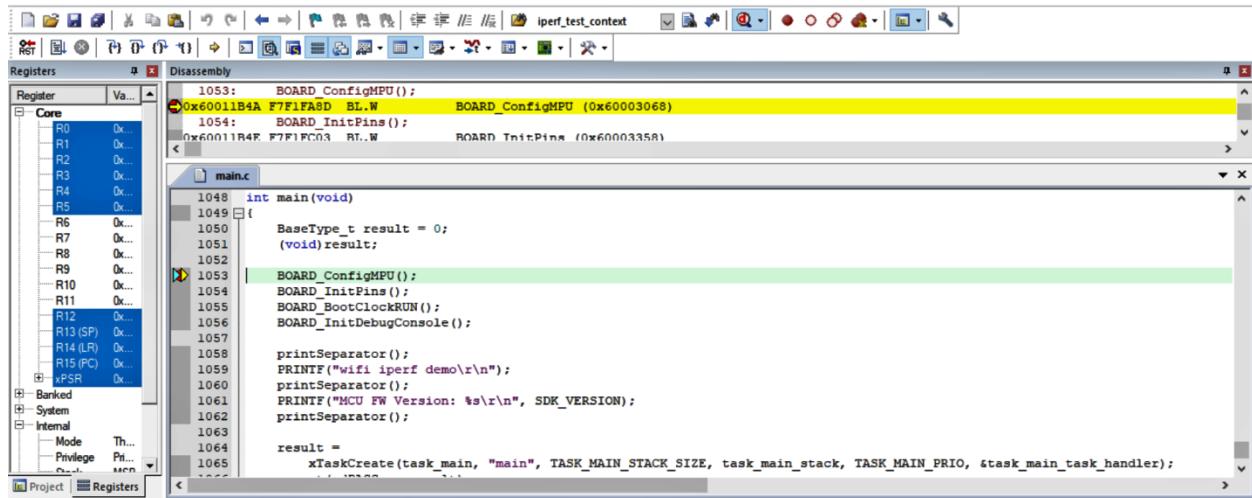


Figure 31: Application Debugging in Keil

- Press Run to start the application. Use Step, Step Over, Step Out and Run to Cursor Line icons in the toolbar to debug the application. To end the debugging session, click the Stop icon.



Figure 32: Application Debugging Features in Keil

**NOTE:** Please refer to the section 3.1.2 for iperf remote host setup.

### 3.1.6.6 Flash the application program (no debugging)

Please refer following steps to flash the application program.

- Click on the **Download** icon in the toolbar to flash the required binary file.

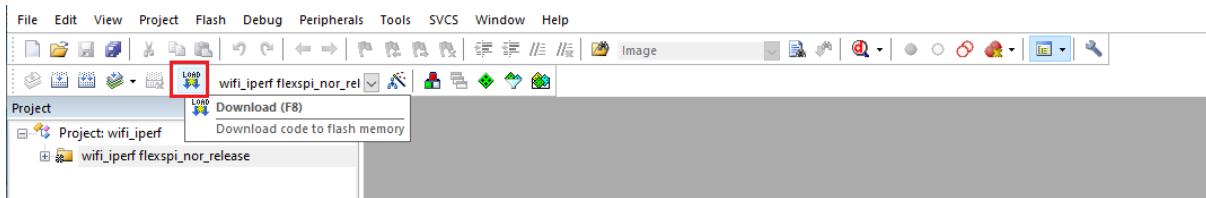


Figure 33: Binary Flashing in Keil

**NOTE:** Please refer to section 3.1.7 to view the output on the console once the application is executed.

### 3.1.7 *wifi\_iperf* Application Execution

#### 3.1.7.1 Start-up logs

The following logs can be observed on the console once the devices (i.MX RT1060 EVK board and NXP-based Wireless module) are up and running. Wi-Fi FW version logs shows that Wi-Fi module is ready for the operations.

```
=====
wifi iperf demo
=====
Initialize WLAN Driver
MAC Address: 00:13:43:7F:9C:9F
=====
For Soft AP demonstration
Start a Soft AP using option "A" in WPA2 security mode from menu
This also starts DHCP Server with IP 192.168.10.1, NETMASK 255.255.255.0
=====
For Station demonstration
Start an External AP with SSID as "nxp_wifi_demo" in Open mode
Start DHCP Server on External AP
Station network is configured with Dynamic address assignment
Application provides IPerf support
Set IPERF_SERVER_ADDRESS while using as IPerf Client
=====
A Start Soft AP
S Stop Soft AP
s Start Scan for external APs
c Connect to External AP (SSID='nxp_wifi_demo')
D Disconnect from External AP
I Enable IEEE PS on Station
i Disable IEEE PS on Station
d Enable Deep sleep on Station
e Disable Deep sleep on Station
p Print All Network info
P Print DHCP Server info
1 TCP server mode (RX only test)
2 TCP client mode (TX only test)
3 TCP client dual mode (TX and RX in parallel)
4 TCP client tradeoff mode (TX and RX sequentially)
5 UDP server mode (RX only test)
6 UDP client mode (TX only test)
7 UDP client dual mode (TX and RX in parallel)
8 UDP client tradeoff mode (TX and RX sequentially)
h Help (print this menu)
H Print extended help
[net] Initialized TCP/IP networking stack
=====
app_cb: WLAN: received event 10
=====
app_cb: WLAN initialized
=====
WLAN Driver Version    : vX.X.rXX.pX
WLAN Firmware Version : IW416-V0, RF878X, FP91, 16.91.21.p11.3, WPA2_CVE_FIX 1,
PVE_FIX 1
=====
```

### 3.1.7.2 Help command

The help command is used to get the list of commands available in the *wifi\_iperf* sample application.

```
Key 'h': Help (print this menu)
  A Start Soft AP
  S Stop Soft AP
  s Start Scan for external APs
  c Connect to External AP (SSID='nxp_wifi_demo')
  D Disconnect from External AP
  I Enable IEEE PS on Station
  i Disable IEEE PS on Station
  d Enable Deep sleep on Station
  e Disable Deep sleep on Station
  p Print All Network info
  P Print DHCP Server info
  1 TCP server mode (RX only test)
  2 TCP client mode (TX only test)
  3 TCP client dual mode (TX and RX in parallel)
  4 TCP client tradeoff mode (TX and RX sequentially)
  5 UDP server mode (RX only test)
  6 UDP client mode (TX only test)
  7 UDP client dual mode (TX and RX in parallel)
  8 UDP client tradeoff mode (TX and RX sequentially)
  h Help (print this menu)
  H Print extended help
```

### 3.1.7.3 Extended Help command

The extended help command lists the major Wi-Fi features and related default configurations which can be exercised through *wifi\_iperf* sample application commands.

```
Key 'H': Print extended help
For Soft AP demonstration
Start a Soft AP using option "A" in WPA2 security mode from menu
This also starts DHCP Server with IP 192.168.10.1, NETMASK 255.255.255.0
=====
For Station demonstration
Start an External AP with SSID as "nxp_wifi_demo" in Open mode
Start DHCP Server on External AP
Station network is configured with Dynamic address assignment
Application provides IPerf support
Set IPERF_SERVER_ADDRESS while using as IPerf Client
=====
```

### 3.1.7.4 Scan command

The scan command is used to scan the visible access points.

```
Key 's': Start Scan for external APs
Scan scheduled...
3 networks found:
C0:CB:38:9C:51:4F "nxp_wifi_demo"
    channel: 1
    rssi: -38 dBm
    security: OPEN
    WMM: YES
38:E6:0A:C6:1A:EC "nxp"
    channel: 6
    rssi: -64 dBm
    security: WPA2
    WMM: YES
6E:C4:D5:AE:D1:2F "Nokia 1"
    channel: 13
    rssi: -91 dBm
    security: WPA2
    WMM: YES
```

### 3.1.7.5 Start Soft AP

The following command is used to start the AP with the default available configurations mentioned in the extended help command.

```
Key 'A': Start Soft AP
=====
app_cb: WLAN: received event 14
=====
app_cb: WLAN: UAP Started
=====
Soft AP "NXP_Soft_AP" Started successfully
=====
DHCP Server started successfully
```

Connect the wireless client to the AP just created, NXP\_Soft\_AP. The logs below can be observed once the client is associated successfully.

```
Client => 38:E6:0A:C6:1A:EC Associated with Soft AP
```

The following command is used to get IP and MAC regarding information of clients which connected to the soft AP.

```
Key 'P': Print DHCP Server info
DHCP Server Lease Duration : 86400 seconds
Client IP      Client MAC
192.168.10.2   38:E6:0A:C6:1A:EC
```

### 3.1.7.6 iPerf Server/Client Commands with AP Mode

iPerf default configurations are mentioned in the section 3.1.1.2. For iperf remote host commands refer Table 6, Table 7 and Table 8 from section 3.1.2 for Windows, Linux and Mobile application respectively.

#### iPerf server

- TCP Server

```
Key '1': TCP server mode (RX only test)
New TCP client (settings flags 0x0)

-----
TCP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 58712
Bytes Transferred XXXX
```

```
Duration (ms) 10250
Bandwidth (Mbitpsec) XX
```

- UDP Server

```
Key '5': UDP server mode (RX only test)
New UDP client (settings flags 0x0)
```

```
Sending report back to client (0x80).
```

```
Jitter X.XXX,
Lost XX/XXX datagrams, OoO XX
```

```
-----
UDP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 60146
Bytes Transferred XXXX
Duration (ms) 10031
Bandwidth (Mbitpsec) XX
```

#### iPerf client

IPERF\_SERVER\_ADDRESS in *main.c* file should be set to the appropriate server IP to run the iperf client.

- TCP Client (Tx Only)

```
Key '2': TCP client mode (TX only test)
-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49153
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10001
Bandwidth (Mbitpsec) XX
```

- TCP Client (Tx and Rx Parallel)

```
Key '3': TCP client dual mode (TX and RX in parallel)
New TCP client (settings flags 0x30313233)
```

```
-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49153
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10000
Bandwidth (Mbitpsec) XX
```

```
-----
TCP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 35576
Bytes Transferred XXXX
Duration (ms) 10211
Bandwidth (Mbitpsec) XX
```

- TCP Client (Tx and Rx sequentially)

```
Key '4': TCP client tradeoff mode (TX and RX sequentially)
```

```
-----  
TCP_DONE_CLIENT (TX)  
Local address : 192.168.10.1 Port 49153  
Remote address : 192.168.10.2 Port 5001  
Bytes Transferred 14278824  
Duration (ms) XXXX  
Bandwidth (Mbitpsec) XX  
New TCP client (settings flags 0x30313233)
```

```
-----  
TCP_DONE_SERVER (RX)  
Local address : 192.168.10.1 Port 5001  
Remote address : 192.168.10.2 Port 35580  
Bytes Transferred XXXX  
Duration (ms) 10226  
Bandwidth (Mbitpsec) XX
```

- UDP Client (Tx Only)

```
Key '6': UDP client mode (TX only test)  
Ideal frame delay: 112 us
```

```
Send 8 frame(s) once per 1000 us
```

```
Received report from server (0x80000000).
```

```
Jitter XX.XXXX,  
Lost XX/XXXX datagrams, OoO XX
```

```
-----  
UDP_DONE_CLIENT (TX)  
Local address : 192.168.10.1 Port 49155  
Remote address : 192.168.10.2 Port 5001  
Bytes Transferred XXXX  
Duration (ms) 10019  
Bandwidth (Mbitpsec) XX
```

- UDP Client (Tx and Rx Parallel)

```
Key '7': UDP client dual mode (TX and RX in parallel)
Dualtest port: 5001

Ideal frame delay: 112 us

Send 8 frame(s) once per 1000 us

New UDP client (settings flags 0x32333435)

Received report from server (0x80000000).

Jitter X.XXXX,
Lost XX/XXXX datagrams, OoO 0

-----
UDP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49155
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10199
Bandwidth (Mbitpsec) XX

Sending report back to client (0x80).

Jitter X.XXXX,
Lost XX/XXXX datagrams, OoO 337

-----
UDP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 45973
Bytes Transferred XXXX
Duration (ms) 9325
Bandwidth (Mbitpsec) XX
```

- UDP Client (Tx and Rx Sequentially)

```
Key '8': UDP client tradeoff mode (TX and RX sequentially)
Dualtest port: 5001
Ideal frame delay: 112 us
Send 8 frame(s) once per 1000 us
Received report from server (0x80000000).
Jitter X.XXX,
Lost XX/XXXX datagrams, OoO XX

-----
UDP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49155
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10015
Bandwidth (Mbitpsec) XX
New UDP client (settings flags 0x32333435)
Sending report back to client (0x80).
Jitter X.XXX,
Lost XX/XXXX datagrams, OoO XX

-----
UDP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 54244
Bytes Transferred XXXX
```

```
Duration (ms) 10053
Bandwidth (Mbitpsec) XX
```

### 3.1.7.7 Stop Soft AP

The following command is used to stop AP.

```
Key 'S': Stop Soft AP
=====
app_cb: WLAN: received event 19
=====
app_cb: WLAN: UAP Stopped
=====
Soft AP "NXP_Soft_AP" Stoped successfully
=====
DHCP Server stopped successfully
=====
```

### 3.1.7.8 Station mode (connect to AP)

The following command is used to connect device to AP.

```
Key 'c': Connect to External AP (SSID='nxp_wifi_demo')
Connecting to nxp_wifi_demo ....
=====
app_cb: WLAN: received event 0
=====
app_cb: WLAN: connected to network
Connected to following BSS:
SSID = [nxp_wifi_demo], IP = [192.168.43.35]
```

#### Enable/disable IEEE PS

The following commands are used to enable/disable IEEE PS (Power Save) on station.

```
Key 'I': Enable IEEE PS on Station
=====
app_cb: WLAN: received event 12
=====
app_cb: WLAN: PS_ENTER
```

```
Key 'i': Disable IEEE PS on Station
=====
app_cb: WLAN: received event 13
=====
app_cb: WLAN: PS EXIT
```

#### Enable/disable Deep Sleep

The following commands are used to enable/disable deep sleep feature on station.

```
Key 'd': Enable Deep sleep on Station
Key 'e': Disable Deep sleep on Station
```

### 3.1.7.9 iPerf Server/Client Commands with STA Mode

iPerf default configurations are mentioned in the section 3.1.1.2. For iperf remote host commands refer Table 6, Table 7 and Table 8 from section 3.1.2 for Windows, Linux and Mobile application respectively.

#### iPerf server

- TCP Server

```
Key '1': TCP server mode (RX only test)
New TCP client (settings flags 0x0)

-----
TCP_DONE_SERVER (RX)
Local address : 192.168.1.18 Port 5001
Remote address : 192.168.1.2 Port 57060
```

```
Bytes Transferred XXXX
Duration (ms) 10165
Bandwidth (Mbitpsec) XX
```

- UDP Server

```
Key '5': UDP server mode (RX only test)
New UDP client (settings flags 0x0)
Sending report back to client (0x80).
Jitter X.XXX,
Lost XX/XXXX datagrams, OoO XX
-----
UDP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 59906
Bytes Transferred XXXX
Duration (ms) 10036
Bandwidth (Mbitpsec) XX
```

#### iPerf client

IPERF\_SERVER\_ADDRESS in *main.c* file should be set to the appropriate server IP to run the iperf client.

- TCP Client (Tx Only)

```
Key '2': TCP client mode (TX only test)
-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.1.18 Port 49153
Remote address : 192.168.1.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10002
Bandwidth (Mbitpsec) XX
```

- TCP Client (Tx and Rx Parallel)

```
Key '3': TCP client dual mode (TX and RX in parallel)
New TCP client (settings flags 0x30313233)
```

```
-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.1.18 Port 49153
Remote address : 192.168.1.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10000
Bandwidth (Mbitpsec) XX
```

```
-----
TCP_DONE_SERVER (RX)
Local address : 192.168.1.18 Port 5001
Remote address : 192.168.1.2 Port 57062
Bytes Transferred XXXX
Duration (ms) 10162
Bandwidth (Mbitpsec) XX
```

- TCP Client (Tx and Rx sequentially)

```
Key '4': TCP client tradeoff mode (TX and RX sequentially)
```

```
-----  
TCP_DONE_CLIENT (TX)  
Local address : 192.168.1.18 Port 49153  
Remote address : 192.168.1.2 Port 5001  
Bytes Transferred XXXX  
Duration (ms) 10001  
Bandwidth (Mbitpsec) XX
```

```
New TCP client (settings flags 0x30313233)
```

```
-----  
TCP_DONE_SERVER (RX)  
Local address : 192.168.1.18 Port 5001  
Remote address : 192.168.1.2 Port 57064  
Bytes Transferred XXXX  
Duration (ms) 10168  
Bandwidth (Mbitpsec) XX
```

- UDP Client (Tx Only)

```
Key '6': UDP client mode (TX only test)  
Ideal frame delay: 112 us  
Send 8 frame(s) once per 1000 us  
Received report from server (0x80000000).  
Jitter X.XXX,  
Lost XX/XXXX datagrams, OoO XX
```

```
-----  
UDP_DONE_CLIENT (TX)  
Local address : 192.168.1.18 Port 49154  
Remote address : 192.168.1.2 Port 5001  
Bytes Transferred XXXX  
Duration (ms) 10021  
Bandwidth (Mbitpsec) XX
```

- UDP Client (Tx and Rx Parallel)

```
Key '7': UDP client dual mode (TX and RX in parallel)  
Dualtest port: 5001
```

```
Ideal frame delay: 112 us
```

```
Send 8 frame(s) once per 1000 us
```

```
New UDP client (settings flags 0x32333435)
```

```
Received report from server (0x80000000).
```

```
Jitter X.XXXX,  
Lost XX/XXXX datagrams, OoO XX
```

```
Sending report back to client (0x80).
```

```
Jitter X.XXXX,  
Lost XX/XXXX datagrams, OoO XX
```

```
-----  
UDP_DONE_CLIENT (TX)  
Local address : 192.168.1.18 Port 49154  
Remote address : 192.168.1.2 Port 5001  
Bytes Transferred XXXX  
Duration (ms) 10069  
Bandwidth (Mbitpsec) XX
```

```
-----  
UDP_DONE_SERVER (RX)  
Local address : 192.168.1.18 Port 5001  
Remote address : 192.168.1.2 Port 48285  
Bytes Transferred XXXX  
Duration (ms) 10073  
Bandwidth (Mbitpsec) XX
```

- UDP Client (Tx and Rx Sequentially)

```
Key '8': UDP client tradeoff mode (TX and RX sequentially)  
Dualtest port: 5001
```

```
Ideal frame delay: 112 us
```

```
Send 8 frame(s) once per 1000 us
```

```
Received report from server (0x80000000).
```

```
Jitter X.XXXX,  
Lost XX/XXXX datagrams, OoO XX
```

```
-----  
UDP_DONE_CLIENT (TX)  
Local address : 192.168.1.18 Port 49154  
Remote address : 192.168.1.2 Port 5001  
Bytes Transferred XXXX  
Duration (ms) 10017  
Bandwidth (Mbitpsec) XX
```

```
New UDP client (settings flags 0x32333435)
```

```
Sending report back to client (0x80).
```

```
Jitter X.XXXX,  
Lost XX/XXXX datagrams, OoO XX
```

```
-----  
UDP_DONE_SERVER (RX)  
Local address : 192.168.1.18 Port 5001  
Remote address : 192.168.1.2 Port 49610  
Bytes Transferred XXXX  
Duration (ms) 10032  
Bandwidth (Mbitpsec) XX
```

### 3.1.7.10 Station disconnection (from AP)

The following command is used to disconnect station.

```
Key 'D': Disconnect from External AP  
=====  
app_cb: WLAN: received event 9  
=====  
app_cb: disconnected
```

### 3.1.7.11 Print network information

The following command is used to print network information either for AP or STA.

```
Key 'p': Print All Network info
Station connected to:
"sta_network"
    SSID: nxp_wifi_demo
    BSSID: C0:CB:38:9C:51:4F
    channel: 1
    role: Infra
    security: none

    IPv4 Address
    address: DHCP
        IP:          192.168.1.18
        gateway:    192.168.1.2
        netmask:    255.255.255.0
        dns1:       192.168.1.2
        dns2:       0.0.0.0

uAP started as:
"uap-network"
    SSID: NXP_Soft_AP
    BSSID: 00:50:43:21:F1:13
    channel: 1
    role: uAP
    security: WPA2

    IPv4 Address
    address: STATIC
        IP:          192.168.10.1
        gateway:    192.168.10.1
        netmask:    255.255.255.0
        dns1:       192.168.1.2
        dns2:       0.0.0.0
        dns1:       192.168.1.2
        dns2:       0.0.0.0
```

## 3.2 wifi\_setup Sample Application

This section describes *wifi\_setup* sample application and its configuration along with the application execution. The *wifi\_setup* sample application is used to demonstrate a Wi-Fi Station mode that connects to AP and starts pinging the IP address provided by the user.

### Wi-Fi Features:

Table 9: wifi\_setup Application Features

Features	Details
Wi-Fi	Wi-Fi Scan Wi-Fi Station mode

### Wi-Fi Configurations:

Some of the Wi-Fi features and feature related macros that user can configure based on requirement are listed in below table along with source file name.

Table 10: Wi-Fi Configurations

Feature	Macro definition	Default value	File name	Details
Wi-Fi STA	AP_SSID AP_PASSPHRASE PING_ADDR	"SSID" "PASSWD" "8.8.8.8"	wifi_setup.c	Default configuration (of targeted AP) to start station mode and ping the ip address using given sample application. It can be modified by changing the macro value.

### 3.2.1 wifi\_setup Application Execution

Please refer to the previous sections 3.1.3, 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console tool setup.

### 3.2.1.1 Run the application

The following logs can be observed on the console once the devices (i.MX RT1060 EVK board and NXP-based wireless module) are up and running. The *Wi-Fi FW version log* shows that the Wi-Fi module is ready to operate.

```
Wifi setup example
Initialize WLAN Driver
MAC Address: 20:4E:F6:EC:1F:3F
[net] Initialized TCP/IP networking stack
app_cb: WLAN: received event 10
app_cb: WLAN initialized
WLAN Driver Version : vX.X.rXX.pX
WLAN Firmware Version : IW416-V0, RF878X, FP91, 16.91.21.p11.3, WPA2_CVE_FIX 1,
PVE_FIX 1
```

Once Wi-Fi module is initialized it'll try to scan nearby networks.

```
Scan scheduled...
9 networks found:
28:3B:82:3F:79:F6 "PS"
    channel: 3
    rssi: -87 dBm
    security: WPA/WPA2 Mixed
    WMM: YES
38:E6:0A:C6:1A:EC "nxp"
    channel: 6
    rssi: -70 dBm
    security: WPA2
    WMM: YES
18:0F:76:C7:5A:46 "Testlab"
    channel: 9
    rssi: -85 dBm
    security: WPA/WPA2 Mixed
    WMM: YES
D4:68:4D:83:58:48 "test_2.4G"
    channel: 13
    rssi: -89 dBm
    security: WPA2
    WMM: YES
...
...
```

It will now try to connect to the defined network.

```
Connecting to nxp ....
app_cb: WLAN: received event 0
Connected to [nxp]
```

Once the connection is established successfully, it will continuously ping the IP and print the received response time in ms(millisecond).

```
ping: send
8.8.8.8

ping: recv
8.8.8.8
278 ms

ping: send
8.8.8.8

ping: recv
8.8.8.8
```

```
173 ms
ping: send
8.8.8.8
...
...
```

### 3.3 wifi\_cli Sample Application

This section describes the *wifi\_cli* application to demonstrate the CLI support to handle and enable Wi-Fi configuration for the features including scan the visible access points, create and configure the access point and connection with the access point. The CLI module in the application allows users to add CLIs in the application. In this sample application Wi-Fi connection manager CLIs are available.

Table 11: wifi\_cli Application Features

Features	Details
Wi-Fi	Wi-Fi Soft AP mode Wi-Fi Station mode Wi-Fi Scan Wi-Fi IEEEPS power saving mode Wi-Fi deep-sleep power saving mode
IPerf	TCP Client and Server TCP Client dual mode (Tx and Rx in simultaneous) TCP Client trade-off mode (Tx and Rx individual)  UDP Client and Server UDP Client dual mode (Tx and Rx in simultaneous) UDP Client trade-off mode (Tx and Rx individual)

#### 3.3.1 wifi\_cli Application Execution

Please refer to the previous sections 3.1.3, 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console tool setup.

### 3.3.1.1 Run the application

This section describes the available Wi-Fi commands. The application starts with the welcome message, press **Enter** for the command prompt.

```
=====
wifi cli demo
=====
Initialize CLI
=====
Initialize WLAN Driver
=====
MAC Address: 00:13:43:7F:9C:9F
[net] Initialized TCP/IP networking stack
=====
app_cb: WLAN: received event 10
=====
app_cb: WLAN initialized
=====
WLAN CLIs are initialized
=====
CLIs Available:
=====

help
wlan-version
wlan-mac
wlan-scan
wlan-scan-opt ssid <ssid> bssid ...
wlan-add <profile_name> ssid <ssid> bssid...
wlan-remove <profile_name>
wlan-list
wlan-connect <profile_name>
wlan-start-network <profile_name>
wlan-stop-network
wlan-disconnect
wlan-stat
wlan-info
wlan-address
wlan-get-uap-channel
wlan-get-uap-sta-list
wlan-ieee-ps <0/1>
wlan-deep-sleep-ps <0/1>
wlan-send-hostcmd
ping [-s <packet_size>] [-c <packet_count>] [-W <timeout in sec>] <ip_address>
iperf [-s|-c <host>|-a|-h] [options]
dhcp-stat
=====
```

The following steps describe how to configure the Wi-Fi features.

#### Wi-Fi scan

The following command is used to scan the visible access points:

```
# wlan-scan
Scan scheduled...

# 1 network found:
38:E6:0A:C6:1A:EC "nxp" Infra
    channel: 11
    rssi: -57 dBm
    security: WPA2
    WMM: YES

# wlan-scan-opt ssid nxp
Scan for ssid "nxp" scheduled...

# 1 network found:
38:E6:0A:C6:1A:EC "nxp" Infra
    channel: 11
    rssi: -54 dBm
    security: WPA2
    WMM: YES
```

Before adding a network profile for Station mode and AP mode, please check command usage.

```
# wlan-add
Usage:
For Station interface
  For DHCP IP Address assignment:
    wlan-add <profile_name> ssid <ssid> [wpa2 <secret>]
      If using WPA2 security, set the PMF configuration if required.
    wlan-add <profile_name> ssid <ssid> [wpa3 sae <secret> mfpc <1> mfpr <0/1>]
      If using WPA3 SAE security, always set the PMF configuration.
  For static IP address assignment:
    wlan-add <profile_name> ssid <ssid>
      ip:<ip_addr>,<gateway_ip>,<netmask>
      [bssid <bssid>] [channel <channel number>]
      [wpa2 <secret>]
For Micro-AP interface
  wlan-add <profile_name> ssid <ssid>
    ip:<ip_addr>,<gateway_ip>,<netmask>
    role uap [bssid <bssid>]
    [channel <channelnumber>]
    [wpa2 <secret>] [wpa3 sae <secret>]
    [mfpc <0/1>] [mfpr <0/1>]
Error: invalid number of arguments
```

### Wi-Fi station mode

#### WPA2 Security

Use the following command to add the network profile to configure the device in station mode. Provide any profile name as well as use your AP's SSID and Passphrase in argument shown below:

```
# wlan-add abc ssid nxp wpa2 1234567890
Added "abc"
```

Connect to the AP network using the saved network profile:

```
# wlan-connect abc
Connecting to network...
Use 'wlan-stat' for current connection status.

=====
app_cb: WLAN: received event 0
=====
app_cb: WLAN: connected to network
Connected to following BSS:
SSID = [nxp], IP = [192.168.43.35]
```

**NOTE:** Once connected to the AP the console output will show Client successfully connected to AP with ssid "nxp" and got ip address "192.168.43.35" from AP.

Disconnect from the AP network profile:

```
# wlan-disconnect

=====
app_cb: WLAN: received event 9
=====
app_cb: disconnected
```

Remove the saved network profile:

```
# wlan-remove abc
Removed "abc"
```

**WPA3 Security**

```
# wlan-add nxp_test_1 ssid wpa3r3 wpa3 sae 12345678 mfpc 1 mfpr 1  
Added "nxp_test_1"
```

Connect to the AP network using the saved network profile:

```
# wlan-connect nxp_test_1  
Connecting to network...  
Use 'wlan-stat' for current connection status.  
  
=====  
app_cb: WLAN: received event 0  
=====  
app_cb: WLAN: connected to network  
Connected to following BSS:  
SSID = [wpa3r3], IP = [192.168.10.2]
```

**NOTE:** Once connected to the AP the console output will show Client successfully connected to AP with ssid "wpa3r3" and got ip address "192.168.10.2" from AP.

Disconnect from the AP network profile:

```
# wlan-disconnect  
  
=====  
app_cb: WLAN: received event 9  
=====  
app_cb: disconnected
```

Remove the saved network profile:

```
# wlan-remove nxp_test_1  
Removed "nxp_test_1"
```

**Wi-Fi AP mode**

Use the following command to add the network profile to configure the device in AP mode. Use your AP's SSID, IP details, role, channel and security (Passphrase if applicable) in argument shown below.

```
# wlan-add xyz ssid NXPAP ip:192.168.10.1,192.168.10.1,255.255.255.0 role uap  
channel 6 wpa2 12345678
```

```
Added "xyz"
```

Start the AP using saved network profile:

```
# wlan-start-network xyz  
[wlcm] Warn: NOTE: uAP will automatically switch to the channel that station is  
on.
```

```
=====  
app_cb: WLAN: received event 14  
=====  
app_cb: WLAN: UAP Started  
=====  
Soft AP "NXPAP" started successfully  
=====  
DHCP Server started successfully  
=====
```

Connect the wireless client to the AP just created, NXPAP. The logs below can be observed once the Client is associated successfully:

```
Client => 38:E6:0A:C6:1A:EC Associated with Soft AP
```

Get the associated clients list:

```
# wlan-get-uap-sta-list  
Number of STA = 1  
  
STA 1 information:  
=====  
MAC Address: 38:E6:0A:C6:1A:EC  
Power mfg status: power save  
Rssi : -58 dBm
```

Get the IP and MAC information for the associated clients:

```
# dhcp-stat  
DHCP Server Lease Duration : 86400 seconds  
Client IP      Client MAC  
192.168.10.2   38:E6:0A:C6:1A:EC
```

Stop the AP:

```
# wlan-stop-network  
=====  
app_cb: WLAN: received event 19  
=====  
app_cb: WLAN: UAP Stopped  
=====  
Soft AP "NXPAP" stopped successfully  
=====  
DHCP Server stopped successfully  
=====
```

### 3.3.1.2 IPerf Server/Client

The following commands are used for IPerf Initialization:

**IPerf Usage :**

```
# iperf
Incorrect usage
Usage:
    iperf [-s|-c <host>|-a] [options]
    iperf [-h]

    Client/Server:
        -u          use UDP rather than TCP
        -B <host>   bind to <host>, a multicast address
        -a          abort ongoing iperf session
    Server specific:
        -s          run in server mode
    Client specific:
        -c <host>   run in client mode, connecting to <host>
        -d          Do a bidirectional test simultaneously
        -r          Do a bidirectional test individually
        -t #        time in seconds to transmit for (default 10 secs)
```

**NOTE:** For iperf Windows, Linux and Mobile application commands refer Table 6, Table 7 and Table 8 respectively from section 3.1.2

Start IPerf server:

```
# iperf -s

# IPERF initialization successful
New TCP client (settings flags 0x0)

-----
TCP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 36874
Bytes Transferred XXXX
Duration (ms) 10130
Bandwidth (Mbitpsec) XX
```

Start IPerf Client (Tx Only):

```
# iperf -c 192.168.10.2

# IPERF initialization successful

-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49153
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10001
Bandwidth (Mbitpsec) XX
```

Start IPerf Client (Tx and Rx simultaneous):

```
# iperf -c 192.168.10.2 -d

IPERF initialization successful
New TCP client (settings flags 0x30313233)

-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49154
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10001
Bandwidth (Mbitpsec) XX

-----
TCP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 36876
Bytes Transferred XXXX
Duration (ms) 10138
Bandwidth (Mbitpsec) XX
```

Start IPerf Client (Tx and Rx individual):

```
# iperf -c 192.168.10.2 -r

# IPERF initialization successful

-----
TCP_DONE_CLIENT (TX)
Local address : 192.168.10.1 Port 49155
Remote address : 192.168.10.2 Port 5001
Bytes Transferred XXXX
Duration (ms) 10001
Bandwidth (Mbitpsec) XX

New TCP client (settings flags 0x30313233)

-----
TCP_DONE_SERVER (RX)
Local address : 192.168.10.1 Port 5001
Remote address : 192.168.10.2 Port 36878
Bytes Transferred XXXX
Duration (ms) 10095
Bandwidth (Mbitpsec) XX
```

### 3.3.1.3 Wi-Fi Power Save

The following commands are used to save Wi-Fi power in different modes:

- IEEE Power Save

For IEEEPS mode Wi-Fi station should be connected with AP.

IEEEPS Usage:

```
# wlan-ieee-ps
Usage: wlan-ieee-ps <0/1>
Error: Specify 0 to Disable or 1 to Enable
```

Enable IEEEPS:

```
# wlan-ieee-ps 1
Turned on IEEE Power Save mode
=====
app_cb: WLAN: received event 12
=====
app_cb: WLAN: PS_ENTER
```

Disable IEEEPS:

```
# wlan-ieee-ps 0
Turned off IEEE Power Save mode
=====
app_cb: WLAN: received event 13
=====
app_cb: WLAN: PS_EXIT
```

- DeepSleep

For DeepSleep mode Wi-Fi should be in disconnected state otherwise it will not enable the deepsleep.

Check Wi-Fi connection:

```
# wlan-info
Station not connected
uAP not started
```

DepSleep Usage:

```
# wlan-deep-sleep-ps
Usage: wlan-deep-sleep-ps <0/1>
Error: Specify 0 to Disable or 1 to Enable
```

Enable DeepSleep:

```
# wlan-deep-sleep-ps 1
Turned on Deep Sleep Power Save mode
=====
app_cb: WLAN: received event 12
=====
app_cb: WLAN: PS_ENTER
```

Disable DeepSleep:

```
# wlan-deep-sleep-ps 0
Turned off Deep Sleep Power Save mode
# =====
app_cb: WLAN: received event 13
=====
app_cb: WLAN: PS_EXIT
```

### 3.3.1.4 Other useful CLI commands

Use the other commands to get the Wi-Fi information, driver version, firmware version, list of the networks and other information.

Get the Wi-Fi information:

```
# wlan-info
Station connected to:
"abc"
    SSID: nxp
    BSSID: 38:E6:0A:C6:1A:EC
    channel: 11
    role: Infra
    security: WPA2

    IPv4 Address
    address: DHCP
        IP:          192.168.43.156
        gateway:     192.168.43.233
        netmask:     255.255.255.0
        dns1:        192.168.43.233
        dns2:        0.0.0.0

uAP started as:
"xyz"
    SSID: NXPAP
    BSSID: 00:13:43:6A:5A:ED
    channel: 11
    role: uAP
    security: WPA2

    IPv4 Address
    address: STATIC
        IP:          192.168.10.1
        gateway:     192.168.10.1
        netmask:     255.255.255.0
        dns1:        192.168.43.233
        dns2:        0.0.0.0
```

Get the Wi-Fi driver and firmware version:

```
# wlan-version
WLAN Driver Version   : vX.X.rXX.pX
WLAN Firmware Version : IW416-V0, RF878X, FP91, 16.91.21.p11.3, WPA2_CVE_FIX_1,
PVE_FIX_1
```

Get the Wi-Fi MAC address:

```
# wlan-mac
MAC address
00:13:43:6A:5A:ED
```

Get the list of Wi-Fi networks:

```
# wlan-list
2 networks:
"xyz"
    SSID: NXPAP
    BSSID: 00:00:00:00:00:00
    channel: 6
    role: uAP
    security: WPA2

    IPv4 Address
    address: STATIC
        IP:          192.168.10.1
        gateway:    192.168.10.1
        netmask:    255.255.255.0
        dns1:       192.168.43.1
        dns2:       0.0.0.0

"abc"
    SSID: nxp
    BSSID: 00:00:00:00:00:00
    channel: (Auto)
    role: Infra
    security: WPA2

    IPv4 Address
    address: DHCP
        IP:          0.0.0.0
        gateway:    0.0.0.0
        netmask:    0.0.0.0
        dns1:       0.0.0.0
        dns2:       0.0.0.0
```

Get the Wi-Fi stats:

```
# wlan-stat
Station connected (Active)
uAP started (Active)
```

Get the AP channel:

```
# wlan-get-uap-channel
uAP channel: 6
```

Ping the IP address:

```
# ping
Incorrect usage
Usage:
    ping [-s <packet_size>] [-c <packet_count>] [-W <timeout in sec>]
<ip_address>
Default values:
    packet_size: 56
    packet_count: 10
    timeout: 2 sec
# ping -s 56 -c 2 -W 2 192.168.43.1
PING 192.168.43.1 (192.168.43.1) 56(84) bytes of data
64 bytes from 192.168.43.1: icmp_req=1 ttl=64 time=196 ms
64 bytes from 192.168.43.1: icmp_req=2 ttl=64 time=95 ms
```

Send RF Calibration host command:

```
# wlan-send-hostcmd
Hostcmd success, response ise0 80      18      0      29      0      0
0      1      0      0      0      38      2      8      0      5      1
```

### 3.3.2 Add CLIs in wifi\_cli Sample Application

APIs can be called using CLI wrappers with the appropriate arguments. The new CLI command can be added in the existing demo application by using the existing structure that defines the list of commands. Command line arguments can be passed based on the API requirement.

The following example shows how to add a new command with arguments in the CLI application.

#### Command structure modification:

File: wlan\_test.c or wlan\_basic\_cli.c

**Structure elements:** {"command-name", "help", handler}

```
{"wlan-command-name", "<argument1><argument2><argument3>...", handler_wlan_command},
```

**Command Handler:** void handler\_wlan\_command (int argc, char \*argv[])

Store the input argv list and pass it to the relative APIs to be used by the driver/firmware.

Return value of API can be used to print the Error/Success message and command output.

```
void handler_wlan_command (int argc, char *argv[])
{
    /* argv contains pointer to the arguments and argc is the number of arguments */

    return_value = wlan_command_driver_API(argument1, argument2, argument3,...);
    if(return_value == WM_SUCCESS)
    {
        /* Print success message and command output */
    }
    else
    {
        /* Print failure message and error number */
    }
}
```

### 3.4 wifi\_webconfig Sample Application

This section describes *wifi\_webconfig* sample application and its configuration along with the application execution. The *wifi\_webconfig* sample application is used to demonstrate a commissioning procedure using the uAP with an HTTP server to configure client mode to connect to an AP.

A simple LED control is implemented to check the operational mode. LED is on if the device is in AP mode and it turns off after device is set to client mode.

The website in AP mode shows the available networks using scan. The desired network can be chosen by clicking on the listed SSID. Once SSID and passphrase are entered and posted, the device attempts to connect to the chosen network with the given configuration.

The Wi-Fi credentials are stored in *mflash*, so the device can connect to the network after a reboot. Once the device comes up with the client mode, the AP mode goes down, and consequently the website closes down.

The website allows the user to reset the device to AP mode.

The following figure shows the logical flow diagram of the *wifi\_webconfig* sample application.

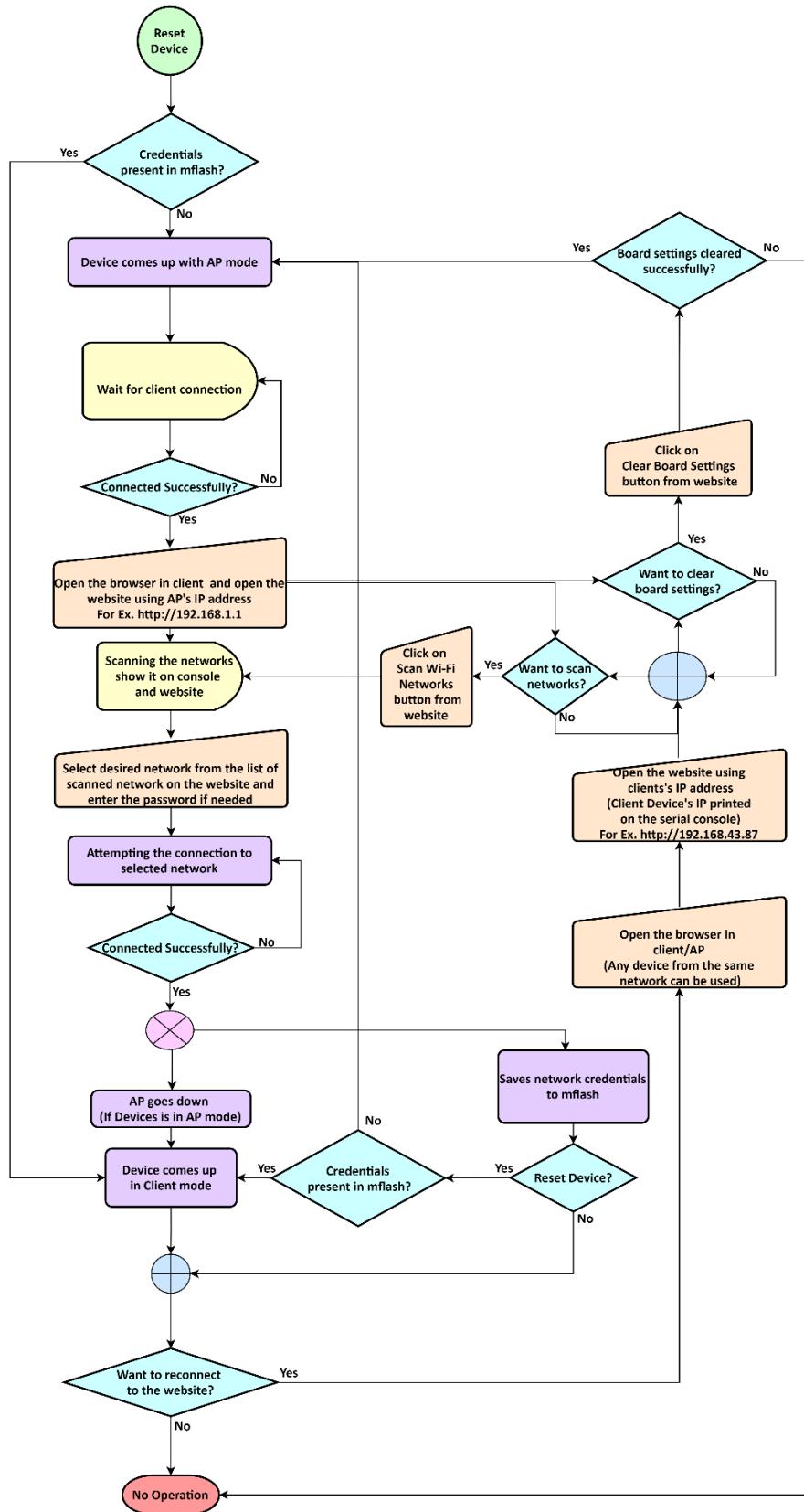


Figure 34: wifi\_webconfig flow diagram

The *wifi\_webconfig* application features are summarized in the table below.

Table 12: wifi\_webconfig Sample Application Features

Features	Details
Wi-Fi and HTTP	Wi-Fi Soft AP mode Wi-Fi Station mode Wi-Fi Security (WPA2 by default for Soft AP) Desired Channel Selection for AP HTTP server (Request GET/POST) DHCP Server/Client

### 3.4.1 User Configurations

Some of the Wi-Fi features and feature related macros that user can configure based on requirement are listed in below table along with source file name.

#### Wi-Fi configurations

Table 13: wifi\_webconfig Application Wi-Fi Configurations

Feature	Macro definition	Default value	File name	Details
Wi-Fi Soft AP	WIFI_SSID	“nxp_configuration_access_point”	webconfig.h	Default SSID and passphrase to start soft AP using the given sample application. It can be modified by changing the macro value. Default wpa2 security is used.
	WIFI_PASSWORD	“NXP0123456789”		
	WIFI_AP_CHANNEL	1		
	WIFI_AP_IP_ADDR	“192.168.1.1”		
	WIFI_AP_NET_MASK	“255.255.0.0”		

### 3.4.2 wifi\_webconfig Application Execution

Please refer to the previous sections 3.1.3, 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console setup.

#### 3.4.2.1 Start-up logs

The following logs can be observed on the console once the devices (i.MX RT1060 EVK board and NXP-based wireless module) are up and running. The *Wi-Fi FW version log* shows that the Wi-Fi module is ready to operate.

```
Starting webconfig DEMO
[i] Trying to load data from mflash.
[i] Nothing stored yet
[i] Initializing WiFi connection...
MAC Address: 00:13:43:7F:9C:9F
[net] Initialized TCP/IP networking stack
WLAN initialized
WLAN FW Version: IW416-V0, RF878X, FP91, 16.91.21.p11.3, WPA2_CVE_FIX 1,
PVE_FIX 1
[i] Successfully initialized WiFi module
Starting Access Point: SSID: nxp_configuration_access_point, Chnl: 1
[wlcm] Warn: NOTE: uAP will automatically switch to the channel that station is
on.
Soft AP started successfully
This also starts DHCP Server with IP 192.168.1.1
Now join that network on your device and connect to this IP: 192.168.1.1
```

#### 3.4.2.2 Connect the client to Soft AP

Connect the client to soft AP and observe the logs with the client mac address.

```
Client => 14:AB:C5:F4:C4:C3 Associated with Soft AP
```

#### 3.4.2.3 Open the website in the client web browser

Use the AP IP-192.168.1.1 open website <http://192.168.1.1> in the client browser. Opening the website triggers the scan in the device and the available wireless networks are listed in the console and webpage. The current Wi-Fi mode AP is highlighted on the web page. See Figure 35.

```
Initiating scan...
```

```
Galaxy M210997
    BSSID      : 8A:A3:03:B3:09:97
    RSSI       : -86dBm
    Channel    : 2
nxp
    BSSID      : 38:E6:0A:C6:1A:EC
    RSSI       : -90dBm
    Channel    : 165
```

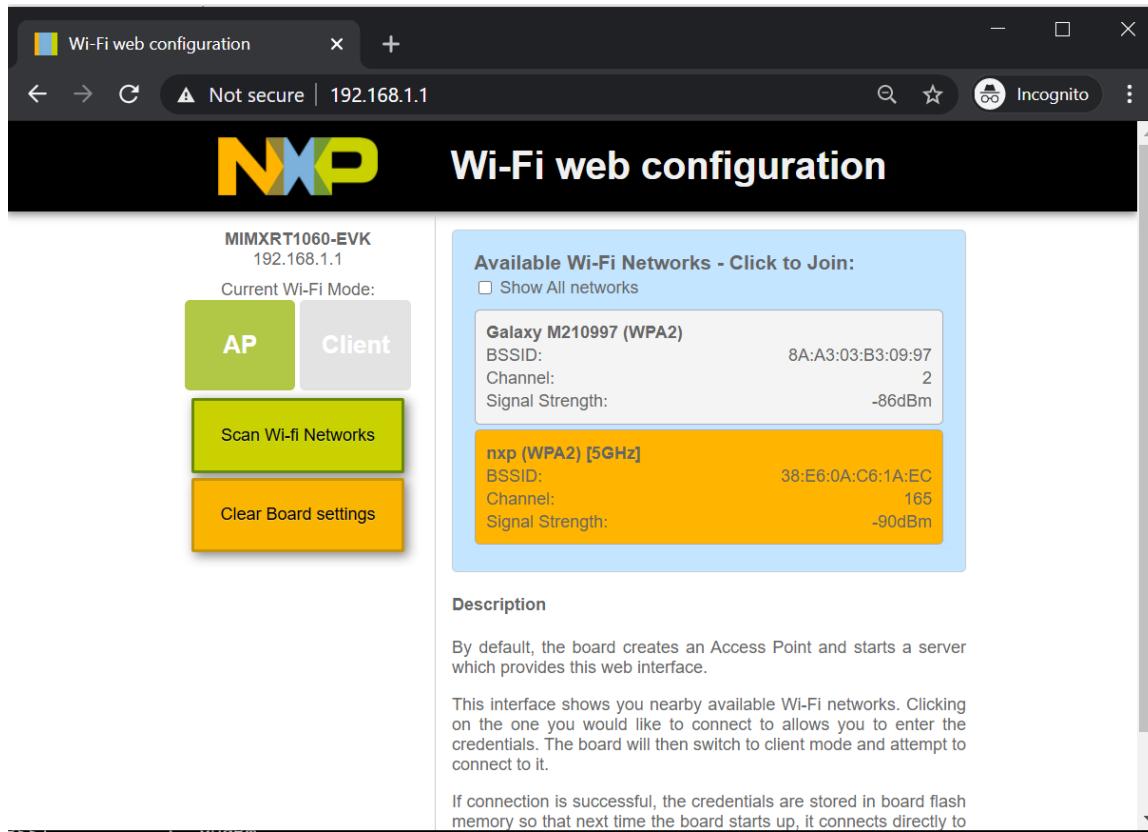


Figure 35: wifi\_webconfig Website in AP Mode

### 3.4.2.4 Connect the device to the AP

Click on the desired SSID on the web page. If the AP uses Wi-Fi security, a dialog box opens and asks to enter a password. Once the credentials are posted, the device attempts the connection to the AP.

```
[i] Chosen ssid: nxp
[i] Chosen passphrase: "12345678"
[i] Joining: nxp
Switch to channel 165 success!
Connected to following BSS:SSID = [nxp], IP = [192.168.43.35]
[i] Successfully joined: nxp
Now join that network on your device and connect to this IP: 192.168.43.35
[i] mflash_save_file success
[i] Stopping AP!
Soft AP stopped successfully
```

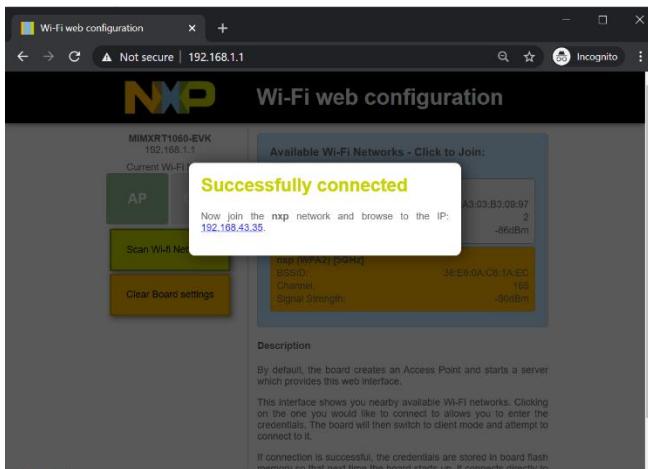
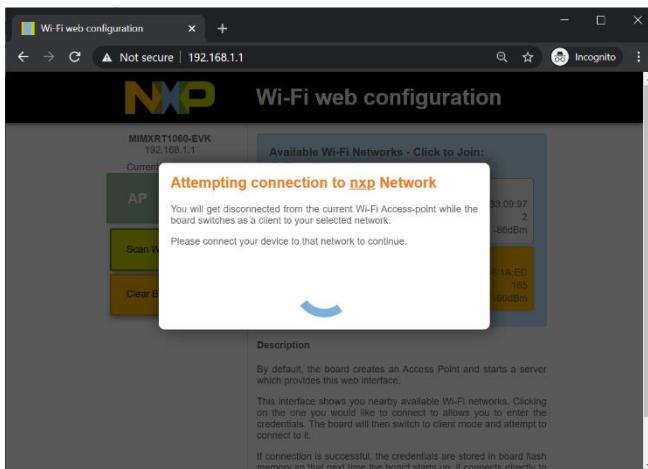
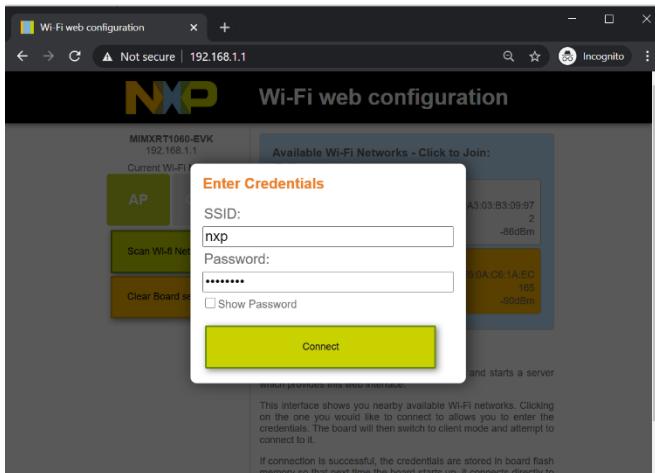


Figure 36: Connection Attempt to AP using wifi\_webconfig Application

**NOTE:** Once the configurations are successfully received by the device, soft AP goes down and the device switches to the client mode. To reconnect to the website, switch to the AP network and use the device (client mode) IP (printed on the console) to open the website.

For example, Figure 37 shows <http://192.168.43.35> to reconnect to website.

The current Wi-Fi mode client is highlighted on the webpage captured in Figure 37.

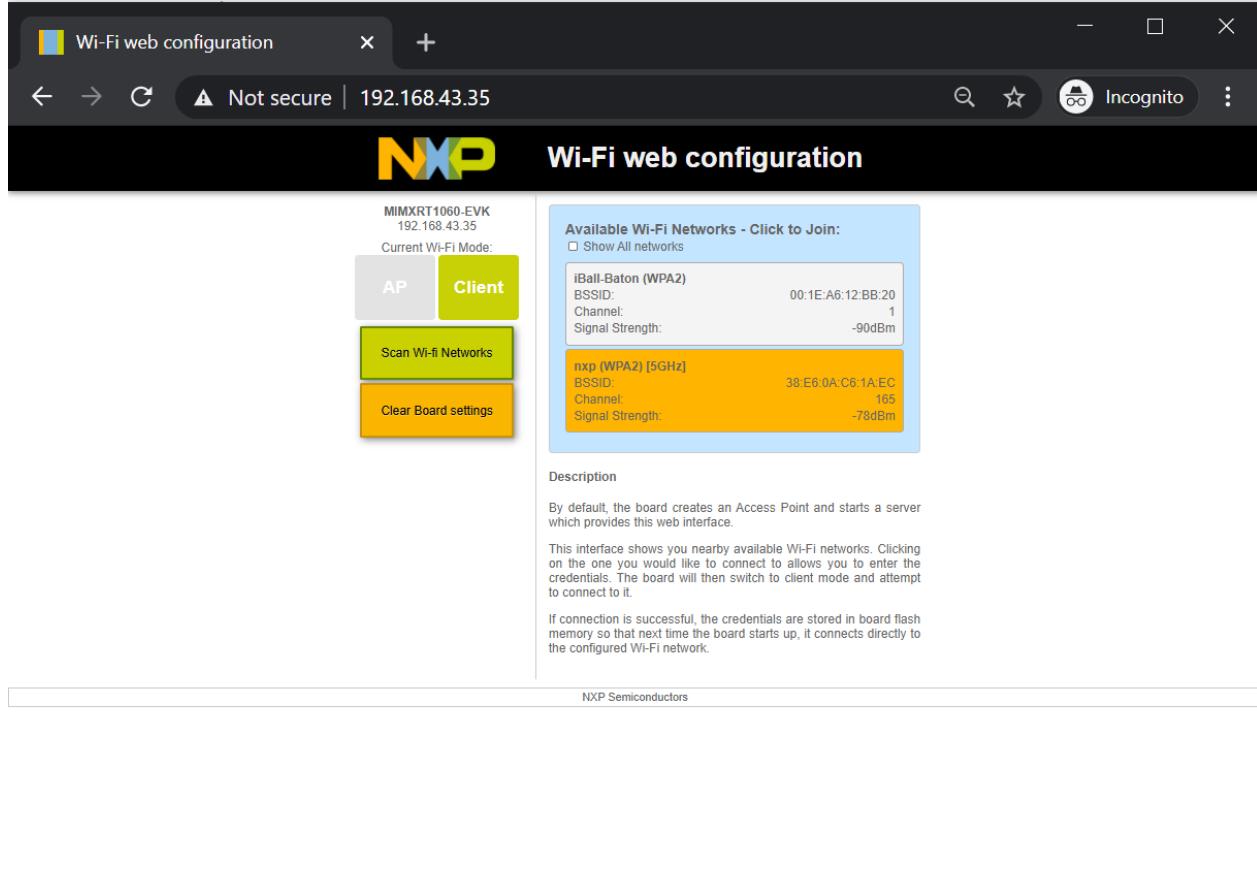


Figure 37: wifi\_webconfig Website in Client Mode

### 3.4.2.5 Device reboot with the configurations stored in mflash

The following logs can be observed when the device has the client configuration saved in *mflash*. It reads the stored information and uses it to configure client mode after a reboot.

```
Starting webconfig DEMO
[i] Trying to load data from mflash.
[i] Saved SSID: nxp, Password: 12345678
[i] Initializing WiFi connection...
MAC Address: 00:13:43:7F:9C:9F
[net] Initialized TCP/IP networking stack
WLAN initialized
WLAN FW Version: IW416-V0, RF878X, FP91, 16.91.21.p11.3, WPA2_CVE_FIX 1,
PVE_FIX 1
[i] Successfully initialized WiFi module
Connecting as client to ssid: nxp with password 12345678
    Connected to following BSS:SSID = [nxp], IP = [192.168.43.35]
[i] Connected to Wi-Fi
ssid: nxp
[!]passphrase: 12345678
Now join that network on your device and connect to this IP: 192.168.43.35
```

### 3.4.2.6 Clear the settings on the website

To clear the configurations saved in mflash, press the **Clear Board settings** button available on the webpage.

```
[i] mflash_save_file success
Dis-connected from: nxp
Starting Access Point: SSID: nxp_configuration_access_point, Chnl: 1
[wlc] Warn: NOTE: uAP will automatically switch to the channel that station is
on.
Soft AP started successfully
This also starts DHCP Server with IP 192.168.1.1
Now join that network on your device and connect to this IP: 192.168.1.1
```



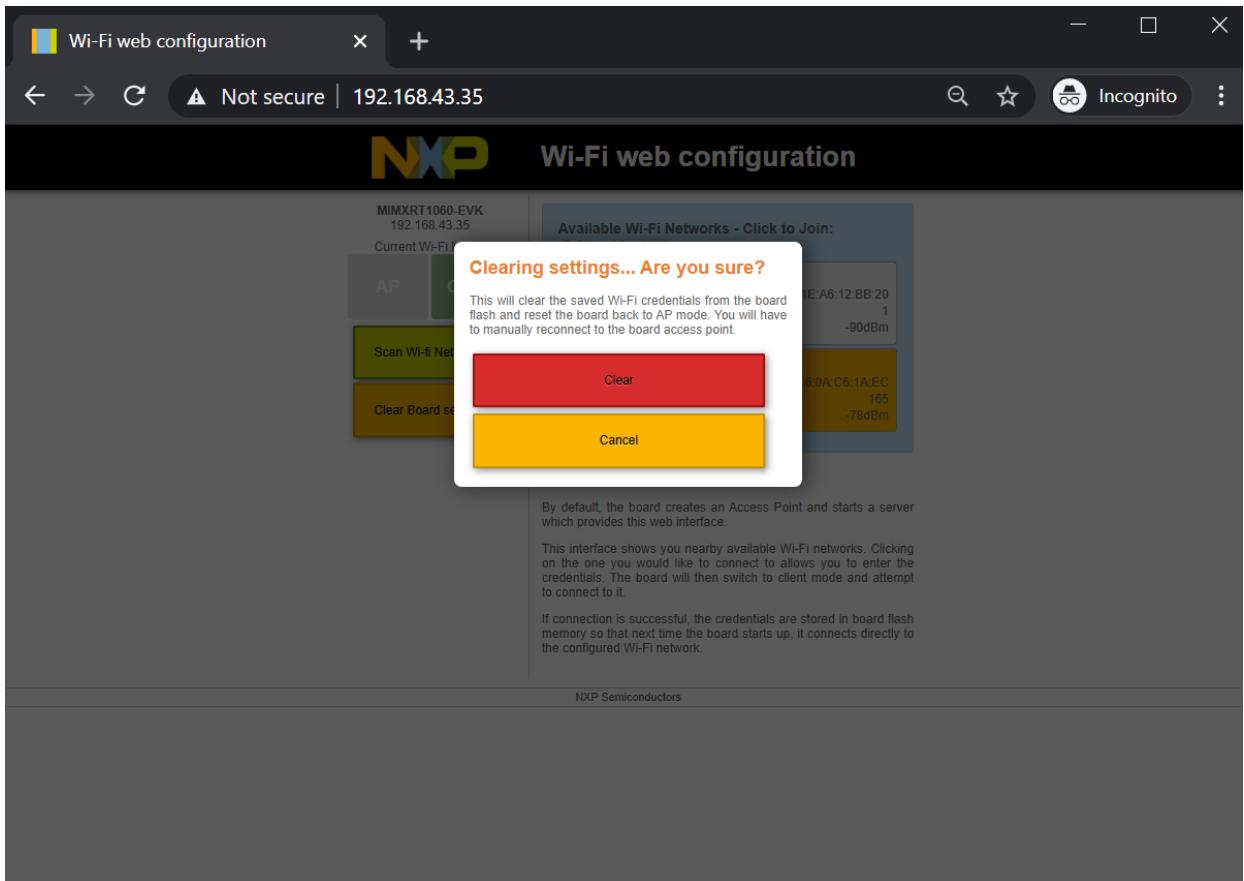


Figure 38: Clear Configurations saved in mflash using website (wifi\_webconfig Application)

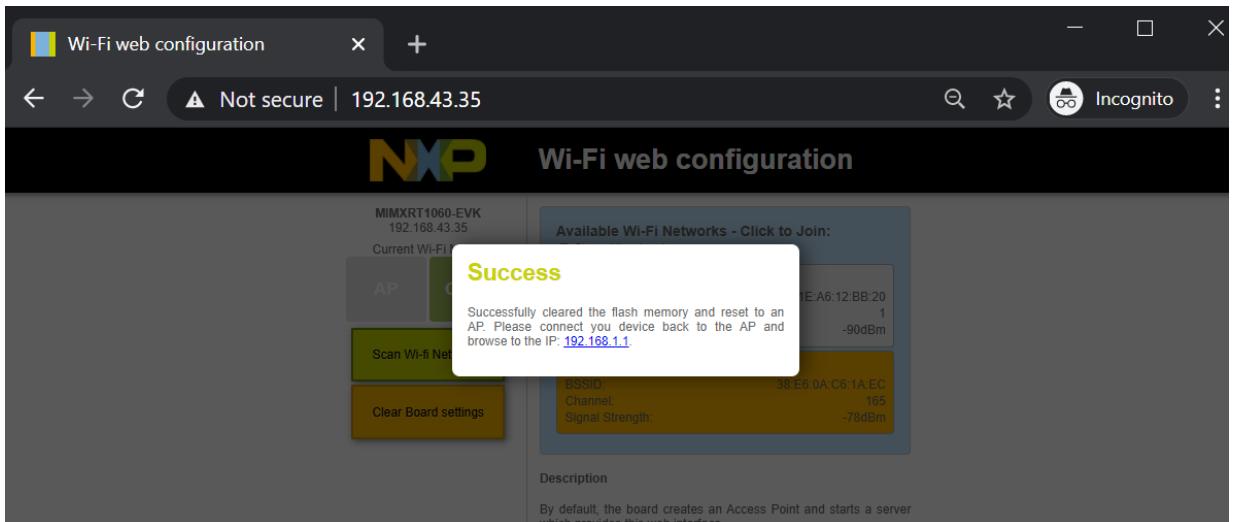


Figure 39: Clear Configuration Success Message in wifi\_webconfig Application

## 3.5 wifi\_test\_mode Sample Application

This section describes the wifi\_test\_mode application to demonstrate the CLI support to enable the user to control the Wi-Fi device to run various RF and regulatory compliance tests. This application enables RF testing for the Wi-Fi module. It helps to Measure RF parameters such as transmit power for both 2.4GHz and 5GHz, display RF packet counts, RF antenna configuration and transmit standard 802.11 packets.

### 3.5.1 wifi\_test\_mode Application Execution

Please refer to the previous sections 3.1.3, 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console setup.

#### 3.5.1.1 Run the application

This section describes the available Wi-Fi commands. The application starts with the welcome message, press **Enter** for the command prompt.

```
=====
wifi test mode demo
=====
Initialize CLI
=====
Initialize WLAN Driver
=====
MAC Address: 00:13:43:7F:9C:9F
[net] Initialized TCP/IP networking stack
=====
app_cb: WLAN: received event 10
=====
app_cb: WLAN initialized
=====
WLAN Test Mode CLIs are initialized
=====
CLIs Available:
=====

help
wlan-version
wlan-mac
wlan-set-rf-test-mode
wlan-set-rf-tx-antenna <antenna>
wlan-get-rf-tx-antenna
wlan-set-rf-rx-antenna <antenna>
wlan-get-rf-rx-antenna
wlan-set-rf-band <band>
wlan-get-rf-band
wlan-set-rf-bandwidth <bandwidth>
wlan-get-rf-bandwidth
wlan-set-rf-channel <channel>
wlan-get-rf-channel
wlan-set-rf-tx-power <tx_power> <modulation> <path_id>
wlan-set-rf-tx-cont-mode <enable_tx> <cw_mode> <payload_pattern> <cs_mode>
<act_sub_ch> <tx_rate>
wlan-set-rf-tx-frame <start> <data_rate> <frame_pattern> <frame_len>
<adjust_burst_sifs> <burst_sifs_in_us> <short_preamble> <act_sub_ch> <short_gi>
<adv_coding> <tx_bf> <gf_mode> <stbc> <bssid>
wlan-get-and-reset-rf-per
=====
```

### 3.5.1.2 Prerequisite Commands

The following steps describe prerequisite commands to start Wi-Fi RF Test.

#### Wi-Fi RF test mode enable

The following command is used to set Wi-Fi mode to rf test mode:

```
# wlan-set-rf-test-mode  
RF Test Mode configuration successful
```

#### Wi-Fi RF band set and get

The following commands are used to set and get Wi-Fi band:

##### Command Usage:

```
# wlan-set-rf-band  
Usage:  
wlan-set-rf-band <band>  
band: 0=2.4G, 1=5G
```

##### Set and Get RF band:

```
# wlan-set-rf-band 1  
RF Band configuration successful
```

```
# wlan-get-rf-band  
Configured RF Band is: 5G
```

#### Wi-Fi RF channel set and get

The following commands are used to set and get Wi-Fi channel:

##### Command Usage:

```
# wlan-set-rf-channel  
Usage:  
wlan-set-rf-channel <channel>
```

##### Set and Get RF channel:

```
# wlan-set-rf-channel 132  
Channel configuration successful
```

```
# wlan-get-rf-channel  
Configured channel is: 132
```

#### Wi-Fi RF bandwidth set and get

The following commands are used to set and get Wi-Fi bandwidth:

##### Command Usage:

```
# wlan-set-rf-bandwidth  
Usage:  
wlan-set-bandwidth <bandwidth>  
  
<bandwidth>:  
    0: 20MHz  
    1: 40MHz
```

##### Set and Ge RF Bandwidth:

```
# wlan-set-rf-bandwidth 0  
Bandwidth configuration successful
```

```
# wlan-get-rf-bandwidth  
Configured bandwidth is: 20MHz
```

### 3.5.1.3 Display and Clear Received Wi-Fi Packet Count

The following command clear the received packet count and displays the received multi-cast and error packet counts.

```
# wlan-get-and-reset-rf-per  
PER is as below:  
    Total Rx Packet Count : 20  
    Total Rx Multicast/Broadcast Packet Count: 20  
    Total Rx Packets with FCS error : 9
```

### 3.5.1.4 Wi-Fi Antenna Configuration

The following commands are used to set and get Wi-Fi Tx/Rx antenna configuration.

**Command Usage:**

```
# wlan-set-rf-tx-antenna  
Usage:  
wlan-set-rf-tx-antenna <antenna>  
antenna: 1=Main, 2=Aux
```

**Set and Get TX antenna configuration:**

```
# wlan-set-rf-tx-antenna 1  
Tx Antenna configuration successful
```

```
# wlan-get-rf-tx-antenna  
Configured Tx Antenna is: Main
```

**Command Usage:**

```
# wlan-set-rf-rx-antenna  
Usage:  
wlan-set-rf-rx-antenna <antenna>  
antenna: 1=Main, 2=Aux
```

**Set and Get RX antenna configuration:**

```
# wlan-set-rf-rx-antenna 2  
Rx Antenna configuration successful
```

```
# wlan-get-rf-rx-antenna  
Configured Rx Antenna is: Aux
```

### 3.5.1.5 Wi-Fi Tx Power configuration

The following command is used to set the transmitter output power at the antenna using stored calibration data. Power level is in dBm.

**Command Usage:**

```
# wlan-set-rf-tx-power  
Usage:  
wlan-set-rf-tx-power <tx_power> <modulation> <path_id>  
Power      (0 to 24 dBm)  
Modulation (0: CCK, 1:OFDM, 2:MCS)  
Path ID   (0: PathA, 1:PathB, 2:PathA+B)
```

**Set Tx Power:**

```
# wlan-set-rf-tx-power 8 1 1  
Tx Power configuration successful  
    Power      : 8 dBm  
    Modulation : OFDM  
    Path ID   : PathB
```

### 3.5.1.6 Wi-Fi set transmitter in CW mode

The following command is used to set Wi-Fi transmitter to Continuous Wave (CW) mode.

Command Usage:

*For different data rate values See Table 14: Data rate parameter*

```
# wlan-set-rf-tx-cont-mode
Usage:
wlan-set-rf-tx-cont-mode <enable_tx> <cw_mode> <payload_pattern> <cs_mode>
<act_sub_ch> <tx_rate>
Enable           (0:disable, 1:enable)
Continuous Wave Mode (0:disable, 1:enable)
Payload Pattern      (0 to 0xFFFFFFFF) (Enter hexadecimal value)
CS Mode            (Applicable only when continuous wave is disabled)
(0:disable, 1:enable)
Active SubChannel   (0:low, 1:upper, 3:both)
Tx Data Rate        (Rate Index corresponding to legacy/HT/VHT rates)

To Disable:
wlan-set-rf-tx-cont-mode 0
```

Enable CW mode:

```
# wlan-set-rf-tx-cont-mode 1 1 B496DEB6 0 1 7
Tx continuous configuration successful
    Enable          : enable
    Continuous Wave Mode : enable
    Payload Pattern : 0x7FFFFFFF
    CS Mode         : disable
    Active SubChannel : upper
    Tx Data Rate    : 7
```

Disable CW mode:

```
# wlan-set-rf-tx-cont-mode 0
Tx continuous configuration successful
    Enable          : disable
    Continuous Wave Mode : disable
    Payload Pattern : 0x00000000
    CS Mode         : disable
    Active SubChannel : low
    Tx Data Rate    : 0
```

**NOTE:** It is require to disable CW mode once test completed. CW mode test and TX frame test not supports parallel operation.

Table 14: Data rate parameter

ID	Data rate
0	1Mbits/sec
1	2Mbits/sec
2	5.5Mbits/sec
3	11Mbits/sec
4	22Mbits/sec
5	6Mbits/sec
6	9Mbits/sec
7	12Mbits/sec
8	18Mbits/sec
9	24Mbits/sec
10	36Mbits/sec
11	48Mbits/sec

12	54Mbits/sec
13	72Mbits/sec
14	HT_MCS 0
15	HT_MCS 1
16	HT_MCS 2
17	HT_MCS 3
18	HT_MCS 4
19	HT_MCS 5
20	HT_MCS 6
21	HT_MCS 7
46	HT_MCS 32

### 3.5.1.7 Transmit standard 802.11 packets

The following command is used to continuously transmit packets, with an adjustable time gap of 0 to 250 microseconds between packets.

Command Usage:

*For different data rate values See Table 14: Data rate parameter*

```
# wlan-set-rf-tx-frame
Usage:
wlan-set-rf-tx-frame <start> <data_rate> <frame_pattern> <frame_len>
<adjust_burst_sifs> <burst_sifs_in_us> <short_preamble> <act_sub_ch> <short_gi>
<adv_coding> <tx_bf> <gf_mode> <stbc> <bssid>
Enable          (0:disable, 1:enable)
Tx Data Rate    (Rate Index corresponding to legacy/HT/VHT rates)
Payload Pattern (0 to 0xFFFFFFFF) (Enter hexadecimal value)
Payload Length  (1 to 0x400) (Enter hexadecimal value)
Adjust Burst SIFS3 Gap (0:disable, 1:enable)
Burst SIFS in us (0 to 255us)
Short Preamble   (0:disable, 1:enable)
Active SubChannel (0:low, 1:upper, 3:both)
Short GI          (0:disable, 1:enable)
Adv Coding        (0:disable, 1:enable)
Beamforming       (0:disable, 1:enable)
GreenField Mode   (0:disable, 1:enable)
STBC              (0:disable, 1:enable)
BSSID             (xx:xx:xx:xx:xx:xx)

To Disable:
wlan-set-rf-tx-frame 0
```

Enable Tx Frame:

```
# wlan-set-rf-tx-frame 1 7 2730 256 0 0 0 0 0 0 0 0 0 0 38:E6:0A:C6:1A:EC
Tx Frame configuration successful
  Enable          : enable
  Tx Data Rate    : 7
  Payload Pattern : 0x00002730
  Payload Length  : 0x00000256
  Adjust Burst SIFS3 Gap : disable
  Burst SIFS in us : 0 us
  Short Preamble   : disable
  Active SubChannel : low
  Short GI          : disable
  Adv Coding        : disable
  Beamforming       : disable
  GreenField Mode   : disable
```

STBC	:	disable
BSSID	:	38:E6:0A:C6:1A:EC

#### Packet Capture:

Please refer section 2.2 for the Wireshark tool setup and start capturing packets for configured channel and bandwidth.

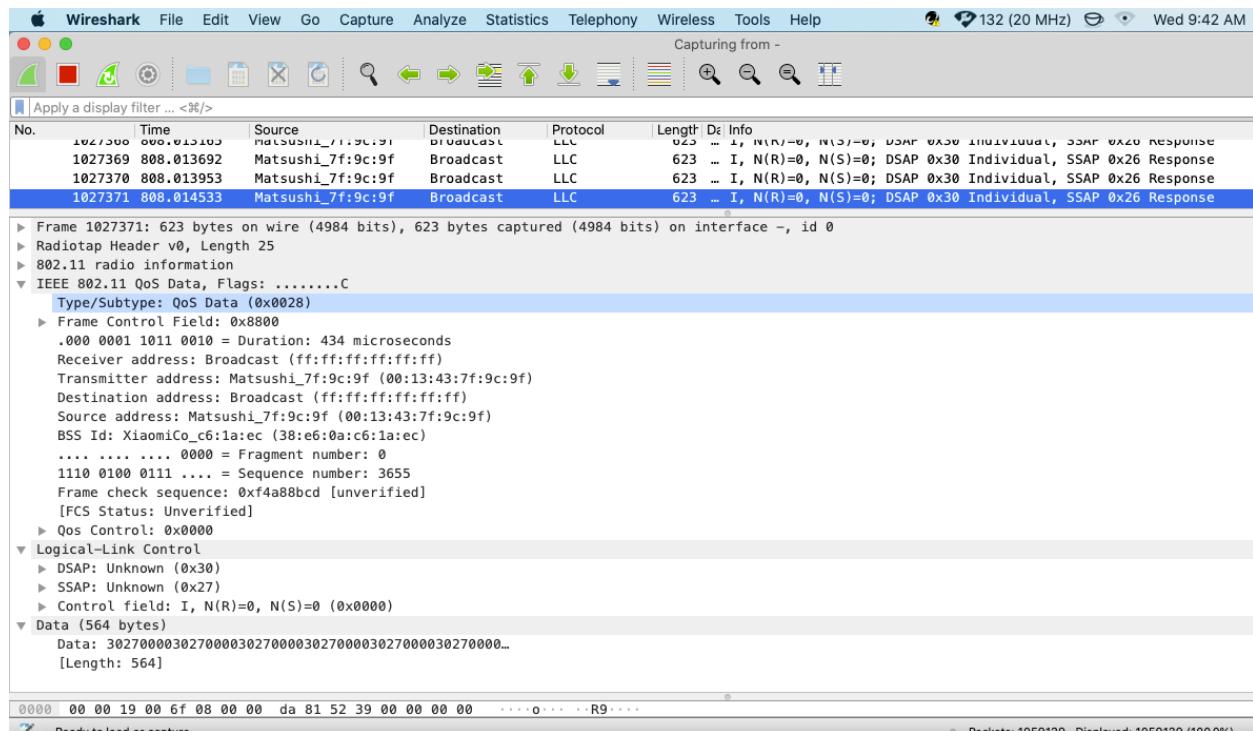


Figure 40: TX Frame Packet Capture

#### Disable TX Frame:

```
# wlan-set-rf-tx-frame 0
Tx Frame configuration successful
  Enable          : disable
  Tx Data Rate    : 0
  Payload Pattern : 0x00000000
  Payload Length  : 0x00000001
  Adjust Burst SIFS3 Gap : disable
  Burst SIFS in us : 0 us
  Short Preamble   : disable
  Active SubChannel : low
  Short GI         : disable
  Adv Coding        : disable
  Beamforming       : disable
  GreenField Mode   : disable
  STBC              : disable
  BSSID             : 00:00:00:00:00:00
```

#### 3.5.1.8 Other useful CLI commands

Use the other commands to get the Wi-Fi information, driver version and firmware version.

##### Get the Wi-Fi driver and firmware version:

```
# wlan-version
WLAN Driver Version    : vX.X.rXX.pX
WLAN Firmware Version : IW416-V0, RF878X, FP91, 16.91.21.p11.3, WPA2_CVE_FIX 1,
PVE_FIX 1
```

##### Get the Wi-Fi MAC address:

```
MAC address
00:13:43:7F:9C:9F
```

### 3.5.1.9 Example command sequences for adjusting Tx power in 2.4GHz

The radio is configured as shown below.

- 2.4 GHz band
- Channel 6
- 20 MHz bandwidth
- 6 Mbps legacy data rate
- Test pattern transmitted is 0x00000AAA
- Output power set to +15 dBm, then adjusted to +14 dBm
- For different data rate values See Table 14: Data rate parameter

Table 15: Tx power command sequences for 2.4GHz

Step	Operation	Command
1	Set RF test mode	# wlan-set-rf-test-mode rf_test_mode set successfully
2	Set RF band	# wlan-set-rf-band 0 RF Band configuration successful
3	Set RF bandwidth (switched order with step 4)	# wlan-set-rf-bandwidth 0 Bandwidth configuration successful
4	Set RF channel	# wlan-set-rf-channel 6 Channel configuration successful
5	Set Tx antenna	# wlan-set-rf-tx-antenna 1 Tx antenna configuration successful
6	Get settings (optional)	# wlan-get-rf-band Configured RF band is: 2.4 G # wlan-get-rf-channel Configured channel is: 6 # wlan-get-rf-bandwidth Configured bandwidth is: 20MHz
7	Set output power to +15 dBm	# wlan-set-rf-tx-power 15 1 0 Tx Power configuration successful Power : 15 dBm Modulation : OFDM Path ID : PathA
8	Set continuous transmit mode	# wlan-set-rf-tx-cont-mode 1 0 0xAAA 0 3 5 Tx continuous mode successful Enable : enable CW mode : disable Payload pattern : 0x00000AAA CS mode : disable Active SubChannel : both Tx Data Rate : 5

9	Stop transmission	<b># wlan-set-rf-tx-cont-mode 0</b>
10	Set output power to +14 dBm	<b># wlan-set-rf-tx-power 14 1 0</b> Tx Power configuration successful Power : 14 dBm Modulation : OFDM Path ID : PathA
11	Restart transmission	<b># wlan-set-rf-tx-cont-mode 1 0 0xAAA 0 3 5</b> Tx continuous mode successful Enable : enable CW mode : disable Payload pattern : 0x00000AAA CS mode : disable Active SubChannel : both Tx Data Rate : 5
12	Stop transmission	<b># wlan-set-rf-tx-cont-mode 0</b>

### 3.5.1.10 Example command sequences for adjusting Tx power in 5GHz

The radio is configured as shown below.

- 5 GHz band
- Channel 44/48
- 40 MHz bandwidth
- MCS0 HT data rate
- Test pattern transmitted is 0x00BBBAAA
- Output power set to +9 dBm, then adjusted to +8 dBm.
- For different data rate values See Table 14: Data rate parameter

Table 16: Tx power command sequences for 5GHz

Step	Operation	Command
1	Set RF test mode	<b># wlan-set-rf-test-mode</b> RF Test Mode configuration successful
2	Set RF band	<b># wlan-set-rf-band 1</b> RF Band configuration successful
3	Set RF bandwidth (switched order with step 4)	<b># wlan-set-rf-bandwidth 1</b> Bandwidth configuration successful
4	Set RF channel	<b># wlan-set-rf-channel 48</b> Channel configuration successful
5	Set Tx antenna	<b># wlan-set-rf-tx-antenna 1</b> Tx antenna configuration successful
6	Get settings (optional)	<b># wlan-get-rf-band</b> Configured RF band is: 5 G <b># wlan-get-rf-channel</b> Configured channel is: 48 <b># wlan-get-rf-bandwidth</b> Configured bandwidth is: 40MHz
7	Set output power to +10 dBm	<b># wlan-set-rf-tx-power 10 1 0</b>

		Tx Power configuration successful Power : 10 dBm Modulation : OFDM Path ID : PathA
8	Set continuous transmit mode	# wlan-set-rf-tx-cont-mode 1 0 <b>0xBBBAAA 0 3 14</b> Tx continous mode successful Enable : enable CW mode : disable Payload pattern : 0x00BBBAAA CS mode : disable Active SubChannel : both Tx Data Rate : 14
9	Stop transmission	# wlan-set-rf-tx-cont-mode 0
10	Set output power to +9 dBm	# wlan-set-rf-tx-power 9 1 0 Tx Power configuration successful Power : 9 dBm Modulation : OFDM Path ID : PathA
11	Restart transmission	# wlan-set-rf-tx-cont-mode 1 0 <b>0xBBBAAA 0 3 14</b> Tx continous mode successful Enable : enable CW mode : disable Payload pattern : 0x00BBBAAA CS mode : disable Active SubChannel : both Tx Data Rate : 14
12	Stop transmission	# wlan-set-rf-tx-cont-mode 0
13	Set output power to +8 dBm	# wlan-set-rf-tx-power 8 1 0 Tx Power configuration successful Power : 8 dBm Modulation : OFDM Path ID : PathA
14	Restart transmission	# wlan-set-rf-tx-cont-mode 1 0 <b>0xBBBAAA 0 3 14</b> Tx continous mode successful Enable : enable CW mode : disable Payload pattern : 0x00BBBAAA CS mode : disable Active SubChannel : both

		Tx Data Rate : 14
<b>15</b>	Stop transmission	<b># wlan-set-rf-tx-cont-mode 0</b>

## 3.6 wifi\_cert Sample Application

This section describes the *wifi\_cert* application to demonstrate the CLI support to handle and enable Wi-Fi configuration for different features. This sample application includes commands related to the Wi-Fi certification process. In this sample application Wi-Fi connection manager CLIs are available.

**NOTE:** Support for this application is available only for i.MX RT1060 EVK board.

Table 17: wifi\_cert Application Features

Features	Details
Wi-Fi	Wi-Fi Soft AP mode Wi-Fi Station mode Wi-Fi Scan Wi-Fi Tx Power Limit Wi-Fi Active/Passive Channel List Wi-Fi Tx Data Rate Wi-Fi Management Frame Protection Wi-Fi Antenna Diversity Wi-Fi ED MAC
IPerf	TCP Client and Server TCP Client dual mode (Tx and Rx in simultaneous) TCP Client trade-off mode (Tx and Rx individual)  UDP Client and Server UDP Client dual mode (Tx and Rx in simultaneous) UDP Client trade-off mode (Tx and Rx individual)

### 3.6.1 wifi\_cert Application Execution

Please refer to the previous sections 3.1.3, 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console setup.

### 3.6.1.1 Run the application

This section describes the available Wi-Fi commands. The application starts with the welcome message, press **Enter** for the command prompt.

```
=====
wifi cert demo
=====
Initialize CLI
=====
Initialize WLAN Driver
=====
MAC Address: 00:13:43:7F:9C:9F
[net] Initialized TCP/IP networking stack
=====
app_cb: WLAN: received event 10
=====
app_cb: WLAN initialized
=====
WLAN CLIs are initialized
=====
ENHANCED WLAN CLIs are initialized
=====
CLIs Available:
=====
help
wlan-version
wlan-mac
wlan-scan
wlan-scan-opt ssid <ssid> bssid ...
wlan-add <profile_name> ssid <ssid> bssid...
wlan-remove <profile_name>
wlan-list
wlan-connect <profile_name>
wlan-start-network <profile_name>
wlan-stop-network
wlan-disconnect
wlan-stat
wlan-info
wlan-address
wlan-get-uap-channel
wlan-get-uap-sta-list
wlan-ieee-ps <0/1>
wlan-deep-sleep-ps <0/1>
wlan-set-regioncode <region-code>
wlan-get-regioncode
wlan-get-txpwrlimit <subband>
wlan-set-txpwrlimit
wlan-set-chanlist-and-txpwrlimit
wlan-set-chanlist
wlan-get-chanlist
wlan-set-txratecfg <format> <index>
wlan-get-txratecfg
wlan-get-data-rate
wlan-set-pmfcfg <mfpc> <mfpr>
wlan-get-pmfcfg
wlan-set-antcfg <ant mode> [evaluate_time]
wlan-get-antcfg
wlan-set-ed-mac-mode <ed_ctrl_2g> <ed_offset_2g> <ed_ctrl_5g> <ed_offset_5g>
wlan-get-ed-mac-mode
ping [-s <packet_size>] [-c <packet_count>] [-W <timeout in sec>] <ip_address>
iperf [-s|-c <host>|-a|-h] [options]
dhcp-stat
```

=====

**Note:** Please refer sections 3.3.1.1 to 3.3.1.4 for basic Wi-Fi features like Wi-Fi Scan, Wi-Fi AP mode, Wi-Fi Station mode, IPerf etc.

### 3.6.1.2 Set/Get Region Code

The following commands are used to set and get region code:

**Command Usage:**

```
# wlan-set-regioncode
```

Usage:

```
wlan-set-regioncode <region-code>
```

where, region code =

- 0xAA : World Wide Safe Mode
- 0x10 : US FCC, Singapore
- 0x20 : IC Canada
- 0x30 : ETSI, Australia, Republic of Korea
- 0x32 : France
- 0x40 : Japan
- 0x41 : Japan
- 0x50 : China
- 0xFE : Japan
- 0xFF : Special

**Set region code:**

```
# wlan-set-regioncode 0xAA
```

```
Region code: 0xaa set
```

**Get region code:**

```
# wlan-get-regioncode
```

```
Region code: 0xaa
```

**NOTE:** We cannot set region code if the region code is programmed in the module's One Time Programmable (OTP) memory during production process of device.

### 3.6.1.3 Set/Get Tx Power Limit

The following commands are used to get and set tx power limit:

Command Usage:

```
# wlan-get-txpwrlimit
```

Usage:

```
wlan-get-txpwrlimit <subband>
```

Where subband is:

```
0x00 2G subband (2.4G: channel 1-14)
0x10 5G subband0 (5G: channel 36,40,44,48,
52,56,60,64)
0x11 5G subband1 (5G: channel 100,104,108,112,
116,120,124,128,
132,136,140,144)
0x12 5G subband2 (5G: channel 149,153,157,161,165,172)
0x13 5G subband3 (5G: channel 183,184,185,187,188,
189, 192,196;
5G: channel 7,8,11,12,16,34)
```

Get Tx Power Limit:

```
# wlan-get-txpwrlimit 0x10
-----
-
Get txpwrlimit: sub_band=10
StartFreq: 5000
ChanWidth: 20
ChanNum: 36
Pwr:1,16,2,16,3,14,4,16,5,16,6,14,7,16,8,16,9,14,10,15,11,14,12,15,13,15,14,14,
15,13
StartFreq: 5000
ChanWidth: 20
ChanNum: 40
Pwr:1,16,2,16,3,14,4,16,5,16,6,14,7,16,8,16,9,14,10,15,11,14,12,15,13,15,14,14,
15,13
StartFreq: 5000
ChanWidth: 20
ChanNum: 44
Pwr:1,16,2,16,3,14,4,16,5,16,6,14,7,16,8,16,9,14,10,15,11,14,12,15,13,15,14,14,
15,13
StartFreq: 5000
ChanWidth: 20
ChanNum: 48
Pwr:1,16,2,16,3,14,4,16,5,16,6,14,7,16,8,16,9,14,10,15,11,14,12,15,13,15,14,14,
15,13
StartFreq: 5000
ChanWidth: 20
ChanNum: 52
Pwr:1,17,2,16,3,14,4,17,5,16,6,14,7,17,8,16,9,14,10,15,11,14,12,15,13,15,14,14,
15,13
StartFreq: 5000
ChanWidth: 20
ChanNum: 56
Pwr:1,17,2,16,3,14,4,17,5,16,6,14,7,17,8,16,9,14,10,15,11,14,12,15,13,15,14,14,
15,13
StartFreq: 5000
ChanWidth: 20
ChanNum: 60
```

```
Pwr:1,17,2,16,3,14,4,17,5,16,6,14,7,17,8,16,9,14,10,15,11,14,12,15,13,15,14,14,  
15,13  
StartFreq: 5000  
ChanWidth: 20  
ChanNum: 64  
Pwr:1,17,2,16,3,14,4,17,5,16,6,14,7,17,8,16,9,14,10,15,11,14,12,15,13,15,14,14,  
15,13
```

## Set Tx Power Limit:

**NOTE:** This command will first set tx power configurations for the band 2GHz or both 2GHz and 5GHz based on the Wi-Fi module selection and then prints saved tx power configurations of all the sub-bands on the console output.

```
# wlan-set-txpwrlimit
-----
-
StartFreq: 2407
ChanWidth: 20
ChanNum: 1
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 2
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 3
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 4
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 5
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 6
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 7
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 8
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 9
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 10
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 11
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14,7,18,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 12
Pwr:0,16,1,16,2,16,3,14,4,16,5,16,6,14,7,16,8,16,9,14,10,16,11,16
StartFreq: 2407
ChanWidth: 20
ChanNum: 13
Pwr:0,16,1,16,2,16,3,14,4,16,5,16,6,14,7,16,8,16,9,14,10,16,11,16
StartFreq: 2414
ChanWidth: 20
ChanNum: 14
```

```
Pwr:0,12,1,12,2,12,3,12,4,12,5,12,6,12,7,12,8,12,9,12,10,12,11,12
...
```

### 3.6.1.4 Set/Get Active/Passive Channel List

The following commands are used to set and get active and passive channel list.

**Set Channel List:**

```
# wlan-set-chanlist
```

```
-
```

```
Number of channels configured: 14
```

```
ChanNum: 1      ChanFreq: 2412  Active
ChanNum: 2      ChanFreq: 2417  Active
ChanNum: 3      ChanFreq: 2422  Active
ChanNum: 4      ChanFreq: 2427  Active
ChanNum: 5      ChanFreq: 2432  Active
ChanNum: 6      ChanFreq: 2437  Active
ChanNum: 12     ChanFreq: 2467  Passive
ChanNum: 36     ChanFreq: 5180  Active
ChanNum: 40     ChanFreq: 5200  Active
ChanNum: 44     ChanFreq: 5220  Active
ChanNum: 48     ChanFreq: 5240  Active
ChanNum: 52     ChanFreq: 5260  Passive
ChanNum: 56     ChanFreq: 5280  Passive
ChanNum: 100    ChanFreq: 5500  Passive
```

**Get Channel List:**

```
# wlan-get-chanlist
```

```
-
```

```
Number of channels configured: 14
```

```
ChanNum: 1      ChanFreq: 2412  Active
ChanNum: 2      ChanFreq: 2417  Active
ChanNum: 3      ChanFreq: 2422  Active
ChanNum: 4      ChanFreq: 2427  Active
ChanNum: 5      ChanFreq: 2432  Active
ChanNum: 6      ChanFreq: 2437  Active
ChanNum: 12     ChanFreq: 2467  Passive
ChanNum: 36     ChanFreq: 5180  Active
ChanNum: 40     ChanFreq: 5200  Active
ChanNum: 44     ChanFreq: 5220  Active
ChanNum: 48     ChanFreq: 5240  Active
ChanNum: 52     ChanFreq: 5260  Passive
ChanNum: 56     ChanFreq: 5280  Passive
ChanNum: 100    ChanFreq: 5500  Passive
```

### 3.6.1.5 Set Channel List and Tx Power Limit

The following command is used to set channel list as well as tx power limit.

**NOTE:** This command will first set defined configuration of channel list and tx power limit and then prints saved configuration on the console output.

```
# wlan-set-chanlist-and-txpwrlimit
```

```
-
```

```
Get txpwrlimit: sub_band=0
StartFreq: 2407
ChanWidth: 20
ChanNum: 1
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14
StartFreq: 2407
```

```
ChanWidth: 20
ChanNum: 2
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14
StartFreq: 2407
ChanWidth: 20
ChanNum: 3
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14
StartFreq: 2407
ChanWidth: 20
ChanNum: 4
Pwr:0,18,1,18,2,16,3,14,4,18,5,16,6,14
StartFreq: 2407
ChanWidth: 20
ChanNum: 5
...
Number of channels configured: 14

ChanNum: 1      ChanFreq: 2412  Active
ChanNum: 2      ChanFreq: 2417  Active
ChanNum: 3      ChanFreq: 2422  Active
ChanNum: 4      ChanFreq: 2427  Active
ChanNum: 5      ChanFreq: 2432  Active
ChanNum: 6      ChanFreq: 2437  Active
ChanNum: 12     ChanFreq: 2467  Passive
ChanNum: 36     ChanFreq: 5180  Active
ChanNum: 40     ChanFreq: 5200  Active
ChanNum: 44     ChanFreq: 5220  Active
ChanNum: 48     ChanFreq: 5240  Active
ChanNum: 52     ChanFreq: 5260  Passive
ChanNum: 56     ChanFreq: 5280  Passive
ChanNum: 100    ChanFreq: 5500  Passive
```

### 3.6.1.6 Set/Get Tx Rate Configuration

The following commands are used to set and get tx rate.

**Command Usage:**

```
# wlan-set-txratecfg
Invalid arguments
Usage:
wlan-set-txratecfg <format> <index>
    Where
        <format> - This parameter specifies the data rate format used in this
command
            0:      LG
            1:      HT
            0xff: Auto
        <index> - This parameter specifies the rate or MCS index
        If <format> is 0 (LG),
            0      1 Mbps
            1      2 Mbps
            2      5.5 Mbps
            3      11 Mbps
            4      6 Mbps
            5      9 Mbps
            6      12 Mbps
            7      18 Mbps
            8      24 Mbps
            9      36 Mbps
            10     48 Mbps
            11     54 Mbps
        If <format> is 1 (HT),
            0      MCS0
            1      MCS1
            2      MCS2
            3      MCS3
            4      MCS4
            5      MCS5
            6      MCS6
            7      MCS7
```

**Set Tx Rate:**

```
# wlan-set-txratecfg ff 0
Configured txratecfg as below:
Tx Rate Configuration:
    Type:          0 (LG)
    Rate Index: 0 (1 Mbps)
```

**Get Tx Rate:**

```
# wlan-get-txratecfg
Tx Rate Configuration:
    Type:          0xFF (Auto)
    Rate Index: 0 (1 Mbps)
```

**Get Data Rate:**

```
# wlan-get-data-rate
Data Rate:
    TX:
        Type: LG
        Rate: 1 Mbps
    RX:
        Type: LG
        Rate: 1 Mbps
```

### 3.6.1.7 Set/Get Management Frame Protection Capability

The following commands are used to set and get MFP capability:

Command Usage:

```
# wlan-set-pmfcfg
```

Usage:

```
wlan-set-pmfcfg <mfpc> <mfpr>
```

<mfpc>: Management Frame Protection Capable (MFPC)

    1: Management Frame Protection Capable

    0: Management Frame Protection not Capable

<mfpr>: Management Frame Protection Required (MFPR)

    1: Management Frame Protection Required

    0: Management Frame Protection Optional

Default setting is PMF not capable.

mfpc = 0, mfpr = 1 is an invalid combination

Set MFP capability:

```
# wlan-set-pmfcfg 1 1
```

PMF configuration successful

Get MFP Capability:

```
# wlan-get-pmfcfg
```

Management Frame Protection Capability: Yes

Management Frame Protection: Required

### 3.6.1.8 Set/Get Antenna Diversity Configuration

The following commands are used to set and get antenna diversity configuration:

**NOTE:** Antenna diversity configuration is currently supported in AW-NM191-uSD (88W8801) module and not supported in PAN9026 (88W8977), AW-AM457-uSD (IW416) and AW-CM358-uSD (88W8987) modules.

Command Usage:

```
# wlan-set-antcfg
```

Usage:

```
wlan-set-antcfg <ant mode> [evaluate_time]
```

<ant mode>:

    Bit 0 -- Tx/Rx antenna 1

    Bit 1 -- Tx/Rx antenna 2

    0xFFFF -- Tx/Rx antenna diversity

[evaluate\_time]:

    if ant mode = 0xFFFF, SAD evaluate time interval,

    default value is 6s(0x1770)

### 3.6.1.9 Set/Get ED MAC Feature

This feature enables the European Union (EU) adaptivity test as per the compliance requirements in the ETSI standard.

Depending on the device and front end loss, the Energy Detection (ED) threshold offset (ed\_ctrl\_2g.offset and ed\_ctrl\_5g.offset) needs to be adjusted. The ED threshold offset can be adjusted in steps of 1 dB.

This section includes definitions of the commands and examples which shows how to adjust ED MAC.

Below are the get and set commands for ED-MAC adjustment.

**#wlan-get-ed-mac-mode**

**#wlan-set-ed-mac-mode <ed\_ctrl\_2g> <ed\_offset\_2g> <ed\_ctrl\_5g> <ed\_offset\_5g>**

Where:

Table 18: ED MAC Parameters

Parameter	Description
ed_ctrl_2_g	0 = disable ED MAC threshold for 2.4GHz band 1 = enable ED MAC threshold for 2.4GHz band
ed_offset_2_g	ED MAC threshold for 2.4 GHz band. Hexadecimal value in units of dB Range: 0x80 to 0x7F, (-128 to 127), 0 = default offset value
ed_ctrl_5_g	0 = disable ED MAC threshold for 5GHz band 1 = enable ED MAC threshold for 5GHz band
ed_offset_5_g	ED MAC threshold for 5 GHz band. Hexadecimal value in units of dB Range: 0x80 to 0x7F, (-128 to 127), 0 = default offset value

#### For 2.4GHz band:

In this example, the 2.4 GHz ED-MAC threshold is lowered by 1 dB.

Table 19: ED MAC 2.4 GHz Command Operations

Step	Operation	Command
1	Get ED-MAC status	<b>#wlan-get-ed-mac-mode</b> EU adaptivity for 2.4GHz band : Enabled Energy Detect threshold offset : 0X9
2	Set ED-MAC threshold	<b>#wlan-set-ed-mac-mode 1 0x8</b> ED MAC MODE settings configuration successful

#### For 5GHz band:

In this example, the 5 GHz ED-MAC threshold is lowered by 2 dB.

Table 20: ED MAC 5 GHz Command Operations

Step	Operation	Command
1	Get ED-MAC status	<b>#wlan-get-ed-mac-mode</b> EU adaptivity for 2.4GHz band : Enabled Energy Detect threshold offset : 0X9 EU adaptivity for 5GHz band : Enabled Energy Detect threshold offset : 0xC
2	Set ED-MAC threshold	<b>#wlan-set-ed-mac-mode 1 0x9 1 0x3</b> ED MAC MODE settings configuration successful

## 4 Useful Wi-Fi APIs

This section describes a few Wi-Fi driver APIs with their usage. These driver APIs can be called from the user application directly with the appropriate arguments to implement the required changes in the driver/firmware.

**NOTE:** Please refer to [wifi\\_cert](#) demo from section 3.6, which has support for these APIs. Please refer to [MCUXSDKGSUG](#) for more details about the Wi-Fi driver APIs.

### 4.1 Set/Get ED MAC Feature

This feature enables the European Union (EU) adaptivity test as per the compliance requirements in the ETSI standard.

Depending on the device and front end loss, the Energy Detection (ED) threshold offset (ed\_ctrl\_2g.offset and ed\_ctrl\_5g.offset) needs to be adjusted. The ED threshold offset can be adjusted in steps of 1 dB.

#### 4.1.1 wlan\_set\_ed\_mac\_mode()

This API is used to configure ED MAC mode in the Wireless Firmware.

**Syntax:** int wlan\_set\_ed\_mac\_mode(wlan\_ed\_mac\_ctrl\_t wlan\_ed\_mac\_ctrl)

Where

Table 21: Set ED MAC API argument

Parameter	Description
[In] wlan_ed_mac_ctrl	A structure with parameters mentioned in section 4.1.3 to enable EU adaptivity.

**Return Value:**

WM\_SUCCESS if the call is successful, -WM\_FAIL if the call failed

#### 4.1.2 wlan\_get\_ed\_mac\_mode()

This API can be used to get current ED MAC mode configuration.

**Syntax:** int wlan\_get\_ed\_mac\_mode(wlan\_ed\_mac\_ctrl\_t \* wlan\_ed\_mac\_ctrl)

Where

Table 22: Get ED MAC API argument

Parameter	Description
[Out] wlan_ed_mac_ctrl	A pointer to a structure with parameters mentioned in section 4.1.3 to get ED MAC mode configuration.

**Return Value:**

WM\_SUCCESS if the call is successful, -WM\_FAIL if the call failed

#### 4.1.3 Usage and Output

This section includes the output console logs and code snippets for the reference and it can be used to add the feature-related commands in the user application.

To add new CLI command in the existing *wifi\_cli* sample application, please refer to section 3.3.2.

##### Usage:

To add `set` command to the command list,

```
#ifdef CONFIG_5GHz_SUPPORT
    {"wlan-set-ed-mac-mode", "<ed_ctrl_2g> <ed_offset_2g> <ed_ctrl_5g>
<ed_offset_5g>", wlan_ed_mac_mode_set},
#else
    {"wlan-set-ed-mac-mode", "<ed_ctrl_2g>
<ed_offset_2g>", wlan_ed_mac_mode_set},
#endif
```

To print the usage regarding `set-ed-mac`

```
static void dump_wlan_set_ed_mac_mode_usage()
{
    PRINTF("Usage:\r\n");
#ifdef CONFIG_5GHz_SUPPORT
    PRINTF("wlan-set-ed-mac-mode <ed_ctrl_2g> <ed_offset_2g> <ed_ctrl_5g>
<ed_offset_5g>\r\n");
#else
    PRINTF("wlan-set-ed-mac-mode <ed_ctrl_2g> <ed_offset_2g>\r\n");
#endif
    PRINTF("\r\n");
    PRINTF("\ted_ctrl_2g \r\n");
    PRINTF("\t # 0      - disable EU adaptivity for 2.4GHz band\r\n");
    PRINTF("\t # 1      - enable EU adaptivity for 2.4GHz band\r\n");
    PRINTF("\ted_offset_2g \r\n");
    PRINTF("\t # 0      - Default Energy Detect threshold\r\n");
    PRINTF("\t #offset value range: 0x80 to 0x7F\r\n");
#endif
    PRINTF("\ted_ctrl_5g \r\n");
    PRINTF("\t # 0      - disable EU adaptivity for 5GHz band\r\n");
    PRINTF("\t # 1      - enable EU adaptivity for 5GHz band\r\n");
    PRINTF("\ted_offset_2g \r\n");
    PRINTF("\t # 0      - Default Energy Detect threshold\r\n");
    PRINTF("\t #offset value range: 0x80 to 0x7F\r\n");
#endif
}
```

To set ed mac mode using the structure parameter in driver (set) API:

```
static void wlan_ed_mac_mode_set(int argc, char *argv[])
{
    int ret;
    wlan_ed_mac_ctrl_t wlan_ed_mac_ctrl;

#ifndef CONFIG_5GHz_SUPPORT
    if (argc != 5)
#else
    if (argc != 3)
#endif
    {
        dump_wlan_set_ed_mac_mode_usage();
        return;
    }

    wlan_ed_mac_ctrl.ed_ctrl_2g    = strtol(argv[1], NULL, 16);
    wlan_ed_mac_ctrl.ed_offset_2g = strtol(argv[2], NULL, 16);
#ifndef CONFIG_5GHz_SUPPORT
    wlan_ed_mac_ctrl.ed_ctrl_5g    = strtol(argv[3], NULL, 16);
    wlan_ed_mac_ctrl.ed_offset_5g = strtol(argv[4], NULL, 16);
#endif

    if (wlan_ed_mac_ctrl.ed_ctrl_2g != 0 && wlan_ed_mac_ctrl.ed_ctrl_2g != 1)
    {
        dump_wlan_set_ed_mac_mode_usage();
        return;
    }
#ifndef CONFIG_5GHz_SUPPORT
    if (wlan_ed_mac_ctrl.ed_ctrl_5g != 0 && wlan_ed_mac_ctrl.ed_ctrl_5g != 1)
    {
        dump_wlan_set_ed_mac_mode_usage();
        return;
    }
#endif

    ret = wlan_set_ed_mac_mode(wlan_ed_mac_ctrl);
    if (ret == WM_SUCCESS)
    {
        PRINTF("ED MAC MODE settings configuration successful\r\n");
    }
    else
    {
        PRINTF("ED MAC MODE settings configuration failed\r\n");
        dump_wlan_set_ed_mac_mode_usage();
    }
}
```

To add get command to the command list,

```
{"wlan-get-ed-mac-mode", NULL, wlan_ed_mac_mode_get},
```

To print the usage regarding get-ed-mac

```
static void dump_wlan_get_ed_mac_mode_usage()
{
    PRINTF("Usage:\r\n");
    PRINTF("wlan-get-ed-mac-mode \r\n");
}
```

To get ed mac mode values filled address of wlan\_ed\_mac\_ctrl structure passed as a parameter to the driver (get) API,

```
static void wlan_ed_mac_mode_get(int argc, char *argv[])
{
    int ret;
    wlan_ed_mac_ctrl_t wlan_ed_mac_ctrl;

    if (argc != 1)
    {
        dump_wlan_get_ed_mac_mode_usage();
        return;
    }

    ret = wlan_get_ed_mac_mode(&wlan_ed_mac_ctrl);
    if (ret == WM_SUCCESS)
    {
        PRINTF("EU adaptivity for 2.4GHz band : %s\r\n",
wlan_ed_mac_ctrl.ed_ctrl_2g == 1 ? "Enabled" : "Disabled");
        if (wlan_ed_mac_ctrl.ed_ctrl_2g)
            PRINTF("Energy Detect threshold offset : 0X%x\r\n",
wlan_ed_mac_ctrl.ed_offset_2g);
#ifdef CONFIG_5GHz_SUPPORT
        PRINTF("EU adaptivity for 5GHz band : %s\r\n",
wlan_ed_mac_ctrl.ed_ctrl_5g == 1 ? "Enabled" : "Disabled");
        if (wlan_ed_mac_ctrl.ed_ctrl_5g)
            PRINTF("Energy Detect threshold offset : 0X%x\r\n",
wlan_ed_mac_ctrl.ed_offset_5g);
#endif
        else
        {
            PRINTF("ED MAC MODE read failed\r\n");
            dump_wlan_get_ed_mac_mode_usage();
        }
    }
}
```

#### Console Output

```
# wlan-set-ed-mac-mode 1 0x9
ED MAC MODE settings configuration successful
```

```
# wlan-get-ed-mac-mode
EU adaptivity for 2.4GHz band : Enabled
Energy Detect threshold offset : 0x9
EU adaptivity for 5GHz band : Enabled
Energy Detect threshold offset : 0xc
```

## 5 Bluetooth Classic/Low Energy Applications

This chapter describes the Bluetooth Classic/Low Energy example applications that are available in the SDK, and the steps to configure, compile, debug, flash, and execute these examples.

The communication between the Host stack and the Link Layer (LL) is implemented via the standard HCI UART interface and PCM interface for voice.

Please refer to “*Hardware Rework Guide for EdgeFast BT PAL.pdf*” guide referenced in the Section 1.3 “References” for details to enable the UART and PCM interfaces.

The setup is done between the single i.MX RT+ IW416 NXP-based wireless module and remote Bluetooth devices. The instructions in this guide use an i.MXRT1060 EVK board. Yet the same steps apply to the other i.MX RT products.

The table lists the Bluetooth module specific preprocessor macro that is common to all Bluetooth examples.

Table 23: Preprocessor Macros for Bluetooth Modules

Module	Chipset	Macro
AzureWave AW-AM457-S-uSD	IW416	WIFI_IW416_BOARD_AW_AM457_USD
AzureWave AW-CM358-uSD	88W8987	WIFI_88W8987_BOARD_AW_CM358_USD
AzureWave AW-AM510-uSD	IW416	WIFI_IW416_BOARD_AW_AM510_USD
Murata Type 1XX	IW416	WIFI_IW416_BOARD_MURATA_1XX_USD WIFI_IW416_BOARD_MURATA_1XX_M2
Murata Type 1ZM	88W8987	WIFI_88W8987_BOARD_MURATA_1ZM_USD WIFI_88W8987_BOARD_MURATA_1ZM_M2

USD = microSD interface

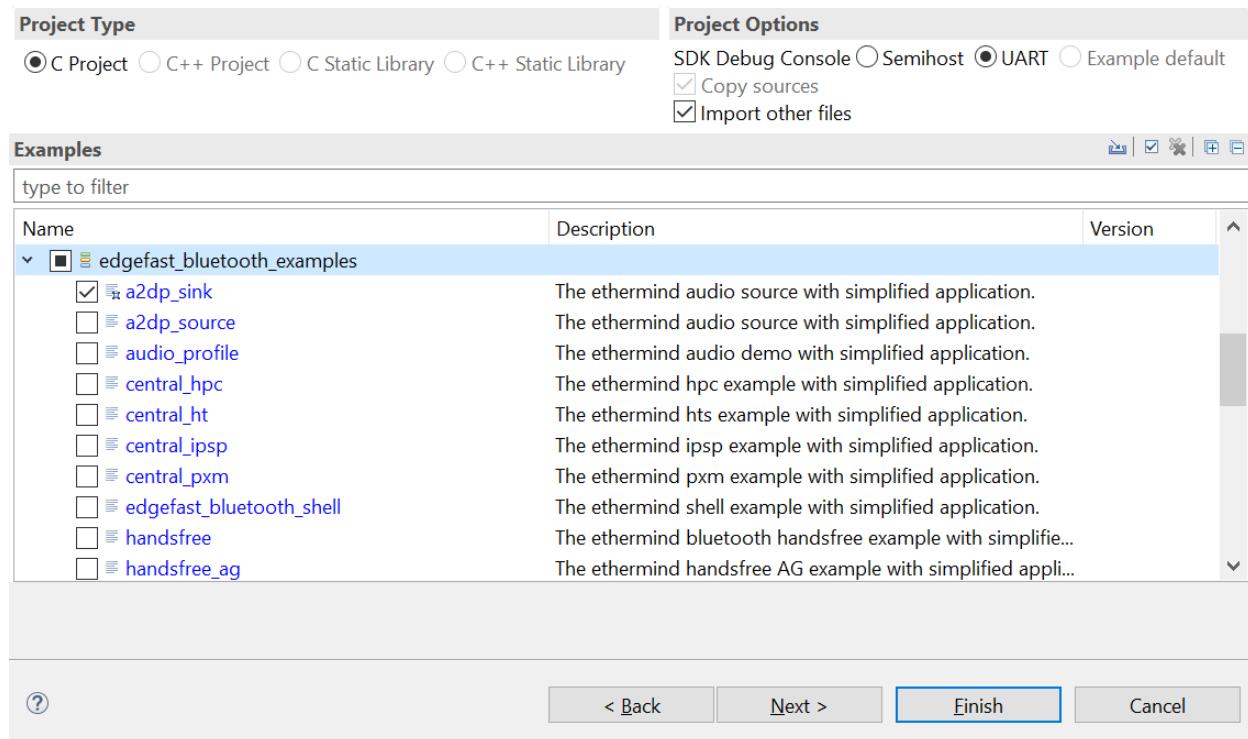
M2 = M.2 interface

## 5.1 a2dp\_sink Sample Application

This sample application section describes the steps to configure the i.MX RT1060 EVK board and IW416 wireless module as an A2DP Sink device.

### 5.1.1 a2dp\_sink Application Execution

Please refer to the previous section 3.1.3 to run the demo using MCUXpresso IDE. Refer below image for selection of Bluetooth example.



Please refer to the previous sections 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.1.1.1 Run the Application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized
BR/EDR set connectable and discoverable done
```

Discover the device “a2dp sink” from peer mobile phone and connect to it. The following logs should be displayed on the console.

```
Connected
Security changed: 7E:5A:23:AE:9E:C3 (0xed) level 2
a2dp connected success
```

Now, user can play music from the cell phone connected and listen on the audio jack of the i.MX RT 1060 EVK board.

Following logs will appear on the console:

```
Init Audio SAI and CODEC
a2dp start playing
```

Stop playing music from the cell phone.

Following logs will appear on the console.

```
Stop SAI and CODEC  
a2dp stop playing
```

```
Disconnect the device from peer cell phone.  
Disconnected (reason 0x13)
```

## 5.2 a2dp\_source Sample Application

This section describes the steps to configure the i.MX RT1060 EVK board and IW416 wireless module as an A2DP Source device.

### 5.2.1 a2dp\_source Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.2.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
USB Host stack successfully initialized  
Bluetooth initialized
```

```
SHELL build: Mar 9 2021  
Copyright 2021 NXP
```

```
>>
```

Input "help" to show the available list of commands:

```
>>  
help  
  
"help": List all the registered commands  
  
"exit": Exit program  
  
"bt": BT related function  
    USAGE: bt [discover|connect|disconnect|delete]  
          discover      start to find BT devices  
          connect       connect to the device that is found, for example: bt  
connectdevice n (from 1)  
          disconnect   disconnect current connection.  
          delete       delete all devices. Ensure to disconnect the HCI link  
connection with the peer device before attempting to delete the bonding  
information.  
>>
```

Input "bt discover" to scan connectable nearby Bluetooth devices.

```
>> bt discover  
Discovery started. Please wait ...  
>> BR/EDR discovery complete  
[1]: 20:39:56:C6:6C:6C, RSSI -89 Nokia 6.1 Plus  
[2]: B8:F6:53:E8:BF:B7, RSSI -84 JBL Flip 5  
[3]: 70:F0:87:C0:FC:0E, RSSI -100  
>>
```

Input "bt connect [index]" to create Bluetooth connection with the discovered device. The music starts playing on successful connection with the Bluetooth device.

```
>> bt connect 2  
Connection pending  
>> SDP discovery started  
Connected  
sdp success callback  
A2DP Service found. Connecting ...
```

```
Security changed: 7E:5A:23:AE:9E:C3 (0xed) level 2
a2dp connected success
a2dp start playing
```

Input “bt disconnect” to disconnect the current connection.

```
>> bt disconnect
>> a2dp disconnected
Disconnected (reason 0x16)
```

Input “bt delete” to delete the bonding information of all the devices.

**NOTE:** Disconnect the HCI link connection with the peer device before attempting to delete the bonding information.

```
>> bt delete
success
>>
```

## 5.3 handsfree Sample Application

This section describes the steps to configure the i.MX RT1060 EVK board and IW416 wireless module as an HF Unit.

### 5.3.1 handsfree Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.3.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized
BR/EDR set connectable and discoverable done
Please use "Y" to accept the call, use "N" to reject or end the call
```

Discover the device “edgefast hfp” from peer mobile phone and connect to it. The following logs should be displayed on the console.

```
HFP HF Connected!
Wideband Config at Controller: Disabled
Sending Vendor command 0028
Sending Vendor command 0007 now
  Sending Vendor command 0029 now
  Sending Vendor command 001d now
  Sending Vendor command 0070 now
Service indicator value: 1
Signal indicator value: 5
```

Make an incoming call to the mobile phone which is connected to the setup:

```
Call Setup indicator value: 1
Incoming Call...
Init Audio CODEC for RingTone
```

Input “Y” to answer the incoming call:

```
Answered the incoming call.
Call indicator value: 1
Call Setup indicator value: 0
Init Audio SCO SAI and CODEC samplingRate :8000 bitWidth:16
Setup for SCO audio: Success
Sending Vendor command 006f now
mic dummy
```

Input “N” to hang-up the active call:

```
Ended the activating call.
Call indicator value: 0
```

Make an incoming call to the mobile phone again and input “N” to reject the incoming call.

```
Rejected the incoming call.  
Call Setup indicator value: 0
```

## 5.4 handsfree\_ag Sample Application

This application demonstrates the HFP audio gateway basic functionality. Currently, the support simulates an incoming call, and the call could be answered and ended.

The HFP audio gateway can be connected to a HFP HF device like headphone or device running HFP HF device.

### 5.4.1 handsfree\_ag Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.4.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
USB Host stack successfully initialized  
Bluetooth initialized
```

```
SHELL build: Mar 18 2021  
Copyright 2021 NXP
```

```
>>
```

Input “help” to show the available list of commands:

```
>> help
```

```
"help": List all the registered commands
```

```
"exit": Exit program
```

```
"bt": BT related function  
    USAGE: bt [discover|connect|disconnect|delete]  
          discover      start to find BT devices  
          connect       connect to the device that is found, for example: bt connect n  
          (from 1)  
          sincall      start an incoming call.  
          aincall      accept the incoming call.  
          eincall      end an incoming call.  
          disconnect   disconnect current connection.  
          delete       delete all devices. Ensure to disconnect the HCI link  
connection with the peer device before attempting to delete the bonding  
information.  
>>
```

Input "bt discover" to scan the nearby Bluetooth devices:

```
>> bt discover
Discovery started. Please wait ...
>> BR/EDR discovery complete
[1]: 50:82:D5:78:31:DA, RSSI -65 iPhone 6
[2]: 80:CE:B9:22:B3:FD, RSSI -79 Galaxy S8
[3]: BC:17:B8:74:2C:9F, RSSI -58 LAPTOP
[4]: 70:F0:87:C0:FC:0E, RSSI -66 iPhone
[5]: 00:00:AB:CD:87:D6, RSSI -38 Airdopes 441
[6]: C8:3D:D4:43:5A:14, RSSI -93 LAPTOP-NLHDBT0E
```

>>

Input "bt connect <number>" to connect to the peer device.

```
>> bt connect 5
Connection pending
>> SDP discovery started
Connected
Security changed: 7A:D6:2C:9B:F7:A3 (0x5e) level 2
HFP AG Connected!
```

Input "bt disconnect" to disconnect from the peer device.

```
>> bt disconnect
>> HFP AG Disconnected!
Disconnected (reason 0x16)
```

Input "bt delete" to delete the bonding information of all the devices.

**Note:** Disconnect the HCI link connection with the peer device before attempting to delete the bonding information.

```
>> bt delete
success
>>
```

## 5.5 spp Sample Application

This application demonstrates the Serial Port Profile on i.MX RT1060 EVK board and IW416 wireless module.

### 5.5.1 spp Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.5.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
USB Host stack successfully initialized
Bluetooth initialized
BR/EDR set connectable and discoverable done
```

SHELL build: Mar 10 2021

Copyright 2021 NXP

>>

Input "help" to display the available options:

```
>> help
```

"help": List all the registered commands

"exit": Exit program

```

"bt": BT related function
  USAGE: bt [discover|connect|disconnect|delete]
    bt discover      start to find BT devices
    bt connect       connect to the device that is found, for example: bt
connectdevice n (from 1)
    bt disconnect   disconnect current connection.
    bt delete        delete all devices. Ensure to disconnect the HCI link
connection with the peer device before attempting to delete the bonding
information.

"spp": SPP related function
  USAGE:
    spp register [5|3]      register a spp server channel
    spp discover            discover spp server channel on peer device
    spp connect [channel]   create spp connection
    spp disconnect          disconnect current spp connection.
    spp send [1|2|3|4]       send data over spp connection.
>>

```

### 5.5.1.2 Serial Port Profile Server Configuration

This section describes the steps to configure the i.MX RT1060 EVK board and IW416 wireless module as an SPP Server.

Register a SPP server channel.

```
>> spp register 5
Register spp server channel 5 successfully!
>>
```

Connect to the device “edgefast spp” from the smartphone Bluetooth pairing settings and enable the pairing.

Following logs will appear on console:

```
>> Connected
Security changed: 7E:5A:2B:2E:9E:C3 (0x6d) level 2
Disconnected (reason 0x13)
```

Now, open the “Serial Bluetooth Terminal” smartphone application and go to settings > devices.

Select the device “edgefast spp”. The connection will be established and following logs will appear on console:

```
Connected
Security changed: 7E:5A:2B:2E:9E:C3 (0x6d) level 2
Security changed: 7E:5A:2B:2E:9E:C3 (0x6d) level 2
SPP connection is created successfully!
```

Write data in the smartphone application and send:

```
Status of SPP data received callback: 0x0000.
Received 34 data, dumped here:

-----CHAR DUMP-----
H e l l o   f r o m   s m a r t p h o n e .

- ö ý ÿ ÿ ÿ þ ý ÿ

-----HEX DUMP-----
48 65 6C 6C 6F 20 66 72 6F 6D 20 73 6D 61 72 74 70 68 6F 6E 65 2E 0D 0A AC F5
FF FF FF FF FE FF FF 08
```

Input “spp send [n]” to send data to peer device.

```
>> spp send 1
>>
Status of SPP data sent callback: 0x0000.
```

```
Sent 11 data, dumped here:
```

```
-----CHAR DUMP-----
A T + C I N D = ? \ r
-----
```

Input “spp disconnect” to disconnect with peer device.

```
>> spp disconnect
>> SPP connection is disconnected successfully!
```

#### 5.5.1.3 Serial Port Profile Client Configuration

This section describes the steps to configure the i.MX RT1060 EVK board and IW416 wireless module as an SPP Client. Here, another setup of i.MX RT1060 EVK board and IW416 wireless module is used as SPP Server.

Start SPP server first then follow the steps to configure SPP client.

Input “bt discover” to start find the nearby Bluetooth devices.

```
>> bt discover
Discovery started. Please wait ...
>> BR/EDR discovery complete
[1]: C8:3D:D4:43:5A:14, RSSI -86 LAPTOP-NLHDBT0E
[2]: 04:D1:3A:82:B5:FE, RSSI -65 LAPTOP-NLHDBT0F
[3]: 18:5E:0F:96:8C:31, RSSI -69 BUILD0
[4]: 20:4E:F6:EC:1F:26, RSSI -79 edgefast spp
>>
```

Input “bt connect <n>” to connect to the device that is found.

```
>> bt connect 4
Connection pending
>> Connected
>>
```

Input “spp discover” to discover the registered SPP server channel in peer device.

```
>> spp discover
>> Discover 1 SPP server channel from device 26:1F:EC:F6:4E:20!
0x0005
>>
```

Input “spp connect [channel]” to create SPP connection with peer SPP server channel.

```
>> spp connect 5
Connect SPP Successful!
>> Security changed: 84:17:25:08:64:F9 (0xef) level 2
SPP connection is created successfully!
>>
```

Input “spp send [1|2|3|4]” to send data over SPP.

```
>> spp send 1
>>
Status of SPP data sent callback: 0x0000.
Sent 11 data, dumped here:
-----CHAR DUMP-----
A T + C I N D = ? \ r
-----
```

Input “spp disconnect” to disconnect with peer device.

```
>> spp disconnect
>> SPP connection is disconnected successfully!
>>
```

## 5.6 peripheral\_hps Sample Application

This application demonstrates the Bluetooth LE Peripheral role, except that this application specifically exposes the HTTP Proxy GATT Service.

### 5.6.1 peripheral\_hps Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.6.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized  
Advertising successfully started
```

The demo does not require user interaction.

The application will automatically start advertising the HTTP Proxy Service and it will accept the first connection request it receives. The application is then ready to process HTTP requests from the peer.

The application simulates processing of the HTTP request. It will always return HTTP Status Code 500 and preset values for HTTP Headers and HTTP Body.

```
Connected to peer: 20:4E:F6:EC:1F:26 (public)  
Processing request..  
Request processed.
```

## 5.7 central\_hpc Sample Application

This application demonstrates very basic Bluetooth LE Central role functionality on i.MX RT1060 EVK board and IW416 wireless module by scanning for other Bluetooth LE devices and establishing a connection to the first one with a strong enough signal.

Except that this application specifically looks for HPS Server and programs a set of characteristics that configures a Hyper Text Transfer Protocol (HTTP) request, initiate this request, and then read the response once connected.

Here, another setup of i.MX RT1060 EVK board and IW416 wireless module is used as *peripheral\_hps*.

### 5.7.1 central\_hpc Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.7.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized  
Scanning started  
[DEVICE]: 20:4E:F6:25:F3:18 (public), AD evt type 0, AD data len 7, RSSI -71
```

The demo does not require user interaction.

The application will automatically start scanning and will connect to the first advertiser who is advertising the HTTP Proxy Service.

If the connection is successful, the application performs service discovery to find the characteristics of the HTTP Proxy Service. If discovery is successful, the application will perform a GET for the URI <http://nxp.com> by writing the URI and the Control Point characteristics of the HTTP Proxy Service.

The application will display the received response in the console after it gets notified through the HTTP Status Code characteristic.

```
Found device: Connected to peer: 20:4E:F6:25:F3:18 (public)
Starting service discovery
GATT Write successful
Subscribed to HTTP Status Code
GATT Write successful
Received HTTP Status 500
Reading Headers..
HTTP Headers: HTTPHEADER
Reading Body...
Unsubscribed
HTTP Body: HTTPBODY
```

## 5.8 peripheral\_pxr Sample Application

This application demonstrates the BLE Peripheral role on i.MX RT1060 EVK board and IW416 wireless module. Except that this application specifically exposes the Proximity Reporter (including LLS, IAS, and TPS) GATT Service.

### 5.8.1 peripheral\_pxr Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.8.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized
Advertising successfully started
```

The demo does not require user interaction.

The application will automatically start advertising the Link Loss Service and it will accept the first connection request it receives. The application is then ready to process operations from the peer.

The application will initially set the default levels for the Link Loss Alert and the Immediate Alert.

```
Connected to peer: 20:4E:F6:25:F3:18 (public)
Locally setting Link Loss Alert Level to OFF
Locally setting Immediate Alert...
```

```
ALERT: OFF
```

The Proximity Monitor peer will trigger or stop the Immediate Alert on the application depending on the connection RSSI.

```
Monitor is setting Link Loss Alert Level to HIGH
Monitor is setting Immediate Alert...
ALERT: HIGH
Monitor is setting Immediate Alert...
ALERT: OFF
```

If the connection with the Proximity Monitor is timed out, the Link Loss Alert will be triggered with the level previously set by the Monitor.

```
Link Loss Alert Triggered...
ALERT: HIGH
```

## 5.9 central\_pxm Sample Application

This application demonstrates very basic Bluetooth LE Central role functionality on i.MX RT1060 EVK board and IW416 wireless module by scanning for other Bluetooth LE devices and establishing a connection to the first one with a strong enough signal.

Except that this application specifically looks for Proximity Reporter.

Here, another setup of i.MX RT1060 EVK board and IW416 wireless module is used as *peripheral\_pxr*.

### 5.9.1 central\_pxm Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.9.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized  
Scanning started
```

The application will automatically start scanning and will connect to the first advertiser who is advertising the Link Loss Service.

If the connection is successful, the application performs service discovery to find the characteristics of the Link Loss Service, as well as additional services and characteristics specified by the Proximity Profile, such as Immediate Alert and Tx Power services.

```
[DEVICE]: 20:4E:F6:25:F3:18 (public), AD evt type 0, AD data len 11, RSSI -83  
Found device: Connected to peer: 20:4E:F6:25:F3:18 (public)  
Starting service discovery  
GATT Write successful  
Connection RSSI: -81
```

If the Tx Power service and its characteristics have been discovered, the application will read the peer's Tx power and display it.

```
Read successful - Tx Power Level: 0
```

If the Immediate Alert service and its characteristics have been discovered, the application will continuously monitor the connection RSSI and will trigger or stop the Immediate Alert on the peer when the value is crossing a preset threshold in either direction.

```
Connection RSSI: -81  
Connection RSSI: -81  
Connection RSSI: -81  
Connection RSSI: -81  
Connection RSSI: -81
```

After the mandatory Link Loss service is discovered, the application will write the Link Loss Alert Level on the peer as HIGH\_ALERT.

To trigger the Link Loss Alert on the peer, the connection will have to be timed out. The user can trigger this by simply resetting the board (press the RST button).

## 5.10 peripheral\_ht Sample Application

This application demonstrates the BLE Peripheral role on i.MX RT1060 EVK board and IW416 wireless module. Except that this application specifically exposes the HT (Health Thermometer) GATT Service. Once a device connects it will generate dummy temperature values.

### 5.10.1 peripheral\_ht Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.10.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board.

When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized  
Advertising successfully started
```

The application does not require user interaction.

The application will automatically start advertising the Health Thermometer Service and it will accept the first connection request it receives. If the peer subscribes to receive temperature indications, these will be sent every 1 second.

The temperature readings are simulated with values between 20 and 25 degrees Celsius.

```
Connected to peer: 20:4E:F6:EC:1F:26 (public)  
temperature is 21C  
Indication success  
temperature is 22C  
Indication success
```

## 5.11 central\_ht Sample Application

This application demonstrates very basic Bluetooth LE Central role functionality on i.MX RT1060 EVK board and IW416 wireless module by scanning for other Bluetooth LE devices and establishing a connection to the first one with a strong enough signal.

Except that this application specifically looks for health thermometer sensor and reports the temperature readings once connected.

Here, another setup of i.MX RT1060 EVK board and IW416 wireless module is used as *peripheral\_ht*.

### 5.11.1 central\_ht Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.11.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized  
Scanning started
```

The demo does not require user interaction.

The application will automatically start scanning and will connect to the first advertiser who is advertising the Health Thermometer Service. If the connection is successful, the application performs service discovery to find the characteristics of the Health Thermometer Service.

If discovery is successful, the application will subscribe to receive temperature indications from the peer.

The application will display the received indications in the console.

```
[DEVICE]: 20:4E:F6:25:F3:18 (public), AD evt type 0, AD data len 9, RSSI -84
Found device: Connected to peer: 20:4E:F6:25:F3:18 (public)
Starting service discovery
Subscribed to HTS
Temperature 21 degrees Celsius
Temperature 22 degrees Celsius
```

## 5.12 peripheral\_ipsp Sample Application

This application demonstrates the BLE Peripheral role on i.MX RT1060 EVK board and IW416 wireless module. Except that this application specifically exposes the Internet Protocol Support GATT Service.

### 5.12.1 peripheral\_ipsp Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.12.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board.

When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized
Advertising successfully started
IPSS Service ready
```

The demo does not require user interaction.

The application will automatically start advertising the IPSP Service and it will accept the first connection request it receives.

The application will perform the required setup for the L2CAP credit-based channel specified by the IPSP Profile. The application will display in console any message it receives from the peer through the L2CAP channel.

```
Connected to peer: 20:4E:F6:EC:1F:26 (public)
Received message: hello
Received message: hello
```

## 5.13 central\_ipsp Sample Application

This application demonstrates very basic BLE Central role functionality by scanning for other BLE devices and establishing a connection to the first one with a strong enough signal.

Except that this application specifically looks for IPSP Service and communicates between the devices that support IPSP is done using IPv6 packets over the Bluetooth Low Energy transport once connected.

Here, another setup of i.MX RT1060 EVK board and IW416 wireless module is used as *peripheral\_ipsp*.

### 5.13.1 central\_ipsp Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

### 5.13.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Bluetooth initialized  
Scanning started
```

The demo does not require user interaction.

The application will automatically start scanning and will connect to the first advertiser who is advertising the IPSP Service.

After the L2CAP credit-based channel specified by the IPSP Profile is established, the application will send a predefined test message every 5 seconds through the channel.

```
[DEVICE]: 20:4E:F6:25:F3:18 (public), AD evt type 0, AD data len 7, RSSI -82  
Found device: Connected  
Starting service discovery  
Sending message...  
Sending message...
```

## 5.14 Wireless UART Sample Application

The application implements a custom GATT based Wireless UART Profile that emulates UART over BLE. Central and peripheral role can be switched by user button (SW8). To test the service/profile the "IoT Toolbox" application can be used which is available for both Android and iOS. IoT Toolbox can be found on Apple App Store or Google Play Store.

### 5.14.1 wireless\_uart Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode, and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.14.1.1 Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
BLE Wireless Uart demo start...  
Bluetooth initialized  
Advertising successfully started
```

The demo requires user interaction. The application will automatically start advertising the wireless uart Service after reset, the application can only accept 1 connection when configured as a peripheral.

The application will start scanning and connect to the wireless uart Service automatically.

Pressing the Button will switch from Peripheral mode to central mode and now it can connect upto 8 devices. We can use "IoT Toolbox" or another wireless\_uart example (use B to refer to) to test the current device.

**peripheral role test:**

Open "IoT Toolbox" application on an Android or iOS smartphone, select the "Wireless UART" option. A device named "NXP\_WU" should appear. Connect to "NXP\_WU" by selecting the device from the scan list. The Android/iOS device should receive a prompt for a Bluetooth Pairing Request. Please complete the pairing process by entering the paskey that is displayed on the debug terminal. Once pairing is completed, we can now transmit and receive data over the emulated UART interface.

```
BLE Wireless Uart demo start...
Bluetooth initialized
Advertising successfully started
Connected to 4B:6B:F0:B6:7C:F8 (random)
GATT MTU exchanged: 65
[ATTRIBUTE] handle 40
[ATTRIBUTE] handle 41
Passkey for 4B:6B:F0:B6:7C:F8 (random): 994660
Security changed: 20:39:56:C6:6C:6C (public) level 4 (error 0)
Data received (length 5): hello
```

**central role test:**

let B work as default state after reset.

short press the user button(SW8), the example will work as central can automatically connect to any discovered wireless uart example. Each time short press, the example will scan and connect to wireless uart service if new device is found.

```
BLE Wireless Uart demo start...
Bluetooth initialized
Advertising successfully started
Scanning successfully started
[DEVICE]: 24:FC:E5:9F:EE:EB (public), AD evt type 3, AD data len 28, RSSI -92
[DEVICE]: 64:86:7F:5A:7C:7F (random), AD evt type 0, AD data len 23, RSSI -81
[DEVICE]: 64:86:7F:5A:7C:7F (random), AD evt type 4, AD data len 0, RSSI -80
[DEVICE]: 65:F2:7E:9A:AF:C7 (random), AD evt type 0, AD data len 19, RSSI -89
[DEVICE]: 65:F2:7E:9A:AF:C7 (random), AD evt type 4, AD data len 0, RSSI -89
[DEVICE]: 63:F2:B1:6A:FC:3D (random), AD evt type 0, AD data len 18, RSSI -80
[DEVICE]: 63:F2:B1:6A:FC:3D (random), AD evt type 4, AD data len 0, RSSI -80
[DEVICE]: 78:B3:AA:89:78:3B (random), AD evt type 0, AD data len 18, RSSI -80
[DEVICE]: 78:B3:AA:89:78:3B (random), AD evt type 4, AD data len 0, RSSI -79
[DEVICE]: 80:D2:1D:E8:2B:7E (public), AD evt type 0, AD data len 21, RSSI -43
Connected to 80:D2:1D:E8:2B:7E (public)
GATT MTU exchanged: 65
[ATTRIBUTE] handle 25
[ATTRIBUTE] handle 26
Security changed: 80:D2:1D:E8:2B:7E (public) level 2 (error 0)
```

**Note:**

*The device address, AD event type data len, and RSSI are variable, it depends on all the Bluetooth device in test environment.*

Send data 12345 in B device's Serial port terminal, then current device will print the following log.

```
Data received (length 5): 12345
```

Send data 123 in current device's Serial port terminal, then B device will print the following log.

```
Data received (length 5): 123
```

## 5.15 Shell Sample Application

Application Demonstrating the Interactive Shell Mode of Bluetooth Commands and APIs. It provides users full control over the Bluetooth Interface. User can control the basic Bluetooth operations such as advertising/scanning , device discovery , connection and pairing as well as direct access to the HCI command interface.

### 5.15.1 Shell Application Execution

Please refer to the previous sections 3.1.3-3.1.6 and 5.1.1 for instructions on importing a project, building an application, running an application in debug mode, and flashing an application program for a few IDEs. Please refer to section 2.1 for serial console tool setup.

#### 5.15.1.1 Shell Run the application

Press the power reset button on i.MX RT1060 EVK board to run the demo application downloaded on the board. When the demo starts, the following message about the demo would appear on the console.

```
Edgefast Bluetooth PAL shell demo start...
SHELL build: Aug 10 2021
Copyright 2020 NXP
@bt>
```

The shell command list can be accessed by typing "help" in serial terminal. The demo can be configured to either "central" or "peripheral" by shell commands.

Here is an example of scan devices (the BLE host must be initialized before executing the scan command):

```
@bt> bt.init
@bt> Bluetooth initialized
Settings Loaded

@bt> bt.scan on
Bluetooth active scan enabled
@bt> [DEVICE]: 44:6D:F5:85:DC:5F (random), AD evt type 0, RSSI -64 C:1 S:1 D:0
SR:0 E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 44:6D:F5:85:DC:5F (random), AD evt type 4, RSSI -63 C:0 S:1 D:0 SR:1
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 6D:B3:D3:8E:ED:A2 (random), AD evt type 0, RSSI -77 C:1 S:1 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 6D:B3:D3:8E:ED:A2 (random), AD evt type 4, RSSI -76 C:0 S:1 D:0 SR:1
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 3F:FB:95:F7:F9:14 (random), AD evt type 3, RSSI -75 C:0 S:0 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 49:A3:4E:86:63:0C (random), AD evt type 0, RSSI -76 C:1 S:1 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 49:A3:4E:86:63:0C (random), AD evt type 4, RSSI -75 C:0 S:1 D:0 SR:1
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 5C:28:50:F9:DD:57 (random), AD evt type 0, RSSI -82 C:1 S:1 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 4A:7D:B4:12:7B:7A (random), AD evt type 0, RSSI -82 C:1 S:1 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 4A:7D:B4:12:7B:7A (random), AD evt type 4, RSSI -82 C:0 S:1 D:0 SR:1
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 5A:54:C8:99:13:4A (random), AD evt type 0, RSSI -76 C:1 S:1 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
```

```
[DEVICE]: 3B:95:00:4D:F3:EB (random), AD evt type 3, RSSI -82 C:0 S:0 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
[DEVICE]: 47:9D:D0:CB:5F:0D (random), AD evt type 0, RSSI -86 C:1 S:1 D:0 SR:0
E:0 Prim: LE 1M, Secn: No packets, Interval: 0x0000 (0 ms), SID: 0xff
@bt> bt.scan off
Scan successfully stopped
@bt>
```

Here is an example of advertising (the BLE host must be initialized before):

```
@bt> bt.init
@bt> Bluetooth initialized

@bt> bt.advertise on
Advertising started
@bt> bt.advertise off
Advertising stopped
@bt>
```

*NOTE. the mentioned "command complete event" can be found in HCI log, U-DISK should be connected to usb port to get HCI log capture. CONFIG\_BT\_SNOOP macro is used to enable stack to capture the HCI log.*

Here is the log of rf\_test\_mode application:

```
>> help

@bt> help
help

+---"help": List all the registered commands
+---"exit": Exit program
+---"echo": Set echo(0 - disable, 1 - enable)
+---"bt": bt "command entry"
    +---"init": init "[no-settings-load], [sync]"
    +---"settings-load": settings-load "[none]"
    +---"id-create": id-create "[addr]"

    .....
    +---"ca_test": ca_test "<test cover art>"
+---"bt_test": bt_test "Bluetooth BR/EDR test mode commands"
    +---"tx_test": tx_test "test_scenario[1] hopping_mode[1] tx_channel[1]
rx_channel[1] tx_test_interval[1] pkt_type[1] data_length[2] whitening[1]
num_pkt[4] tx_pwr[1]"
    +---"rx_test": rx_test "test_scenario[1] tx_channel[1] rx_channel[1]
pkt_type[1] num_pkt[4] data_length[2] tx_addr[6] report_err_pkt[1]"
    +---"reset": reset " Reset the HCI interface"
+---"le_test": le_test "Bluetooth BLE test mode commands"
    +---"tx_test": tx_test "tx_channel[1] data_length[1] payload[1] phy[1]"
    +---"rx_test": rx_test "rc_channel[1] phy[1] modulation[1]"
    +---"end_test": end_test "end the le test"
@bt>
>>
>> bt.init
bt.init

>>
>> Bluetooth initialized
Settings Loaded

>>
```

**To start the transmitter test packets for Bluetooth Classic:**

```
@bt> bt_test.tx_test 01 01 01 01 0D 03 0F 00 00 00 00 00 00 00 04  
rx_on_start default set to=80  
synt_on_start default set to=80  
tx_on_start default set to=80  
phd_off_start default set to=80  
test_scenario= 1  
hopping_mode= 1  
tx_channel= 1  
rx_channel= 1  
tx_test_interval= d  
pkt_type= 3  
data_length= f 0  
whitening= 0  
num_pkt= 0 0 0 0  
tx_pwr= 4  
@bt> API returned success...
```

Observe the DM1 packets over the air logs.

**To stop the transmitter test packets for Bluetooth Classic:**

```
@bt> bt_test.tx_test FF 01 01 01 0D 03 0F 00 00 00 00 00 00 00 04  
rx_on_start default set to=80  
@bt> synt_on_start default set to=80  
tx_on_start default set to=80  
phd_off_start default set to=80  
test_scenario= ff  
hopping_mode= 1  
tx_channel= 1  
rx_channel= 1  
tx_test_interval= d  
pkt_type= 3
```

```
data_length= f 0
whitening= 0
num_pkt= 0 0 0 0
tx_pwr= 4
API returned success...
```

Observe that DM1 packets are now stopped transmitting in over the air logs.

#### To perform HCI reset

```
@bt> bt_test.reset
API returned success...
```

#### To start the receiving test packets for Bluetooth Classic:

```
@bt> bt_test.rx_test 01 01 01 03 10 00 00 00 0F 00 20 4E F6 EC 1F 26 00
test_scenario= 1

tx_channel= 1

rx_channel= 1

pkt_type= 3

num_pkt= 10 0 0 0

data_length= f 0

tx_am_addr default set to= 1

tx_addr:
20
4e
f6
ec
1f
26

report_err_pkt= 0

@bt> API returned success...
```

#### To stop receiving test packets for Bluetooth Classic:

```
@bt> bt_test.rx_test FF 01 01 03 10 00 00 00 0F 00 20 4E F6 EC 1F 26 00
test_scenario= ff

tx_channel= 1

rx_channel= 1

pkt_type= 3

num_pkt= 10 0 0 0
```

```
data_length= f 0
tx_am_addr default set to= 1
tx_addr:
20
4e
f6
ec
1f
26
report_err_pkt= 0
API returned success...
```

Observe the packet count in vendor-specific command complete event.

**To start the transmitter test packets for Bluetooth LE:**

```
@bt> le_test.tx_test 01 FF 00 01
tx_channel= 1
test_data_len= ff
pkt_payload= 0
phy= 1
@bt> API returned success...
```

Observe the transmitter test packets over the air logs.

**To stop the transmitter test packets for Bluetooth LE:**

```
@bt> le_test.end_test
API returned success...
>>
```

**To start the receiving test packets for Bluetooth LE:**

```
@bt> le_test.rx_test 01 01 00
rx_channel= 1
phy= 1
modulation_index= 0
@bt> API returned success...
```

**To stop the receiver test packets for Bluetooth LE:**

```
@bt> le_test.end_test
API returned success...
```

&gt;&gt;

Observe the packet count in command complete event.

### Running a2dp

The commands are as follows:

```
+---"a2dp": a2dp Bluetooth A2DP shell commands
    +---"register_sink_ep": register_sink_ep <select codec.
1:SBC
2:MPEG-1,2
3:MPEG-2,4
4:vendor
5:sbc with delay report and content protection services
6:sbc with all other services(don't support data transfer yet)>
    +---"register_source_ep": register_source_ep <select codec.
1:SBC
2:MPEG-1,2
3:MPEG-2,4
4:vendor
5:sbc with delay report and content protection services
6:sbc with all other services(don't support data transfer yet)>
    +---"connect": connect [none]
    +---"disconnect": disconnect [none]
    +---"configure": configure [none]
    +---"discover_peer_eps": discover_peer_eps [none]
    +---"get_registered_eps": get_registered_eps [none]
    +---"set_default_ep": set_default_ep <select endpoint>
    +---"configure_ep": configure_ep "configure the default selected ep"
    +---"deconfigure": deconfigure "de-configure the default selected ep"
    +---"start": start "start the default selected ep"
    +---"stop": stop "stop the default selected ep"
    +---"send_media": send_media <second> "send media data to the default
selected ep"
```

### Test flow:

1. Create ACL connection between two devices (A and B).
2. In device B, input "a2dp.register\_sink\_ep x" to initialize sink endpoint.
3. In device A, input "a2dp.register\_source\_ep x" to initialize source endpoint.
4. In device A, input "a2dp.connect" to create a2dp connection with the default ACL connection.
5. In device A, input "a2dp.configure" to configure the a2dp connection.
6. In device A, input "a2dp.start" to start the a2dp media.
7. In device A, input "a2dp.send\_media x" to send media data for x seconds.
8. For other commands:
  - i. "a2dp.disconnect" is used to disconnect the a2dp.
  - ii. "a2dp.discover\_peer\_eps" is used to discover peer device's endpoints.
  - iii. "a2dp.get\_registered\_eps" is used to get the local registered endpoints.
  - iv. "a2dp.set\_default\_ep" is used to set the default selected endpoint.
  - v. "a2dp.deconfigure" de-configure the endpoint, then it can be configured again.
  - vi. "a2dp.stop" stops media.
  - vii. "a2dp.send\_delay\_report" send delay report.

### Running avrcp

The commands are as follows:

```
+---"avrcp": avrcp Bluetooth AVRCP shell commands
+---"init_ct": init_ct [none]
+---"init_tg": init_tg [none]
+---"ctl_connect": ctl_connect "create control connection"
+---"brow_connect": brow_connect "create browsing connection"
+---"ct_list_all_cases": ct_list_all_cases "display all the test cases"
+---"ct_test_case": ct_test_case <select one case to test>
+---"ct_test_all": ct_test_all "test all cases"
+---"ct_reg_ntf": ct_reg_ntf <Register Notification. select event:
    1. EVENT_PLAYBACK_STATUS_CHANGED
    2. EVENT_TRACK_CHANGED
    3. EVENT_TRACK_REACHED_END
    4. EVENT_TRACK_REACHED_START
    5. EVENT_PLAYBACK_POS_CHANGED
    6. EVENT_BATT_STATUS_CHANGED
    7. EVENT_SYSTEM_STATUS_CHANGED
    8. EVENT_PLAYER_APPLICATION_SETTING_CHANGED
    9. EVENT_NOW_PLAYING_CONTENT_CHANGED
    a. EVENT_AVAILABLE_PLAYERS_CHANGED
    b. EVENT_ADDRESSED_PLAYER_CHANGED
    c. EVENT_UIDS_CHANGED
    d. EVENT_VOLUME_CHANGED>
+---"tg_notify": tg_notify <Notify event. select event:
    1. EVENT_PLAYBACK_STATUS_CHANGED
    2. EVENT_TRACK_CHANGED
    3. EVENT_TRACK_REACHED_END
    4. EVENT_TRACK_REACHED_START
    5. EVENT_PLAYBACK_POS_CHANGED
    6. EVENT_BATT_STATUS_CHANGED
    7. EVENT_SYSTEM_STATUS_CHANGED
    8. EVENT_PLAYER_APPLICATION_SETTING_CHANGED
    9. EVENT_NOW_PLAYING_CONTENT_CHANGED
    a. EVENT_AVAILABLE_PLAYERS_CHANGED
    b. EVENT_ADDRESSED_PLAYER_CHANGED
    c. EVENT_UIDS_CHANGED
    d. EVENT_VOLUME_CHANGED>
+---"ca_init_i": ca_init_i "Init cover art initiator"
+---"ca_init_r": ca_init_r "Init cover art responder"
+---"ca_connect": ca_connect "create cover art connection"
+---"ca_test": ca_test "cover art test all cases"
```

Test flow:

1. Create ACL connection between two devices (A and B).
2. In device B, input "avrcp.init\_tg" to initialize Target.
3. In device A, input "avrcp.init\_ct" to initialize Controller.
4. In device B, input "avrcp.ca\_init\_r" to initialize Cover Art responder.
5. In device A, input "avrcp.ca\_init\_i" to initialize Cover Art Initiator.
6. In device A, input "avrcp.ctl\_connect" to create AVRCP Control connection.
7. In device A, input "avrcp.brow\_connect" to create AVRCP Browsing connection.
8. In device A, input "avrcp.ct\_test\_all" to test all the cases.
9. In device A, input "avrcp.ct\_reg\_ntf" to register notification.
10. In device B, input "avrcp.tg\_notify" to notify.
11. In device A, input "avrcp.ca\_test" to test all the cover art commands.

12. For other commands:
  - i. In device A, input "avrccp.ct\_list\_all\_cases" to list all the test cases.
  - ii. In device A, input "avrccp.ct\_test\_case x" to test one selected case.

### Running BR/EDR L2CAP

1. Create ACL connection between two devices (A and B).
2. In device A and B, input "br.l2cap-register <psm>" to register one psm (for example: br.l2cap-register 1001).
3. In device A, input "br.l2cap-connect <psm>" to create l2cap connection (for example: br.l2cap-connect 1001).
4. In device A, input "br.l2cap-send x" to send data.
5. In device A, input "br.l2cap-disconnect" to disconnect the l2cap connection.
6. In device A and B, input "br.l2cap-register-mode <psm>" to register one psm (for example: br.l2cap-register 1003).
7. In device A, input "br.l2cap-connect <psm>" to create l2cap connection (for example: br.l2cap-connect 1003).
8. In device A, input "br.l2cap-send x" to send data.
9. In device A, input "br.l2cap-disconnect" to disconnect the l2cap connection.

Note: The shell information "SHELL build: Aug 10 2021" may be different, which depends on the compile date.

## 5.16 audio\_profile Sample Application

There are five parts working in the demo: AWS cloud, Android app, audio demo (running on i.MX RT1060 EVK board), U-disk and Bluetooth headset.

- With an app running on the smart phone (Android phone), the end users could connect to AWS cloud and control the audio demo running on the i.MX RT1060 EVK board through AWS cloud. Some operations like play, play next, pause, etc., could be used to control the media play functionalities.
- Audio demo running on the RT1060 EVK board connects to the AWS through Wi-Fi, also a connection could be established between the i.MX RT1060 EVK board and a Bluetooth headset.
- To get the media resource (mp3 files) from the U-disk, an HS USB host is enabled, and a U-disk with mp3 files should be connected to i.MX RT1060 EVK board via the USB port.
- After that, the audio demo will search the root directory of U-disk for the audio files and upload the audio file list to AWS, then the list would be shown in the app running on the smart phone.
- Finally, the music could be played out via the Bluetooth headset once end user controls the app to play the mp3 file.

**NOTE:**

- This demo could NOT function with the default setting provided in SDK package because a AWS account is mandatory to run to the demo, the end users must create their owner AWS account and configure the IoT Thing before the functionality of the demo could be used. Also some information specified by the end customers, like Thing name, Wi-Fi SSID, Wi-Fi password, etc., must be updated accordingly before the demo would work. Please follow the steps mentioned here to configure the same.
- The music files names in U-disk need to be English.
- The volume of audio adjustment is not supported.

### 5.16.1 User Configurations

Some of the AWS Client Credentials related macros that user need to configure based on requirement are listed in below table along with source file name.

The `aws_clientcredential.h` file is available in the SDK source, path is given in section 1.3 “References”.

Table 24: audio\_profile Application Configurations

Feature	Macro definition	Value set for Example	File name	Details
AWS Client Credentials	clientcredentialMQTT_BR_OKER_ENDPOINT	“a2qkq65ssjggf7-ats.iot.us-east-1.amazonaws.com”	aws_clientcredential.h	These credentials are required to connect the correct end point of AWS IoT Thing.
	clientcredentialIOT_THIN_G_NAME	“MusicPlayer”		
	clientcredentialWIFI_SSID	“NXP_Demo”		
	clientcredentialWIFI_PASWORD	“123456789”		
	clientcredentialMQTT_BR_OKER_PORT	“8883”		

### 5.16.2 audio\_profile Application Execution

Please refer to the previous section 3.1.3 to run the demo using MCUXpresso IDE. Refer below figures for importing Bluetooth example and selection of Bluetooth module.

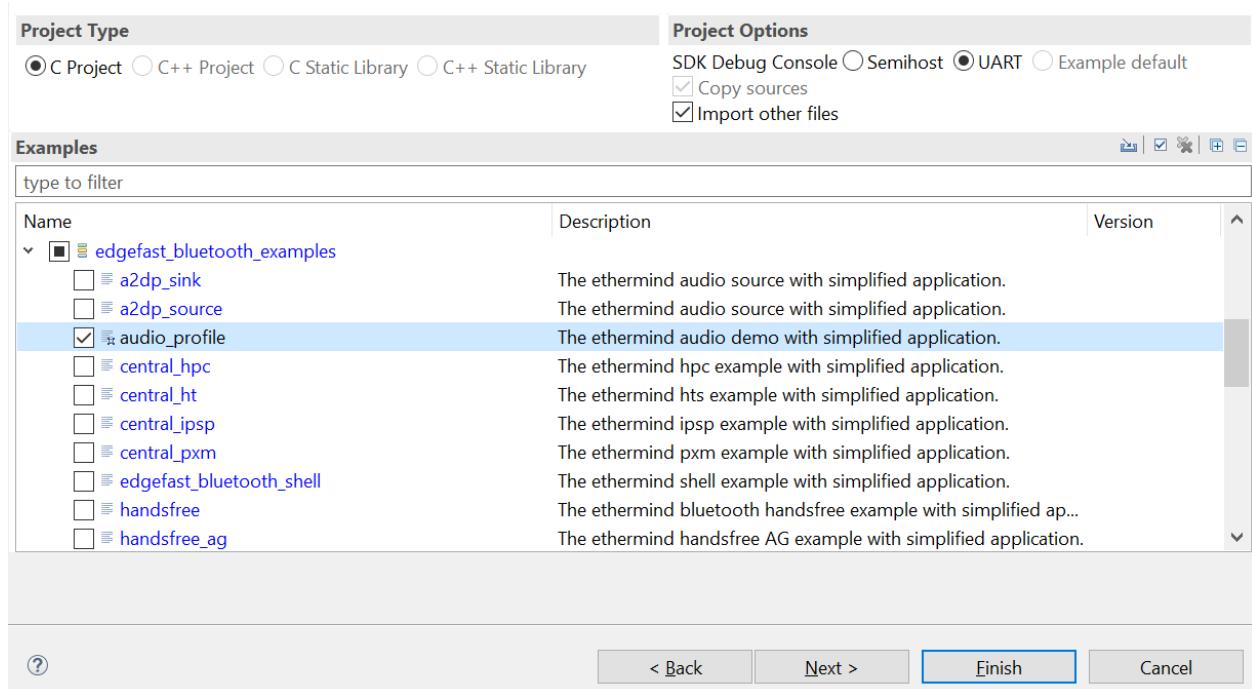


Figure 41 : Selection of audio\_profile application in MCUXpresso IDE

Please refer to the previous sections 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console setup.

#### 5.16.2.1 Create and configure AWS Account

Follow the link to create a new AWS account:

<https://console.aws.amazon.com/console/home>

Follow the instructions to create a new AWS account:

<https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/>

#### 5.16.2.2 Create an AWS IoT Policy

This section describes the steps to create a policy for AWS IoT.

Browse to the AWS IoT console

<https://console.aws.amazon.com/iotv2/>

Click "Policies" inside the "Secure" tab:

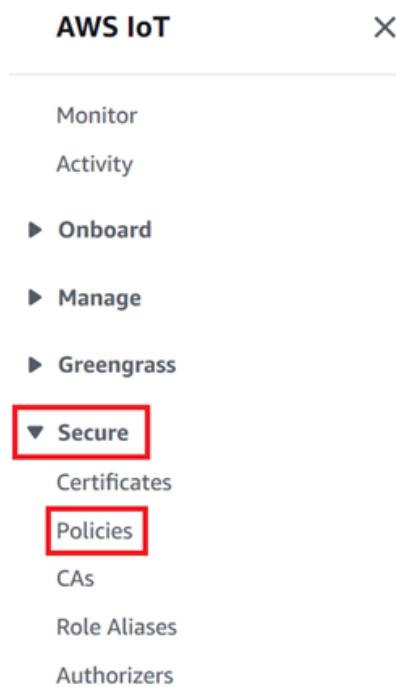


Figure 42: Selection of Policies from AWS IoT tab

Create a new policy:

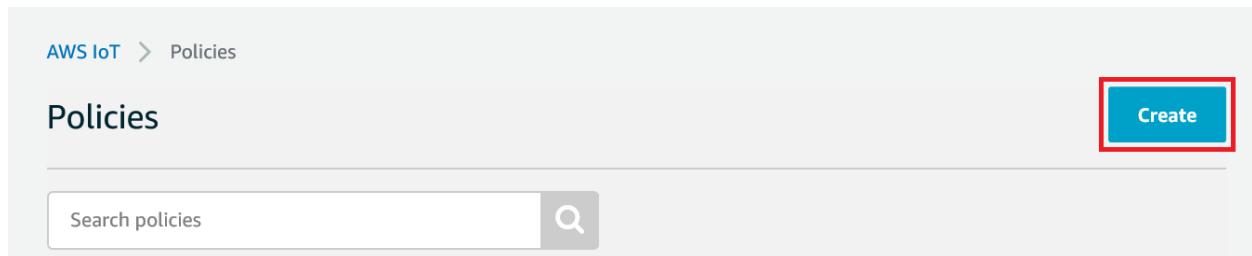
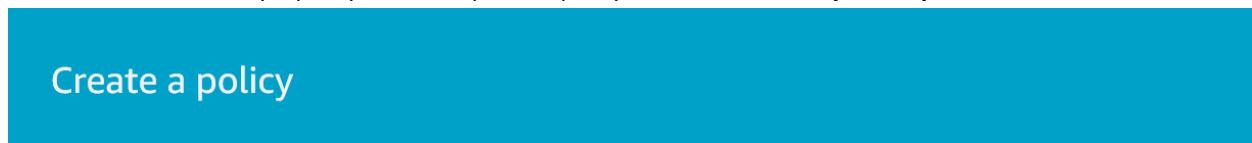


Figure 43: Creating a new policy

Enter a name to identify a policy. For example, the policy name is "**MusicPlayerPolicy**".



Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the [AWS IoT Policies documentation page](#).

A form input field with the label 'Name' above it. The input field contains the text 'MusicPlayerPolicy' and has a red border around it. A small 'X' icon is located to the right of the input field.

Figure 44: Policy Name

In the Add statements section, click “Advanced mode”.

### Add statements

Policy statements define the types of actions that can be performed by a resource.

**Advanced mode**

The screenshot shows a user interface for defining policy statements. At the top right, a button labeled "Advanced mode" is highlighted with a red border. Below it, there are three input fields: "Action" (with placeholder text "Please use commas to separate actions. e.g. iot:Publish, iot:Subscribe"), "Resource ARN" (with placeholder text "Specific resources could include client ID ARN, topic ARN, or topic filter ARN."), and "Effect" (with checkboxes for "Allow" and "Deny"). In the bottom right corner of the main form area, there is a "Remove" button. At the bottom left, there is a "Add statement" button.

Figure 45: Opening the Advanced mode

Add the JSON into the Policy editor window and create a policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iot:*",  
            "Resource": "*"  
        }  
    ]  
}
```

### Add statements

Policy statements define the types of actions that can be performed by a resource.

**Basic mode**



Figure 46: Adding the required JSON into the policy editor window

Upon successful creation of Policy following screen will appear:

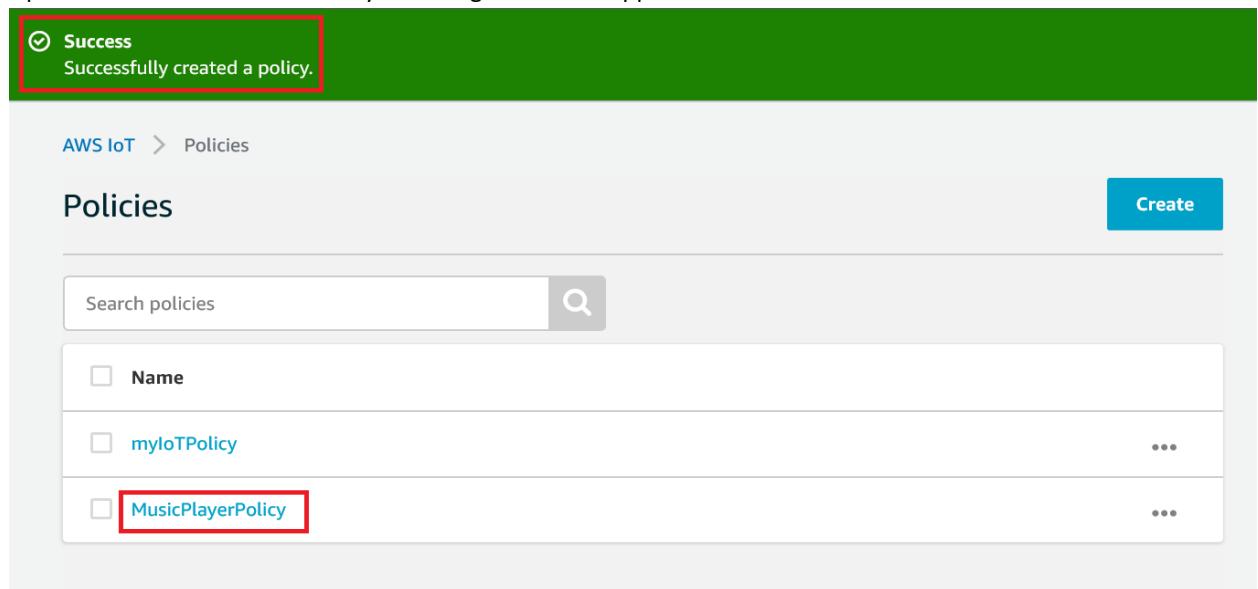


Figure 47: Showing the success of policy creation

### 5.16.2.3 Create IoT thing, private key, and certificate for device

Open the "AWS IoT console"

<https://console.aws.amazon.com/iot/>

From the navigation pane, click "Things" inside "Manage" tab.

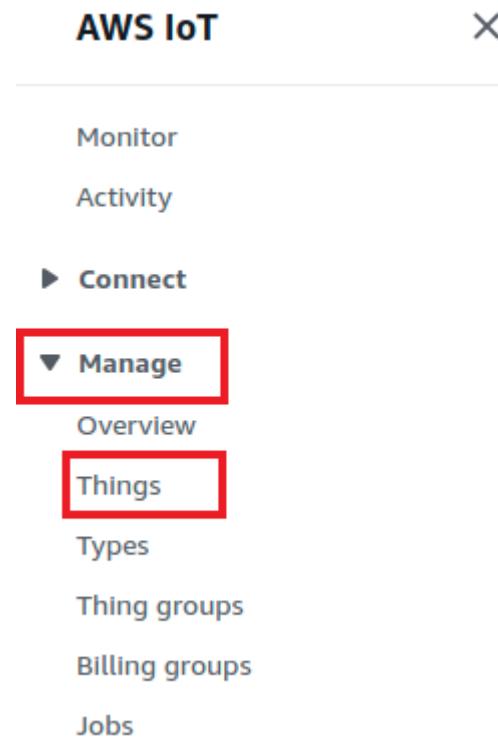


Figure 48: Selection of Things from AWS IoT tab

Click on "Create".

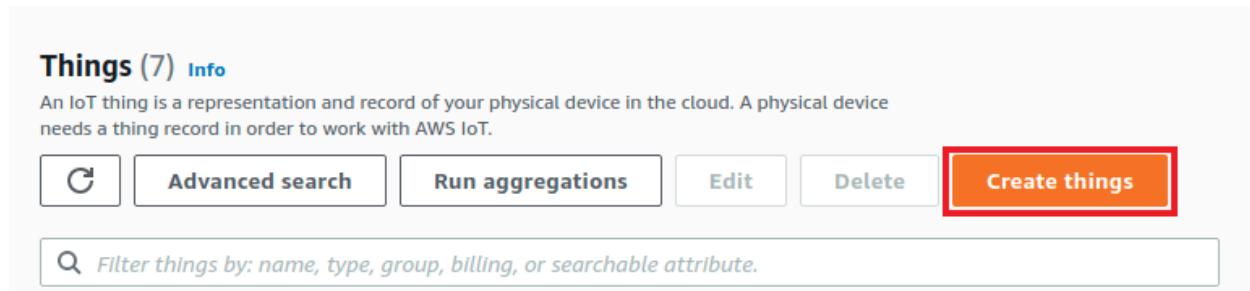


Figure 49: Creating a new Thing

Click "Create a single thing"

AWS IoT > Manage > Things > Create things

## Create things Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

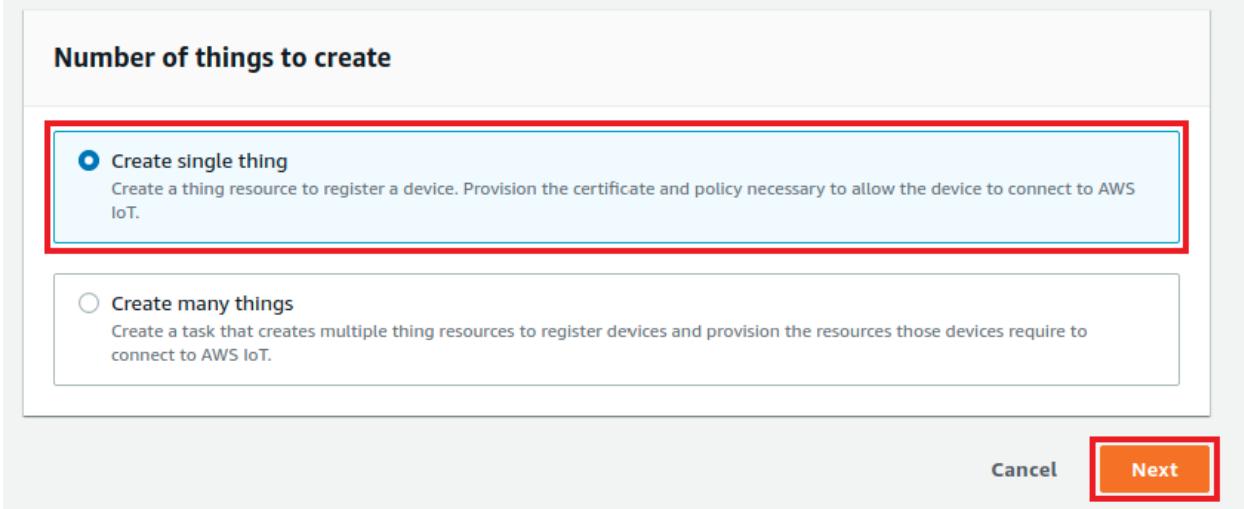


Figure 50: Creating a new Thing

Enter a name for your device, and then choose "Next". For example, the thing name is "MusicPlayer".

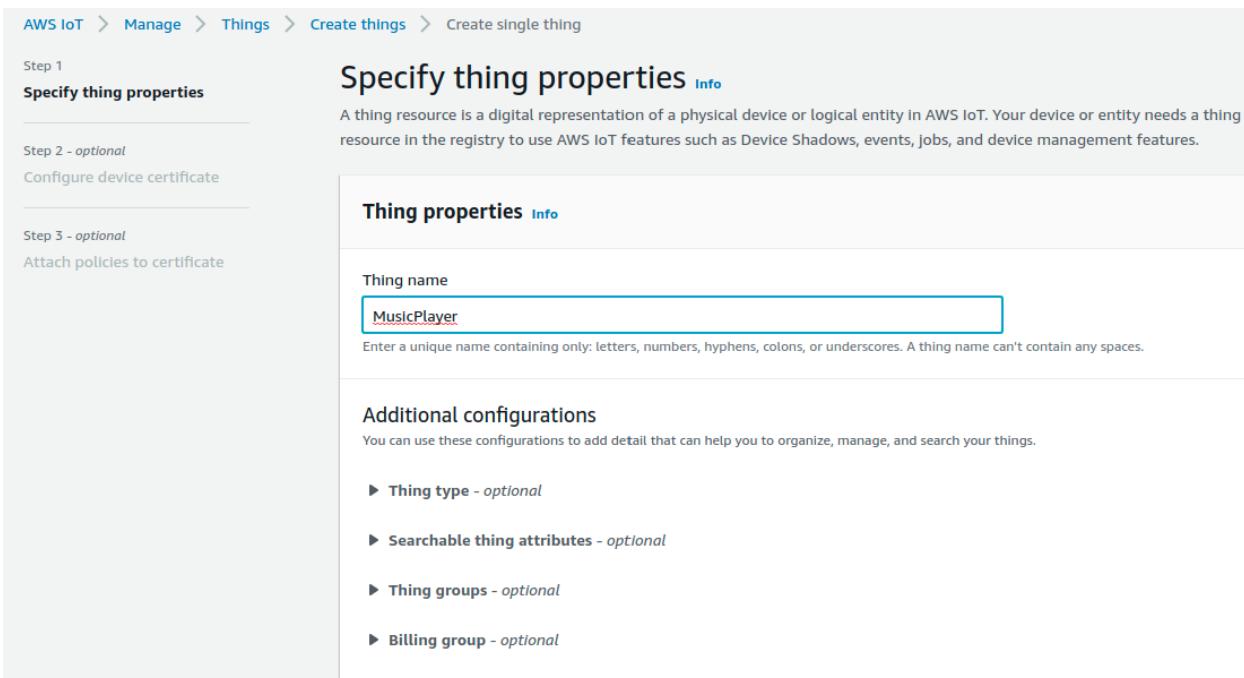


Figure 51: Giving name to a new thing

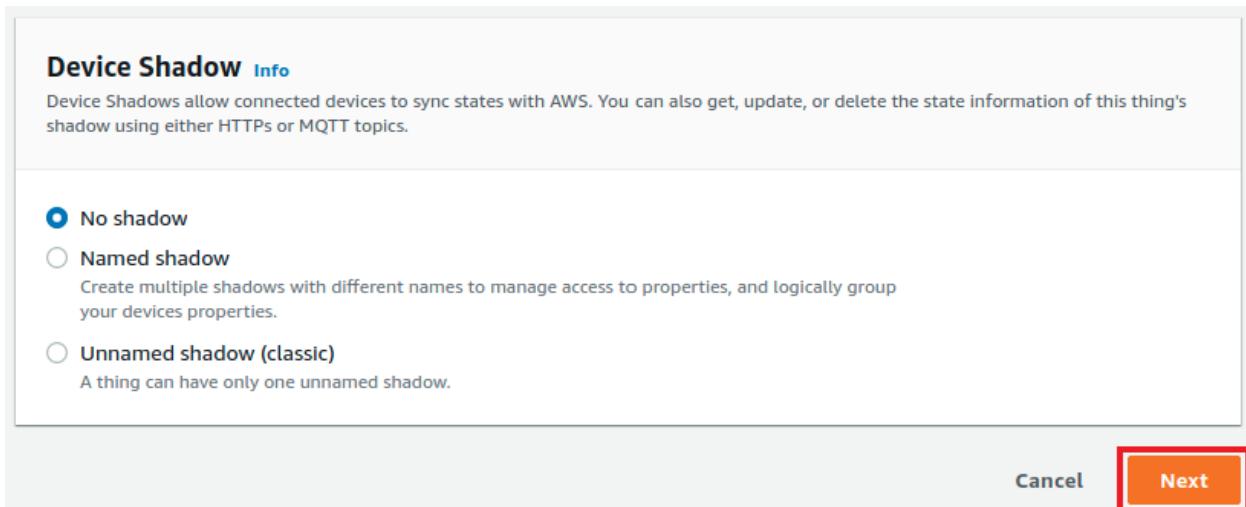


Figure 52: Click next to proceed for creating a new thing

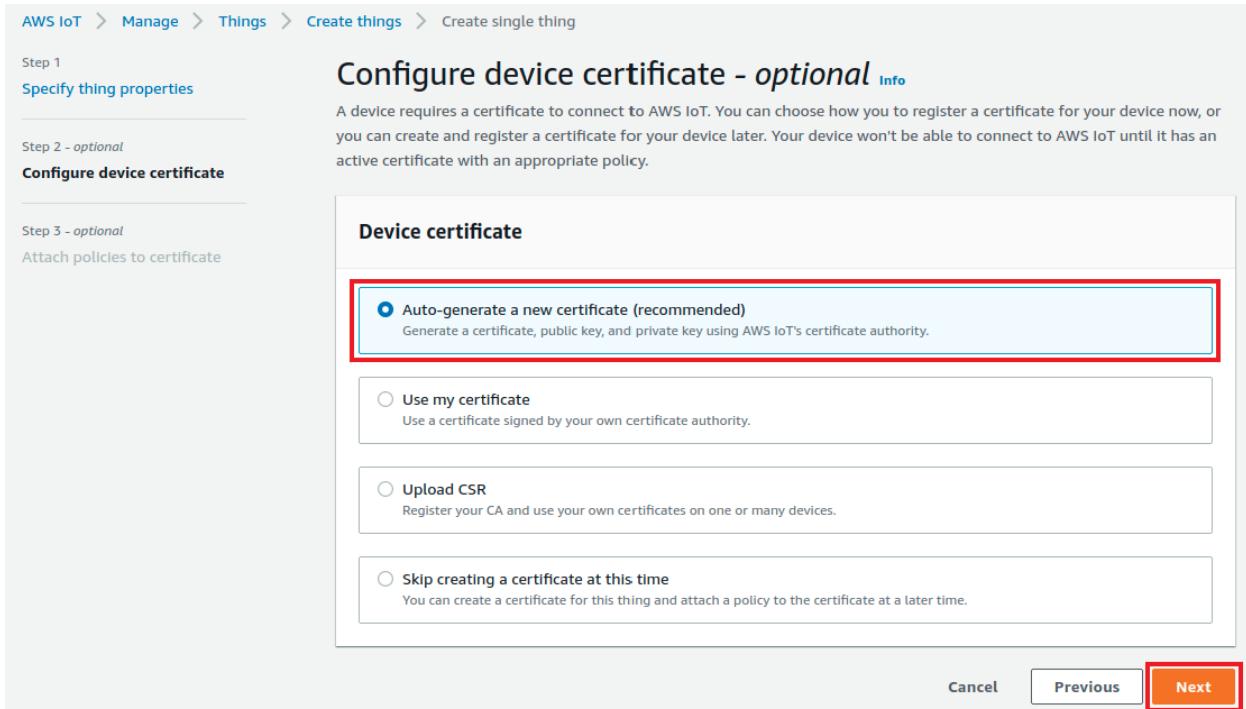
**Click "Create certificate"**

Figure 53: Selecting Device Certificate configuration for a new Thing

Select a policy to attach to your certificate that grants your device access to AWS IoT operations and click "Create Thing".

Step 1  
Specify thing properties

Step 2 - optional  
Configure device certificate

Step 3 - optional  
Attach policies to certificate

**Attach policies to certificate - optional** Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

**Policies (1/7)**  
Select up to 10 policies to attach to this certificate.

Filter policies

Name
<input type="checkbox"/> test_aws_wifi_provisioning_policy
<input type="checkbox"/> myIoTPolicy
<input type="checkbox"/> aws_wifi_provisioning_policy
<input type="checkbox"/> TestMyMusic
<input type="checkbox"/> MyWifiPro
<input type="checkbox"/> MyMusic
<input checked="" type="checkbox"/> MusicPlayerPolicy

Cancel Previous Create thing

Figure 54: Attach a policy and create a Thing

Download the thing's certificate, public key, and private key.

## Download certificates and keys



Download certificate and key files to install on your device so that it can connect to AWS.

### Device certificate

You can activate the certificate now, or later. The certificate must be active for a device to connect to AWS IoT.

Device certificate

5ff36500294...te.pem.crt

[Deactivate certificate](#)

[Download](#)

### Key files

The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.



This is the only time you can download the key files for this certificate.

Public key file

5ff36500294bace6e709df4...f13bfff-public.pem.key

[Download](#)

Private key file

5ff36500294bace6e709df4...13bfff-private.pem.key

[Download](#)

Figure 55: Downloading the certificate, public and private keys

Click the thing that you just created from the list.

The screenshot shows the AWS IoT Things interface. At the top, there's a breadcrumb navigation: AWS IoT > Things. On the right, there's a blue 'Create' button. Below the navigation is a search bar with placeholder text 'Search things' and a magnifying glass icon. To the right of the search bar are two buttons: 'Fleet Indexing' and 'Info'. The main area displays a table of registered things:

<input type="checkbox"/>	Name	Type	...
<input type="checkbox"/>	myNXThing	NO TYPE	...
<input type="checkbox"/>	MusicPlayer	NO TYPE	...

Figure 56: Selecting the policy and Register Thing

Click "Interact" from your thing's page and open "View Settings".

The screenshot shows the 'MusicPlayer' thing details page. At the top, there are 'Edit' and 'Delete' buttons. Below that is a 'Thing details' section with fields for Name (MusicPlayer), Type (-), ARN (arn:aws:iot:us-east-1:533155200463:thing/MusicPlayer), and Billing group (-). Below this is a navigation bar with tabs: Attributes, Certificates, Thing groups, Device Shadows, Interact (highlighted with a red box), Activity, Jobs, Alarms, and Defender metrics.

Under the 'Device Shadows' tab, there are two informational boxes:

- The device data endpoint has moved to Settings.** Your device data endpoint can be found under **Settings**. HTTP prefixes for Device Shadow interactions that use this endpoint can be found on the **Device Shadows** tab. **View Settings** button (highlighted with a red box).
- MQTT and HTTP prefixes have moved to Device Shadows.** MQTT topic prefixes and HTTP URLs are created for each Device Shadow and you can find them in the **Device Shadows** tab. **View Device Shadows tab** button.

Figure 57: Selecting Interact and opening View Settings to get Endpoint

Make a note of the AWS IoT REST API endpoint to use it for next sections.

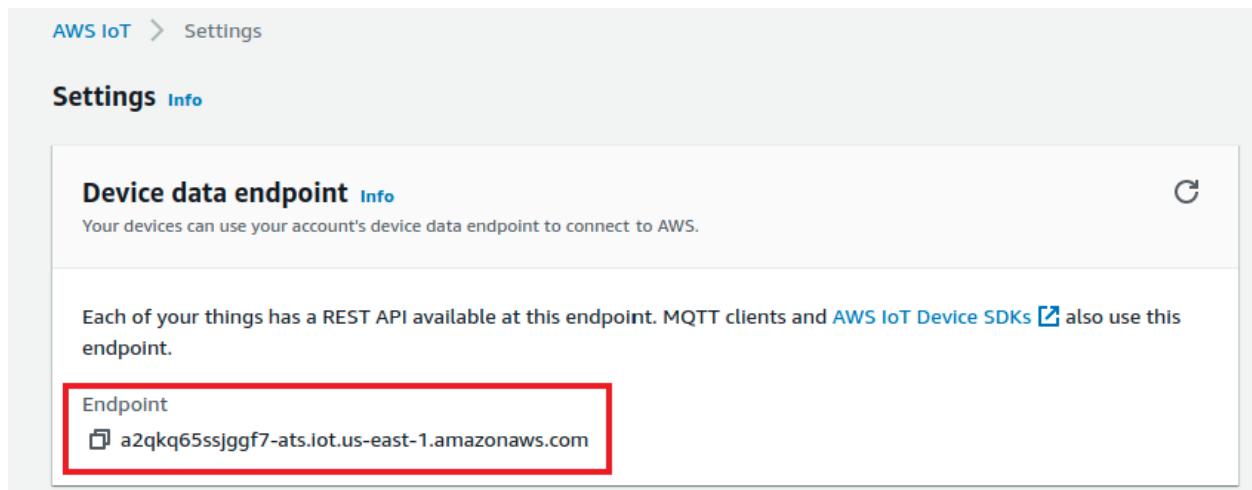


Figure 58: Copy the AWS IoT REST API endpoint

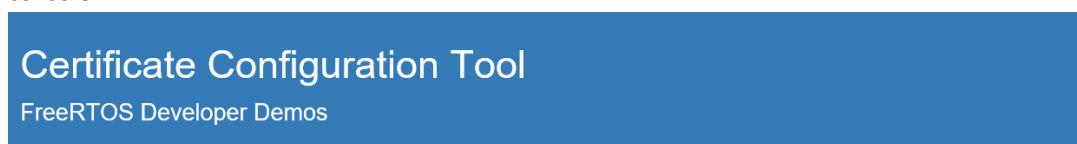
#### 5.16.2.4 Configure the AWS IoT Certificate and Private Keys

FreeRTOS needs the AWS IoT certificate and private keys associated with your registered thing and its permissions policies to successfully communicate with AWS IoT on behalf of your device.

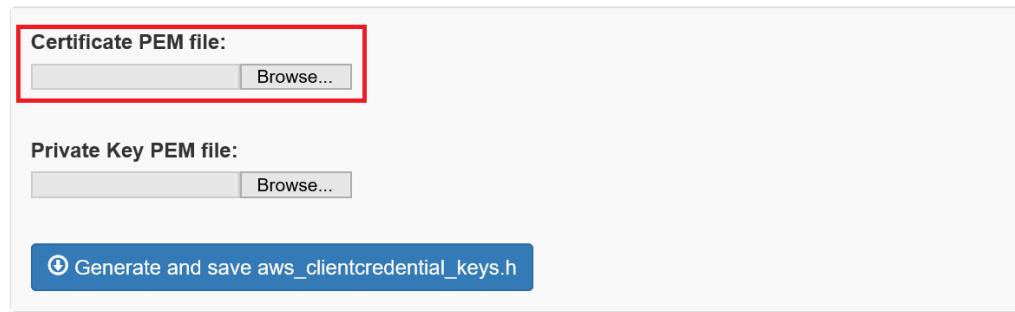
FreeRTOS is a C language project, and the certificate and private key must be specially formatted to be added to the project.

In a browser window, open *CertificateConfigurator.html* file referenced in section 1.3 “References”.

Under “**Certificate PEM file**”, choose the ID-certificate.pem.crt that you downloaded from the AWS IoT console.



Provide client certificate and private key PEM files downloaded from the AWS IoT Console.



**⚠ Save the generated header file to the *demos/common/include* folder of the demo project.**

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Figure 59: Selecting the certification

Under “**Private Key PEM file**”, choose the ID-private.pem.key that you downloaded from the AWS IoT console.

## Certificate Configuration Tool

FreeRTOS Developer Demos

Provide client certificate and private key PEM files downloaded from the AWS IoT Console.

**Certificate PEM file:**

**Private Key PEM file:**

 **Generate and save aws\_clientcredential\_keys.h**

 Save the generated header file to the `demos/common/include` folder of the demo project.

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Figure 60: Selecting the Private key

Choose "**Generate and save aws\_clientcredential\_keys.h**", and then save the file in `SDK_<PATH>/rtos/freertos/demos/include`. This overwrites the existing file in the directory.

**NOTE:** The certificate and private key are hard-coded for demonstration purposes only. Production-level applications should store these files in a secure location.

## Certificate Configuration Tool

FreeRTOS Developer Demos

Provide client certificate and private key PEM files downloaded from the AWS IoT Console.

**Certificate PEM file:**

5ff36500294...icate.pem.crt

**Private Key PEM file:**

5ff36500294...vate.pem.key

 **Generate and save aws\_clientcredential\_keys.h**

 Save the generated header file to the `demos/include` folder of the demo project.

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Figure 61: Generate and save aws\_clientcredential\_keys.h

### 5.16.2.5 Configure the AWS IoT endpoint

User need to update FreeRTOS with your AWS IoT endpoint so the application running on the board can send requests to the correct endpoint.

Open “*aws\_clientcredential.h*” file.

Set the “clientcredentialMQTT\_BROKER\_ENDPOINT” as per the Rest API Endpoint.

```
#define clientcredentialMQTT_BROKER_ENDPOINT "a2gkq65ssjggf7-  
ats.iot.us-east-1.amazonaws.com"
```

Set the “clientcredentialIOT\_THING\_NAME” as per the name of IoT Thing

```
#define clientcredentialIOT_THING_NAME "MusicPlayer"
```

Set the “clientcredentialWIFI\_SSID” as the connected Wi-Fi SSID

```
#define clientcredentialWIFI_SSID "NXP_Demo"
```

Set the “clientcredentialWIFI\_PASSWORD” as the connected Wi-Fi Password.

```
#define clientcredentialWIFI_PASSWORD "123456789"
```

Set the “clientcredentialMQTT\_BROKER\_PORT” as 443

```
#define clientcredentialMQTT_BROKER_PORT 8883
```

Rebuild the application and flash it on the target board.

Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.

Prepare the Android application

Open the Amazon Cognito console,  
<https://console.aws.amazon.com/cognito/home>  
Choose "Manage Identity Pools"

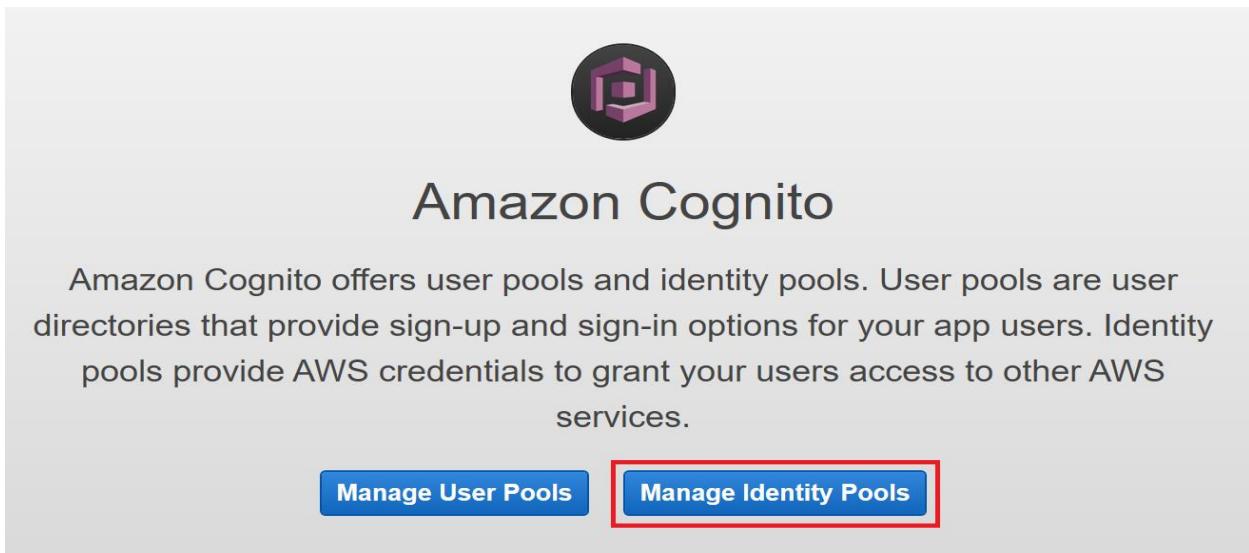


Figure 62: Manage Identity Pools

Click "Create new identity pool".

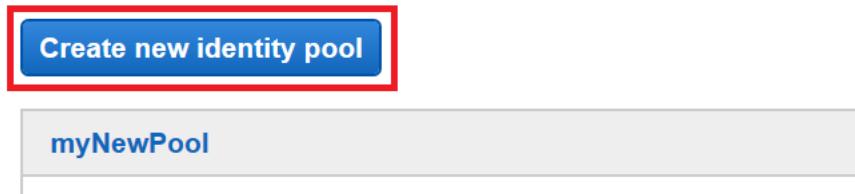


Figure 63: Create new identity pool

Enter a name for your identity pool. Such as the pool name is "MusicPlayerIdentity", enable unauthenticated access.

### Create new identity pool

Identity pools are used to store end user identities. To declare a new identity pool, enter a unique name.

Identity pool name\*    
Example: My App Name

#### ▼ Unauthenticated identities

Amazon Cognito can support unauthenticated identities by providing a unique identifier and AWS credentials for users who do not authenticate with an identity provider. If your application allows customers to use the application without logging in, you can enable access for unauthenticated identities. [Learn more about unauthenticated identities.](#)

Enable access to unauthenticated identities

Enabling this option means that anyone with internet access can be granted AWS credentials. Unauthenticated identities are typically users who do not log in to your application. Typically, the permissions that you assign for unauthenticated identities should be more restrictive than those for authenticated identities.

Figure 64: Identity pool name

Click "Create Pool".

▼ Authentication flow settings ⓘ

A user authenticating with Amazon Cognito will go through a multi-step process to bootstrap their credentials. Amazon Cognito has two different flows for authentication with public providers: enhanced and basic. Cognito recommends the use of enhanced authentication flow. However, if you still wish to use the basic flow, you can enable it here. [Learn more about authentication flows.](#)

Allow Basic (Classic) Flow

▶ Authentication providers ⓘ

\* Required

Cancel

Create Pool

Figure 65: Create pool

Click "Allow" to create a pool

▶ View Details

Cancel

Allow

Figure 66: Allow to create a pool

Click "Services"

The screenshot shows the AWS Services menu. On the left, there's a sidebar with links: Identity pool, Dashboard, Sample code (which is selected and highlighted with an orange border), and Identity browser. On the right, the main content area is titled "Getting started with Amazon Cognito". It shows a "Platform" dropdown set to "Android" and a "Download the AWS SDK" section with a blue "Download the AWS SDK for Android" button and a "Developer Guide" link. The "Services" button in the top navigation bar is also highlighted with a red box.

Figure 67: Open services menu

Click “IAM” inside “All Services”

The screenshot shows the AWS Services navigation bar at the top. Below it, the "All services" section is displayed. The "IAM" service is located in the "Security, Identity, & Compliance" category, which is also highlighted with a red box.

Category	Service	Category	Service
Networking & Content Delivery	VPC	Media Services	Kinesis Video Streams
CloudFront	Route 53	MediaConnect	MediaConvert
API Gateway	Direct Connect	MediaLive	MediaPackage
AWS App Mesh	AWS Cloud Map	MediaStore	MediaTailor
Global Accelerator	Elemental Appliances ...	Amazon Interactive Vid...	GuardDuty
Developer Tools	Elastic Transcoder	Inspector	Amazon Macie

Figure 68: Open IAM

Click “Roles” from “IAM dashboard”.

## IAM dashboard

### Security recommendations 1

#### ! Add MFA for root user

Enable multi-factor authentication (MFA) for the root user to improve security for this account.

[Add MFA](#)

#### ✓ Root user has no active access keys

Using access keys attached to an IAM user instead of the root user improves security.

### IAM resources

User groups	Users	Roles	Policies	Identity providers
0	0	19	2	0

Figure 69: Open available roles

Click “Cognito\_MusicPlayerIdentityUnauth\_Role”.

The screenshot shows the "Create role" and "Delete role" buttons at the top. Below is a search bar and a table with columns: Role name, Trusted entities, and Last activity. The "Cognito\_MusicPlayerIdentityUnauth\_Role" is highlighted with a red box and a tooltip showing its details: Identity Provider: cognito-identity.amazonaws.com and Last activity: Today.

Role name	Trusted entities	Last activity
Cognito_myNewPoolUnauth_Role	Identity Provider: cognito-identity.amazonaws.com	101 days
Cognito_myNewPoolAuth_Role	Identity Provider: cognito-identity.amazonaws.com	None
Cognito_MusicPlayerIdentityUnauth_Role	Identity Provider: cognito-identity.amazonaws.com	None
Cognito_MusicPlayerIdentityUnauth_Role	Identity Provider: cognito-identity.amazonaws.com	Today
Cognito_MusicPlayerIdentityAuth_Role	Identity Provider: cognito-identity.amazonaws.com	None

Figure 70: Selecting un-authentication role

Click the arrow as shown to edit the policy.

The screenshot shows the AWS IAM Policy Editor interface. At the top, there are tabs: Permissions (selected), Trust relationships, Tags, Access Advisor, and Revoke sessions. Below the tabs, it says "Permissions policies (1 policy applied)". There is a blue "Attach policies" button and a "Add inline policy" button. The main area shows a table with columns "Policy name" and "Policy type". One row is visible, labeled "oneClick\_Cognito\_MusicPlayerIdentityUnauth\_Role\_1607781" with "Inline policy" type. A red box highlights the "Edit" icon (a pencil icon) next to the policy name.

Figure 71: Open policy content

Click “Edit Policy”.

The screenshot shows the AWS IAM Policy Editor interface with the "Edit policy" button highlighted by a red box. Above the JSON editor, there are tabs: Policy summary, JSON (disabled), and Edit policy. To the right is a "Simulate policy" button. The JSON code in the editor is:

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Effect": "Allow",  
6       "Action": [  
7         "mobileanalytics:PutEvents",  
8         "cognito-sync:*"  
9       ],  
10      "Resource": [  
11        "*"  
12      ]  
13    }  
14  ]  
15 }
```

Figure 72: Edit the policy

Click “JSON”

Edit oneClick\_Cognito\_MusicPlayerIdentityUnauth\_Role\_1607781324848

1 2

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

The screenshot shows the AWS IAM Policy Editor interface. At the top, there are tabs for "Visual editor" and "JSON", with "JSON" being the active tab and highlighted with a red box. Below the tabs, there are buttons for "Import managed policy", "Clone", and "Remove". Under the tabs, there are sections for "Mobile Analytics (1 action)" and "Cognito Sync (All actions)". At the bottom right, there are buttons for "Cancel" and "Review policy". A note at the bottom left states: "Character count: 130 of 10,240. The current character count includes character for all inline policies in the role: Cognito\_MusicPlayerIdentityUnauthentication\_Role." A link to "Learn more" is also present.

Figure 73: Open JSON tab

Fill the below content to the policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

**Click “Review Policy”**

Edit oneClick\_Cognito\_MusicPlayerIdentityUnauth\_Role\_1607781324848

1 2

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

```

1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "iot:Connect"
8             ],
9             "Resource": [
10                 "*"
11             ]
12         }
13     ]
14 }
  
```

Character count: 237 of 10,240.  
The current character count includes character for all inline policies in the role: Cognito\_MusicPlayerIdentityUnauthentication\_Role.

Cancel

Review policy

Figure 74: Review policy

**Click “Save changes”**

Edit oneClick\_Cognito\_MusicPlayerIdentityUnauth\_Role\_1607781324848

1 2

**Review policy**

Review this policy before you save your changes.

Summary			
<input type="text"/> Filter			
Service	Access level	Resource	Request condition
Allow (1 of 258 services) <a href="#">Show remaining 257</a>			
IoT	Limited: Write	All resources	None

\* Required

Cancel

Previous

Save changes

Figure 75: Save changes for the role selected

The pool is created successfully.

Click “Trust relationships”

The screenshot shows the AWS IAM Trust relationships tab. At the top, there are tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The Trust relationships tab is selected and highlighted with a red border. Below the tabs, a section titled "Permissions policies (1 policy applied)" is shown. A blue button labeled "Attach policies" is on the left, and a blue button labeled "+ Add inline policy" is on the right. A table below lists the attached policy, showing "Policy name" as "oneClick\_Cognito\_MusicPlayerIdentityUnauth\_Role\_1607781", "Policy type" as "Inline policy", and a delete icon on the far right.

Figure 76: Open Trust relationships tab

Make a copy of the Identity pool ID to use for next section.

The screenshot shows the "Edit trust relationship" page. At the top, there are tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The Trust relationships tab is selected and highlighted with a red border. Below the tabs, a message says "You can view the trusted entities that can assume the role and the access conditions for the role." A blue button labeled "Edit trust relationship" is present. The page is divided into two main sections: "Trusted entities" and "Conditions".

**Trusted entities:** The following trusted entities can assume this role: cognito-identity.amazonaws.com

**Conditions:**

Condition	Key	Value
StringEquals	cognito-identity.amazonaws.com:aud	us-east-1:408bb2c1-0728-4af9-97ce-c0f13c268a21
ForAnyValue:StringLike	cognito-identity.amazonaws.com:amr	unauthenticated

Figure 77: copy Identity pool ID

### 5.16.2.6 Prepare Configuration File for the Android Application

Prepare "AwsMusicControlPreferences.properties" file with yours AWS credentials.

It's structure looks like this:

```
customer_specific_endpoint=<REST API ENDPOINT>
cognito_pool_id=<COGNITO POOL ID>
thing_name=<THING NAME>
region=<REGION>
```

Where:

**customer\_specific\_endpoint** is the endpoint that is configured in *aws\_clientcredential.h*

**cognito\_pool\_id** is the copied pool id in above step.

**thing\_name** is the created Thing name.

**region** is the front part of the cognito pool id.

For Example:

```
customer_specific_endpoint=a2qkq65ssjggf7-ats.iot.us-east-1.amazonaws.com
cognito_pool_id=us-east-1:408bb2c1-0728-4af9-97ce-c0f13c268a21
thing_name=MusicPlayer
```

```
region=us-east-1
```

To run Android application,

Install and run pre-build *AwsRemoteControl.apk* on Android device, file path is referenced in section 1.3 "References".

Then in both cases when asked select *AwsMusicControlPreferences.properties* file with AWS IoT preferences. Then control the music.

**NOTE:** Application requires at least Android version 5.1 (Android SDK 22).

#### 5.16.2.7 Run the application

The log below shows the output of the demo in the console window. The log can be different based on your Wi-Fi network configuration and based on the actions, which you have done in the Android application.

After the log "**Use mobile application to control the remote device.**", the shell command can be used to connect to Bluetooth headset.

```
usb host init done
mass storage device attached:pid=0x5bavid=0x54c address=1
usb msd device is ready
0 254 [main_task] Write certificate...
1 285 [main_task] Starting WiFi...
Available audio files:
 1.mp3
 2.mp3
 3.mp3
 4.mp3
MAC Address: 20:4E:F6:25:F3:19
[net] Initialized TCP/IP networking stack
[wm_wlan] WLAN_REASON_INITIALIZED

2 4296 [main_task] WiFi module initialized.
[wm_wlan] Connecting to NXP_Demo .....

[wm_wlan] Connected to with IP = [192.168.43.149]

3 24498 [main_task] WiFi connected to AP NXP_Demo.
4 24498 [main_task] IP Address acquired 192.168.43.149
5 24499 [AWS-RemoteCtrl] [INFO ][INIT][lu] SDK successfully initialized.
6 24499 [AWS-RemoteCtrl] [INFO ][Shadow][lu] Shadow library successfully
initialized.
Bluetooth initialized
7 40582 [AWS-RemoteCtrl] [INFO ][MQTT][lu] Establishing new MQTT connection.
8 40588 [AWS-RemoteCtrl] [INFO ][MQTT][lu] Anonymous metrics (SDK language, SDK
version) will be provided to AWS IoT. Recompile with
AWS_IOT_MQTT_ENABLE_METRICS set to 0 to disable.
9 40591 [AWS-RemoteCtrl] [INFO ][MQTT][lu] (MQTT connection 0x20230c48, CONNECT
operation 0x20231c10) Waiting for operation completion.
10 40897 [AWS-RemoteCtrl] [INFO ][MQTT][lu] (MQTT connection 0x20230c48,
CONNECT operation 0x20231c10) Wait complete with result SUCCESS.
11 40898 [AWS-RemoteCtrl] [INFO ][MQTT][lu] New MQTT connection 0x20228a08
established.
12 40899 [AWS-RemoteCtrl] [INFO ][MQTT][lu] (MQTT connection 0x20230c48)
SUBSCRIBE operation scheduled.
13 40899 [AWS-RemoteCtrl] [INFO ][MQTT][lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x20231c10) Waiting for operation completion.
```

```

14 43301 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x20231c10) Wait complete with result SUCCESS.
15 43302 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48)
SUBSCRIBE operation scheduled.
16 43302 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x20231c10) Waiting for operation completion.
17 43566 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x20231c10) Wait complete with result SUCCESS.
18 43568 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48) MQTT
PUBLISH operation queued.
19 43838 [iot_thread] [INFO ] [Shadow] [lu] Shadow DELETE of MusicPlayer was
ACCEPTED.
20 43839 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48)
UNSUBSCRIBE operation scheduled.
21 43839 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
UNSUBSCRIBE operation 0x20231c10) Waiting for operation completion.
22 44384 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
UNSUBSCRIBE operation 0x20231c10) Wait complete with result SUCCESS.
23 44385 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48)
UNSUBSCRIBE operation scheduled.
24 44385 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
UNSUBSCRIBE operation 0x20231c10) Waiting for operation completion.
25 44658 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
UNSUBSCRIBE operation 0x20231c10) Wait complete with result SUCCESS.
26 44658 [AWS-RemoteCtrl] [INFO ] [Shadow] [lu] (MusicPlayer) Modifying Shadow
DELTA callback.
27 44659 [AWS-RemoteCtrl] [INFO ] [Shadow] [lu] (MusicPlayer) Adding new DELTA
callback.
28 44659 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48)
SUBSCRIBE operation scheduled.
29 44660 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x20230b60) Waiting for operation completion.
30 44923 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x20230b60) Wait complete with result SUCCESS.
31 44924 [AWS-RemoteCtrl] [INFO ] [Shadow] [lu] (MusicPlayer) Shadow DELTA
callback operation complete with result SUCCESS.
32 44924 [AWS-RemoteCtrl] [INFO ] [Shadow] [lu] (MusicPlayer) Modifying Shadow
UPDATED callback.
33 44924 [AWS-RemoteCtrl] [INFO ] [Shadow] [lu] (MusicPlayer) Shadow UPDATED
callback operation complete with result SUCCESS.
34 44926 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48)
SUBSCRIBE operation scheduled.
35 44926 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x202325c8) Waiting for operation completion.
36 45186 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x202325c8) Wait complete with result SUCCESS.
37 45186 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48)
SUBSCRIBE operation scheduled.
38 45187 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x202322b0) Waiting for operation completion.
39 45526 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48,
SUBSCRIBE operation 0x202322b0) Wait complete with result SUCCESS.
40 45529 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48) MQTT
PUBLISH operation queued.
41 47132 [iot_thread] [INFO ] [Shadow] [lu] Shadow UPDATE of MusicPlayer was
ACCEPTED.
42 47133 [AWS-RemoteCtrl] AWS Remote Control Demo initialized.
43 47133 [AWS-RemoteCtrl] Use mobile application to control the remote device.

```

SHELL build: Mar 12 2021  
 Copyright 2021 NXP

&gt;&gt;

Use command “**help**” to list the available options:

```
>> help

"help": List all the registered commands

"exit": Exit program

"bt": BT related function
    USAGE: bt [connectaddress|finddevice|connectdevice|disconnect|deletedevice]
        connectaddress connect to the device of the address parameter, for example:
        bt connectaddress xx:xx:xx:xx:xx:xx. Address format(LSB-MSB): xx:xx:xx:xx:xx:xx
        finddevice      start to find BT devices
        connectdevice   connect to the device that is found, for example: bt
        connectdevice n (from 1)
        disconnect     disconnect current connection
        deletedevice   delete all devices. Ensure to disconnect the HCI link
        connection with the peer device before attempting to delete the bonding
        information.
>>
```

Use command “**bt finddevice**” to scan nearby Bluetooth devices

```
>> bt finddevice
>> Discovery started. Please wait ...
BR/EDR discovery complete
[1]: 70:F0:87:C0:FC:0E, RSSI -65 iPhone
[2]: BC:17:B8:74:2C:9F, RSSI -52 Galaxy
[3]: 50:82:D5:78:31:DA, RSSI -78 iPhone 6
[4]: 00:00:AB:CD:87:D6, RSSI -38 Airdopes 441
[5]: 04:C8:07:25:29:73, RSSI -69 Mi A3
```

Use command “**bt connectdevice <number>**” to connect to remote Bluetooth device

Here, “number” value can be found from the logs of “**bt finddevice**” command used above.

```
>> bt connectdevice 4
>> Connection pending
SDP discovery started
Connected
sdp success callback
A2DP Service found. Connecting ...
Security changed: 7E:5E:2B:2E:9A:C3 (0xed) level 2
44 173173 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x2022fdb0) MQTT
PUBLISH operation queued.
45 173755 [iot_thread] [INFO ] [Shadow] [lu] Shadow UPDATE of MusicPlayer was
ACCEPTED.
46 173756 [AWS-RemoteCtrl] Successfully performed update.
```

Open the android app and load the *AwsRemoteControlPreferences.properties*, wait for connection to get complete.

Use the smartphone application to play the music.

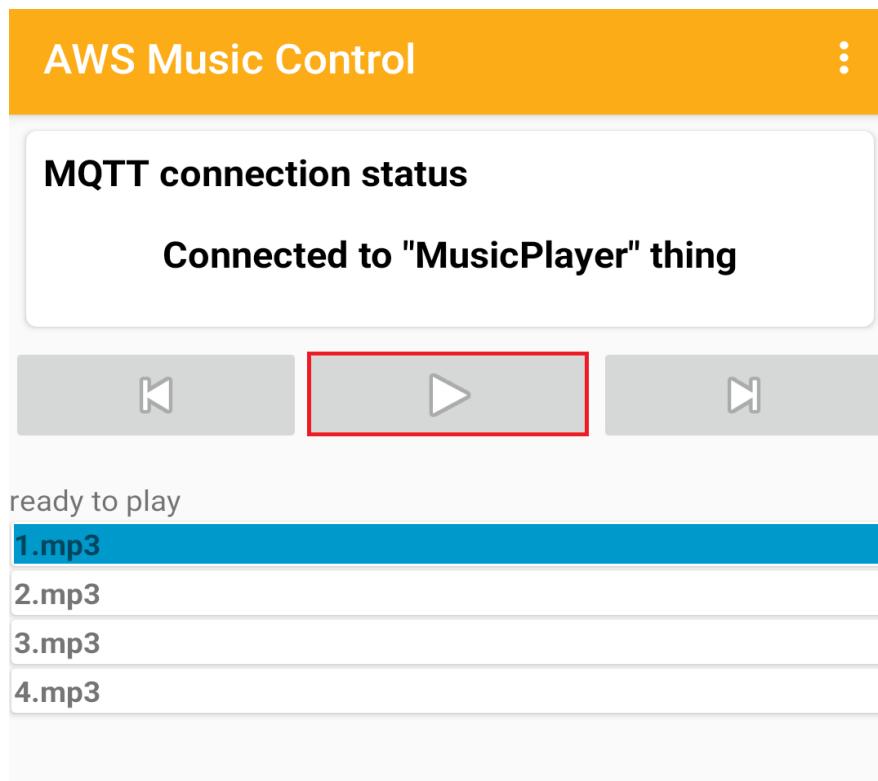


Figure 78: Play music using Android application

Following logs will be appear on the i.MX RT1060 EVK board console:

```
>> start play
[STREAMER] Message Task started
[STREAMER] start playback
Starting playback 0
47 118031 [AWS-RemoteCtrl] [INFO ] [MQTT] [lu] (MQTT connection 0x20230c48) MQTT
PUBLISH operation queued.
48 118045 [iot_thread] [WARN ] [Shadow] [lu] Received a Shadow UPDATE response
with no client token. This is possibly a response to a bad JSON document:
{"state":{"desired":{"playIndex":0,"playState":true}}, "metadata":{"desired":{"playIndex": {"timestamp":1649}}}}
48 118045 [iot_thread] [WARN ] [Shadow] [lu] Shadow
UPDATE callback received an unknown operation.
50 118695 [iot_thread] [INFO ] [Shadow] [lu] Shadow UPDATE of MusicPlayer was
ACCEPTED.
51 118695 [AWS-RemoteCtrl] Successfully performed update.
STREAM_MSG_UPDATE_POSITION
  position: 1005 ms
STREAM_MSG_UPDATE_POSITION
  position: 2011 ms
STREAM_MSG_UPDATE_POSITION
  position: 3004 ms
```

Use the smartphone application to pause the music.

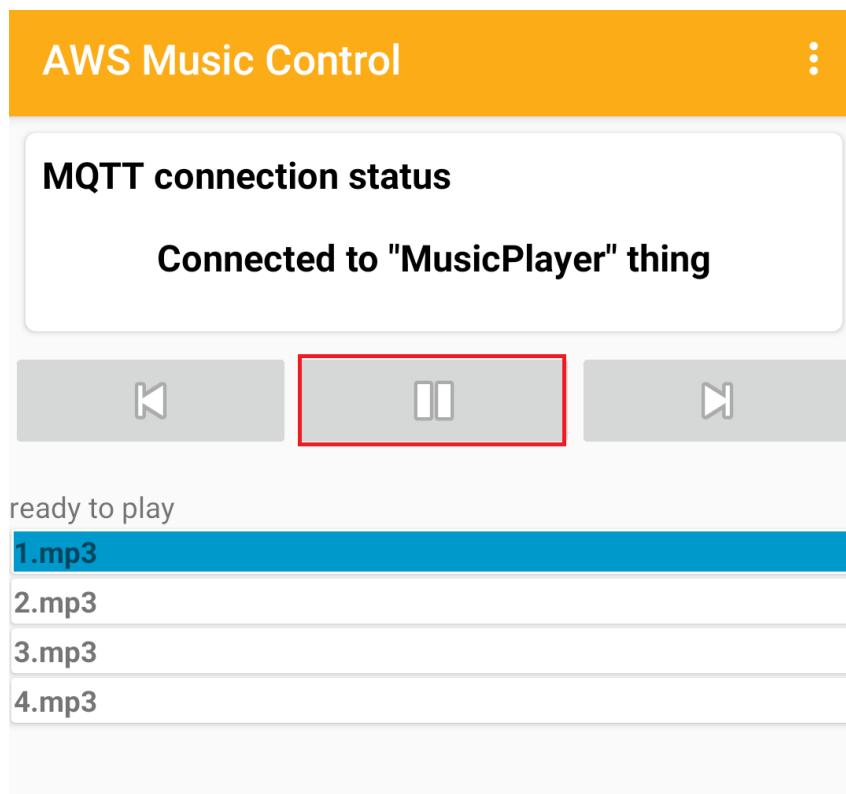


Figure 79: Pause music using Android application

Following logs will be appear on the i.MX RT1060 EVK board console:

```
stop play
52 214884 [AWS-RemoteCtrl] [INFO] [MQTT] [lu] (MQTT connection 0x2022fdb0) MQTT
PUBLISH operation queued.
53 214885 [iot_thread] [WARN] [Shadow] [lu] Received a Shadow UPDATE response
with no client token. This is possibly a response to a bad JSON document:
{"state":{"desired":{"playIndex":0,"playState":false}}, "metadata":{"desired":{"playIndex":{"timestamp":154 214885 [iot_thread] [WARN] [Shadow] [lu] Shadow
UPDATE callback received an unknown operation.
55 215504 [iot_thread] [INFO] [Shadow] [lu] Shadow UPDATE of MusicPlayer was
ACCEPTED.
56 215504 [AWS-RemoteCtrl] Successfully performed update.
```

Use command “**bt disconnect**” to release the connection with Headset.

```
>> bt disconnect
>> 46 689546 [iot_thread] [ERROR] [NET] [lu] Error -27648 while sending data.
47 689547 [AWS-RemoteCtrl] [INFO] [MQTT] [lu] (MQTT connection 0x20230c48) MQTT
PUBLISH operation queued.
Disconnected (reason 0x16)
58 249328 [iot_thread] [INFO] [Shadow] [lu] Shadow UPDATE of MusicPlayer was
ACCEPTED.
59 249329 [AWS-RemoteCtrl] Successfully performed update.
```

Use command “**bt deletedevice**” to remove bound and authentication information of all the connected devices.

```
>> bt deletedevice
```

&gt;&gt;

## 5.17 wifi\_provisioning Sample Application

This example demonstrates how the Wi-Fi of the i.MX RT1060 EVK board with IW416 NXP-based wireless module can be configured by Android mobile application, and publish the Wi-Fi AP information to the AWS IoT.

AWS Wi-Fi provisioning example will start advertising if the Wi-Fi AP is not configured, wait till the Wi-Fi AP configuration is completed. After connected to the Android APK, the example will execute the request from cellphone and reply the response. When the Wi-Fi AP is configured, the Shadow demo will connect to the AWS via Wi-Fi and publish the configured Wi-Fi AP information.

**NOTE:**

- This example could NOT function with the default setting provided in SDK package because an AWS account is mandatory to run to the example, the end users must create their own AWS account and configure the IoT Console before the functionality of the example could be used. Also, some information specified by the end customers, like Thing name, broker endpoint, etc., must be updated accordingly before the example would work. Please follow the steps mentioned here to configure the same.
- To make sure the NVM area is clean, all flash sector need be erased before downloading demo code.

### 5.17.1 User Configurations

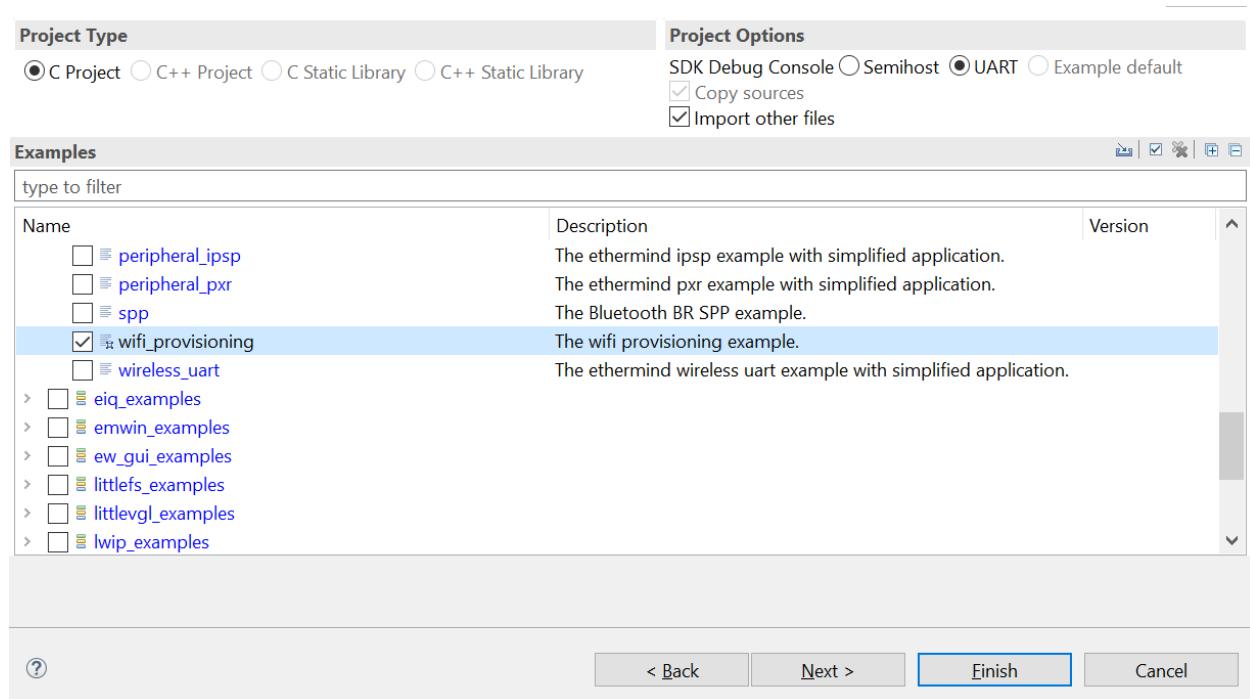
The user can modify the demo-related configurations file “*aws\_clientcredential.h*”. File path is referenced in section 1.3 “References”.

Table 25: wifi\_provisioning Application Configurations

Feature	Macro definition	Value set for Example	File name	Details
AWS Client Credentials	clientcredentialMQTT_BROKER_ENDPOINT	“a2qkq65ssjggf7-ats.iot.us-east-1.amazonaws.com”	aws_clientcredential.h	These credentials are required to connect the correct end point of AWS IoT Thing.
	clientcredentialIOT_THING_NAME	“aws_wifi_provisioning”		

### 5.17.2 wifi\_provisioning Application Execution

Please refer to the previous section 3.1.3 to run the demo using MCUXpresso IDE. Refer below figures for importing Bluetooth example and selection of Bluetooth module.



Please refer to the previous sections 3.1.4, 3.1.5 and 3.1.6 for instructions on importing a project, building an application, running an application in debug mode and flashing an application program for a few IDEs. Please refer to section 2.1 for information about the serial console setup.

#### 5.17.2.1 Create and configure AWS Account

Follow the steps described in section 5.16.2.1 to create a new AWS account.

#### 5.17.2.2 Create an AWS IoT Policy

Follow the steps described in section 5.16.2.2 to create an AWS IoT Policy.

⌚ Success  
Successfully created a policy.

AWS IoT > Policies

## Policies

Create

Search policies	🔍
<input type="checkbox"/> Name	...
<input type="checkbox"/> myIoTPolicy	...
<input type="checkbox"/> MusicPlayerPolicy	...
<input type="checkbox"/> aws_wifi_provisioning_policy	...

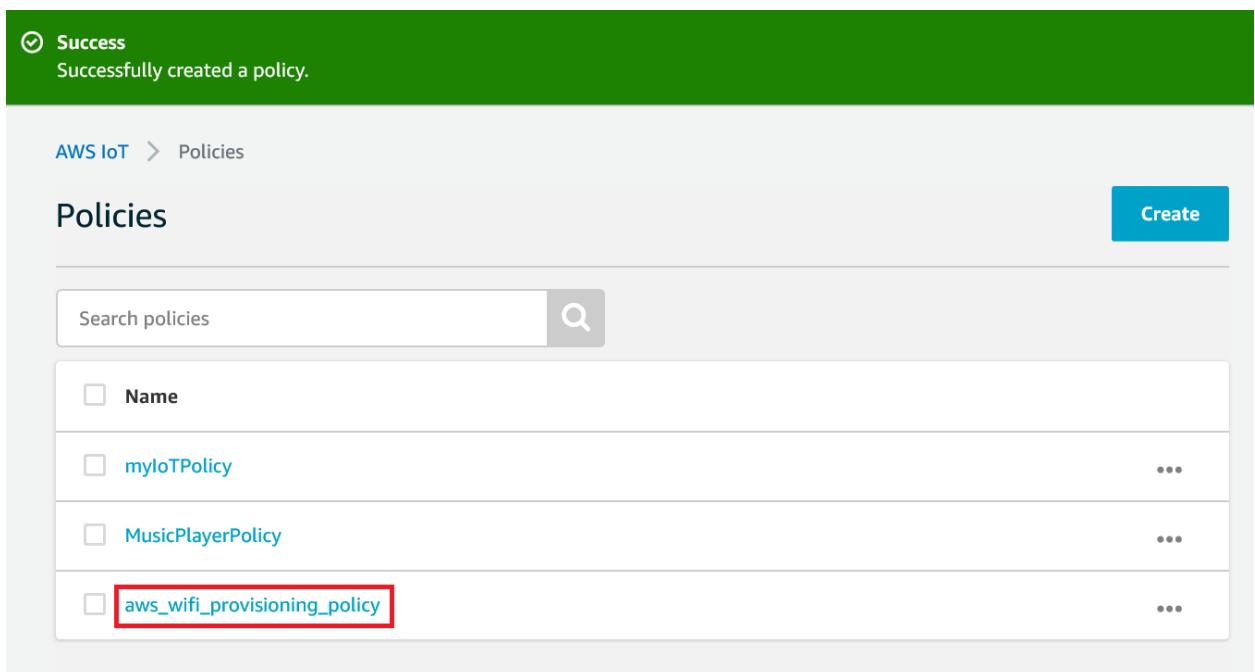


Figure 80: New policy named “aws\_wifi\_provisioning\_policy” is created

### 5.17.2.3 Create IoT thing, private key, and certificate for device

Follow the steps described in section 5.16.2.3 to Create IoT thing, private key and certification for device.

⌚ Success  
Successfully registered your thing

AWS IoT > Things

## Things

Create

Search things	🔍	Fleet Indexing	Info
<input type="checkbox"/> Name	Type		...
<input type="checkbox"/> MusicPlayer	NO TYPE	...	...
<input type="checkbox"/> myNXPthing	NO TYPE	...	...
<input type="checkbox"/> aws_wifi_provisioning	NO TYPE	...	...

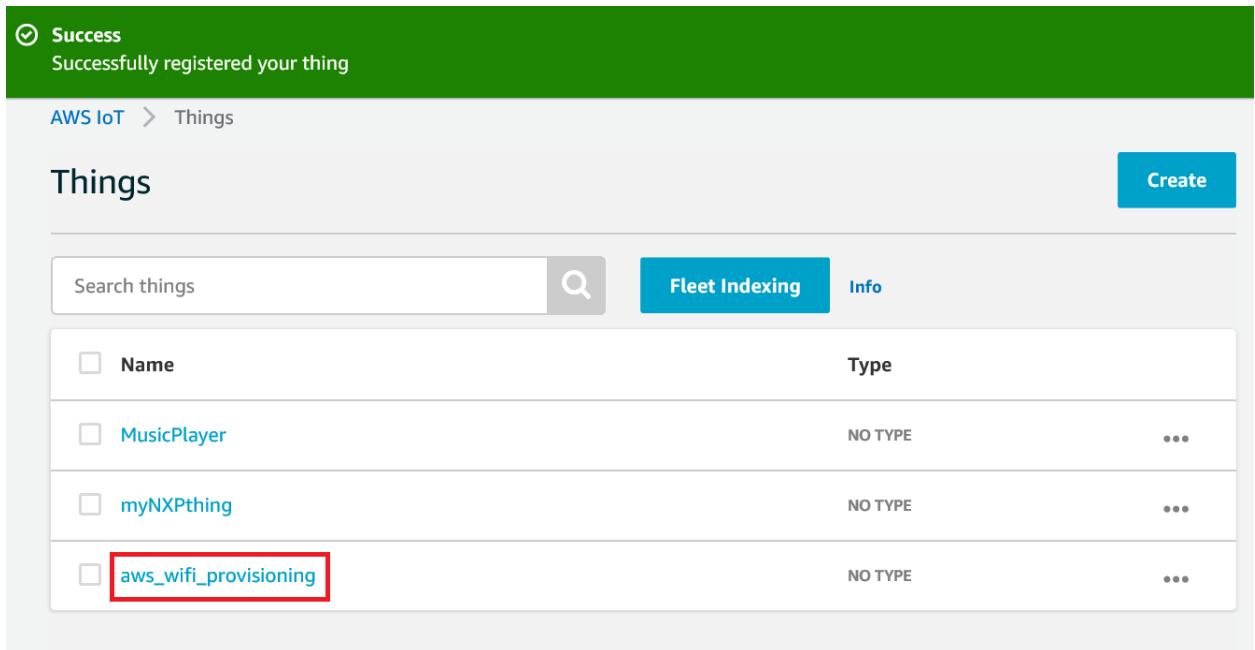


Figure 81: Create new thing name “aws\_wifi\_provisioning”

#### 5.17.2.4 Configure the AWS IoT Certificate and Private Keys

Follow the steps described in section 5.16.2.4 to configure the AWS IoT certificate and private keys.

#### 5.17.2.5 Configure the AWS IoT endpoint

User need to update FreeRTOS with your AWS IoT endpoint so the application running on the board can send requests to the correct endpoint.

Open file "aws\_clientcredential.h".

Set the "**clientcredentialMQTT\_BROKER\_ENDPOINT**" as per the Rest API Endpoint.

```
#define clientcredentialMQTT_BROKER_ENDPOINT      "a2qkq65ssjggf7-ats.iot.us-  
east-1.amazonaws.com"
```

Set the "**clientcredentialIOT\_THING\_NAME**" as per the name of IoT Thing

```
#define clientcredentialIOT_THING_NAME          "aws_wifi_provisioning"
```

Rebuild the application and flash it on the target board.

Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.

#### 5.17.2.6 Prepare the Android application

To create an Amazon Cognito user pool follow the steps:

Open the Amazon Cognito console,

<https://console.aws.amazon.com/cognito/home>

Choose "Manage User Pools"

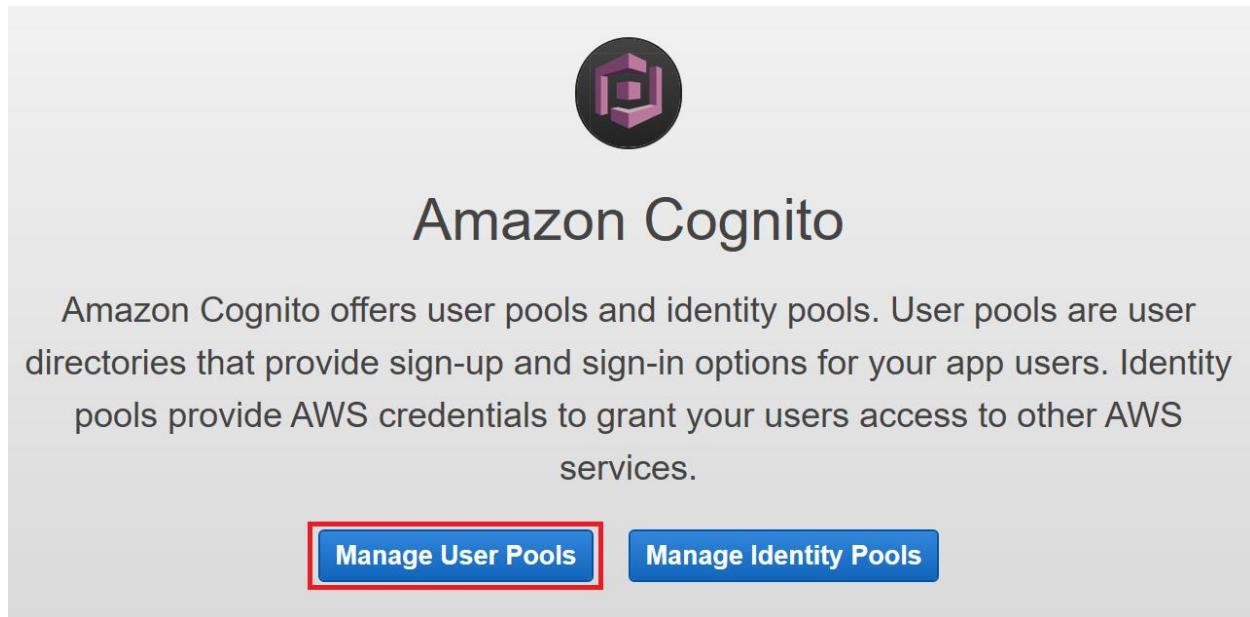


Figure 82: Manage User Pools

Click "Create a User Pool"

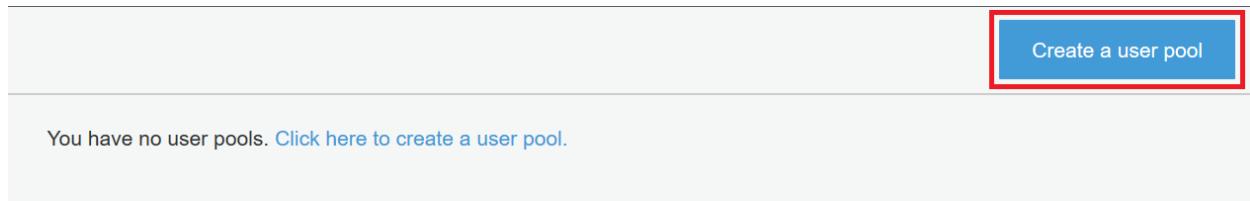


Figure 83: Create a new user pool

Give the user pool a name, and then choose "Review defaults". Such as the pool name is "aws\_wifi\_provisioning\_user\_pool".

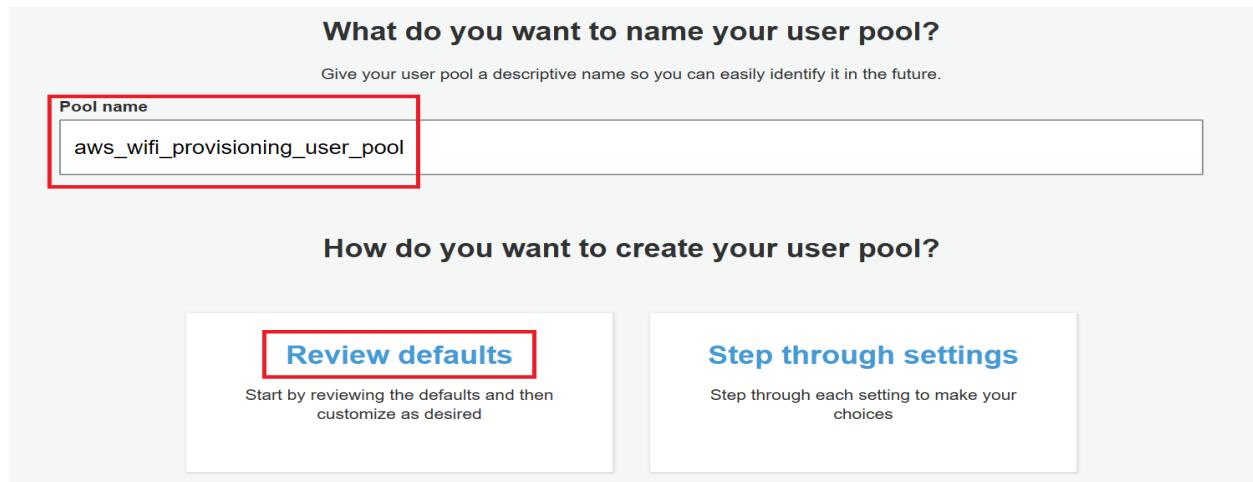


Figure 84: Name the new user pool

From the navigation pane, choose "App clients", and then choose "Add an app client".

User Pools | Federated Identities

## Create a user pool

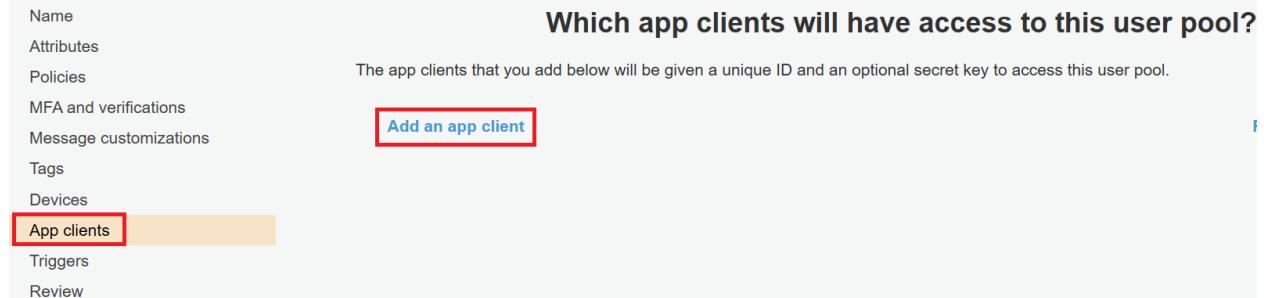


Figure 85: Add an application client

Enter a name for the app client, and then choose "**Create app client**". Such as the app client name is "aws\_wifi\_provisioning\_apk".

App client name  
aws\_wifi\_provisioning\_apk

Refresh token expiration (days)  
30

Generate client secret

### Auth Flows Configuration

Enable sign-in API for server-based authentication (ADMIN\_NO\_SRP\_AUTH) [Learn more.](#)

Only allow Custom Authentication (CUSTOM\_AUTH\_FLOW\_ONLY) [Learn more.](#)

Enable username-password (non-SRP) flow for app-based authentication (USER\_PASSWORD\_AUTH) [Learn more.](#)

[Set attribute read and write permissions](#)

[Cancel](#) [Create app client](#)

Figure 86: Name the new application client and create an application client

From the navigation pane, choose "**Review**", and then choose "**Create pool**".

User Pools | Federated Identities

### Create a user pool

Name  
aws\_wifi\_provisioning\_user\_pool

Attributes

Policies

MFA and verifications

Message customizations

Tags

Devices

App clients

Triggers

**Review**

Required attributes email

Alias attributes [Choose alias attributes...](#)

Username attributes [Choose username attributes...](#)

Custom attributes [Choose custom attributes...](#)

Minimum password length 8

Password policy uppercase letters, lowercase letters, special characters, numbers

User sign ups allowed? Users can sign themselves up

Figure 87: Create a user pool

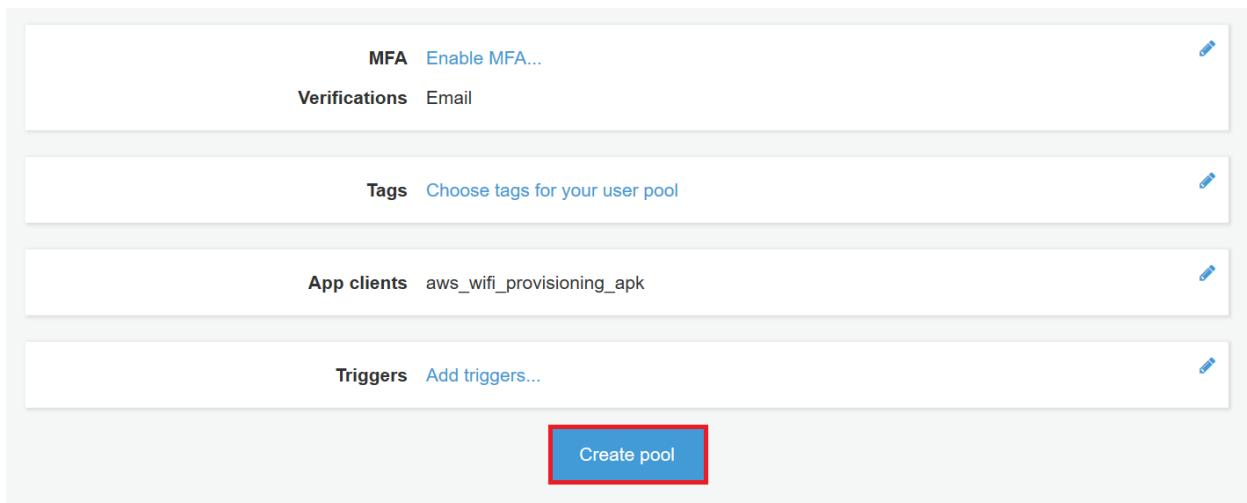


Figure 88: Create pool

Make a note of the pool ID that appears on the "General Settings" page of your user pool.

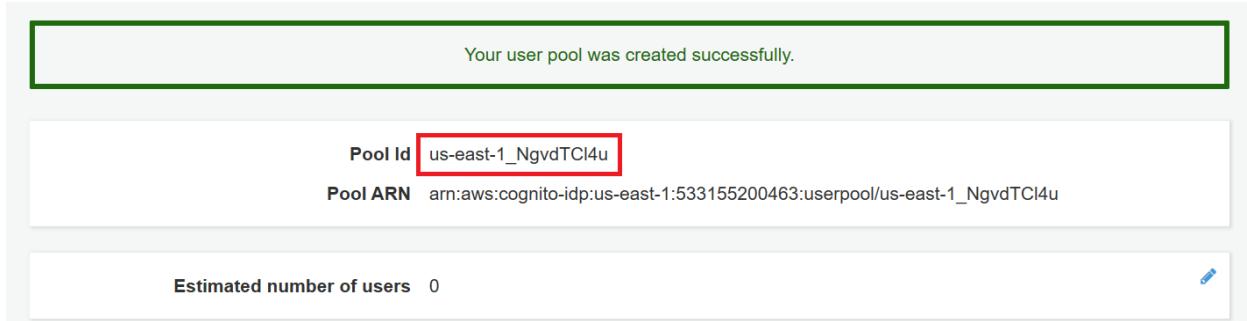


Figure 89: User pool created successfully

From the navigation pane, choose "App clients", and then choose "Show details".

The screenshot shows the 'App clients' section of the navigation pane with 'App clients' highlighted by a red box. On the right, a modal window titled 'Which app clients will have access to this user pool?' displays the app client details. The 'App client id' is listed as 'c32m77h1h4tsv4csdr2gqbj3s'. The 'Show Details' button at the bottom of the modal is highlighted with a red rectangular border.

Figure 90: Show Details

Make a note of the “App client ID” and “App client secret”.

The screenshot shows a configuration form with the following fields:

- aws\_wifi\_provisioning\_apk
- App client id: c32m77h1h4tsv4csdr2gqbj3s
- App client secret: 1r4oqfipu6il3slkbusjra2uf55rn9sdhpupctnh05ke8srln2
- Refresh token expiration (days): 30

Figure 91: App client ID and App client secret ID

To create an Amazon Cognito identity pool follow the steps:

Open the Amazon Cognito console,

<https://console.aws.amazon.com/cognito/home>

Choose "Manage Identity Pools"

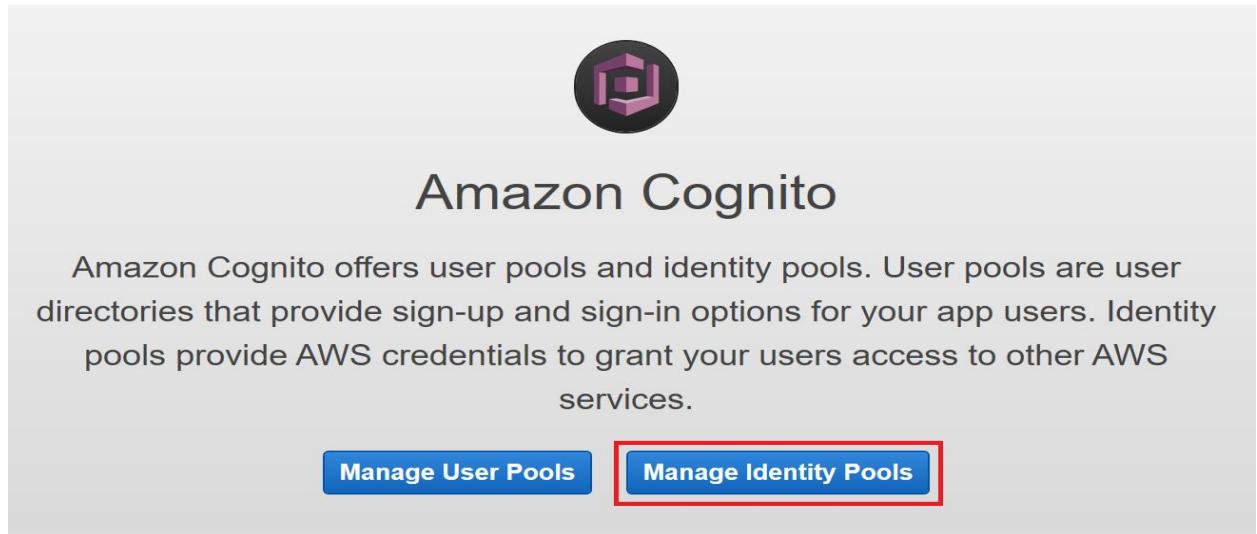


Figure 92: Manage Identity Pools

Click “Create new identity pool”.

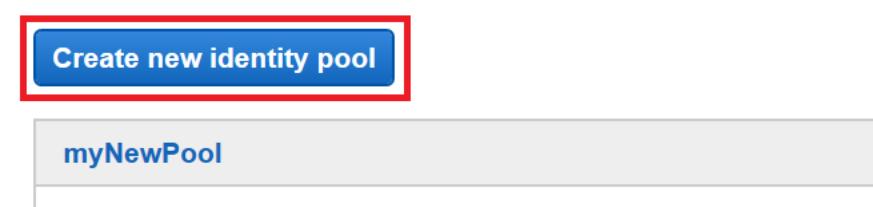


Figure 93: Manage Identity Pools

Enter a name for your identity pool. Such as the pool name is "aws\_wifi\_provisioning\_identity\_pool", enable unauthenticated access.

### Create new identity pool

Identity pools are used to store end user identities. To declare a new identity pool, enter a unique name.

Identity pool name\*  

Example: My App Name

#### ▼ Unauthenticated identities

Amazon Cognito can support unauthenticated identities by providing a unique identifier and AWS credentials for users who do not authenticate with an identity provider. If your application allows customers to use the application without logging in, you can enable access for unauthenticated identities. [Learn more about unauthenticated identities](#).

Enable access to unauthenticated identities

Enabling this option means that anyone with internet access can be granted AWS credentials. Unauthenticated identities are typically users who do not log in to your application. Typically, the permissions that you assign for unauthenticated identities should be more restrictive than those for authenticated identities.

Figure 94: Add name for Identity pool

### Expand "Authentication providers"

#### ▼ Authentication flow settings

A user authenticating with Amazon Cognito will go through a multi-step process to bootstrap their credentials. Amazon Cognito has two different flows for authentication with public providers: enhanced and basic. Cognito recommends the use of enhanced authentication flow. However, if you still wish to use the basic flow, you can enable it here. [Learn more about authentication flows](#).

Allow Basic (Classic) Flow

 Authentication providers 

\* Required

Cancel

**Create Pool**

Figure 95: Authentication providers

Choose the "**Cognito**" tab, and then enter your user pool ID and app client ID and choose "**Create Pool**".

▼ Authentication providers ⓘ

Amazon Cognito supports the following authentication methods with Amazon Cognito Sign-In or any public provider. If you allow your users to authenticate using any of these public providers, you can specify your application identifiers here. Warning: Changing the application ID that your identity pool is linked to will prevent existing users from authenticating using Amazon Cognito. [Learn more about public identity providers](#).

Configure your Cognito Identity Pool to accept users federated with your Cognito User Pool by supplying the User Pool ID and the App Client ID.

User Pool ID: us-east-1\_NgvdTCI4u

App client id: c32m77h1h4tsv4csdr2gqbj3s

Add Another Provider

\* Required

Create Pool

Figure 96: Create Pool

Expand "**View Details**", and make a note of the two IAM role names.

Choose "**Allow**" to create the IAM roles for authenticated and unauthenticated identities to access Amazon Cognito.

▼ Hide Details

**Role Summary** ⓘ

Role Description Your authenticated identities would like access to Cognito.

IAM Role Create a new IAM Role

Role Name Cognito\_aws\_wifi\_provisioning\_identi

View Policy Document

**Role Summary** ⓘ

Role Description Your unauthenticated identities would like access to Cognito.

IAM Role Create a new IAM Role

Role Name Cognito\_aws\_wifi\_provisioning\_identi

Allow

Figure 97: Role names

Make a note of the “*Identity pool ID*”.

## Getting started with Amazon Cognito

Platform Android ▾

### ▼ Download the AWS SDK

[Download the AWS SDK for Android](#) [Developer Guide](#)

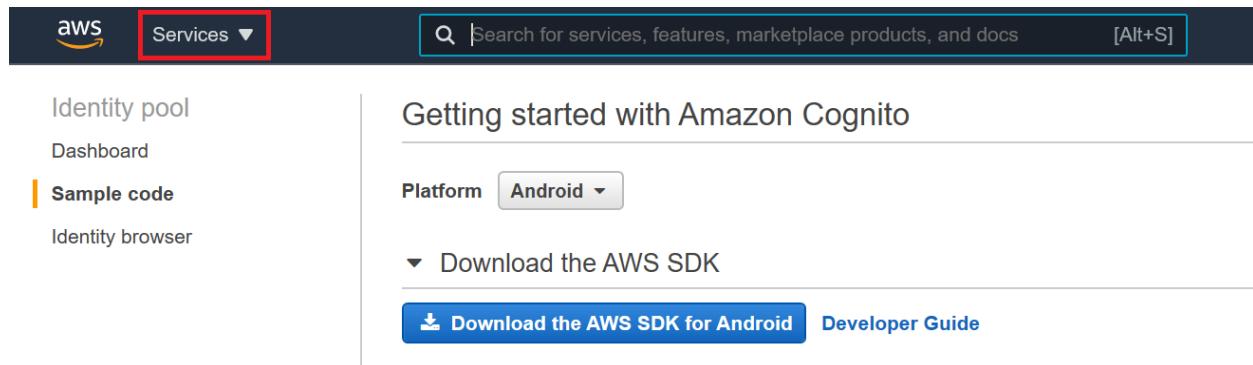
### ▼ Get AWS Credentials

```
// Initialize the Amazon Cognito credentials provider
CognitoCachingCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
    getApplicationContext(),
    "us-east-1:9217806e-f108-4751-a3b9-b2b0c1b4df69", // Identity pool ID
    Regions.US_EAST_1 // Region
);
```

Figure 98: Identity pool ID

**To create and attach an IAM policy to the authenticated identity follow the steps:**

Click “Services”



The screenshot shows the AWS Management Console interface. At the top, there's a navigation bar with the AWS logo, a search bar containing "Search for services, features, marketplace products, and docs [Alt+S]", and a "Services ▾" button. On the left, a sidebar menu includes "Identity pool", "Dashboard", "Sample code" (which is highlighted with an orange border), and "Identity browser". The main content area is titled "Getting started with Amazon Cognito" and shows the same "Getting started with Amazon Cognito" page as Figure 98, with the "Identity pool ID" field highlighted. The "Platform" dropdown is set to "Android".

Figure 99: Open services tab

Click “IAM” inside “All Services”

The screenshot shows the AWS Services menu with the search bar at the top. Below it is a grid of services. In the 'Security, Identity, & Compliance' column, the 'IAM' service is highlighted with a red box.

Networking & Content Delivery	Media Services	MSK	AppStream 2.0
VPC	Kinesis Video Streams	AWS Glue DataBrew	WorkDocs
CloudFront	MediaConnect	Resource Access Man...	WorkLink
Route 53	MediaConvert	IAM	Internet of Things
API Gateway	MediaLive	Cognito	IoT Core
Direct Connect	MediaPackage	Secrets Manager	FreeRTOS
AWS App Mesh	MediaStore	GuardDuty	IoT 1-Click
AWS Cloud Map	MediaTailor	Amazon Interactive Vid...	IoT Analytics
Global Accelerator	Elemental Appliances ...	Inspector	IoT Device Defender
	Elastic Transcoder	Amazon Macie	IoT Device Management
			IoT Events

Figure 100: Selecting IAM from Security, Identity and Compliance section

Click “Roles” from “IAM dashboard”.

## IAM dashboard

### Security recommendations 1

#### ! Add MFA for root user

Enable multi-factor authentication (MFA) for the root user to improve security for this account.

[Add MFA](#)

#### ✓ Root user has no active access keys

Using access keys attached to an IAM user instead of the root user improves security.

### IAM resources

User groups	Users	Roles	Policies	Identity providers
0	0	19	2	0

Figure 101: Open the available roles

Find and choose your authenticated identity's role.

[Create role](#) [Delete role](#)

Search			
Role name	Trusted entities	Last activity	
<input type="checkbox"/> AWSServiceRoleForSupport	AWS service: support (Service-Linked role)	None	
<input type="checkbox"/> AWSServiceRoleForTrustedAdvisor	AWS service: trustedadvisor (Service-Linked ...)	None	
<input type="checkbox"/> Cognito_aws_wifi_provisioning_id...	Identity Provider: cognito-identity.amazonaw... Cognito_aws_wifi_provisioning_identity_poolAuth_Role	None	
<input type="checkbox"/> Cognito_aws_wifi_provisioning_id...	Identity Provider: cognito-identity.amazonaw...	None	

Figure 102: Select authentication role

choose "Permissions policies", and then choose "Add inline policy".

The screenshot shows the AWS IAM 'Permissions policies' page. At the top, there are tabs: 'Permissions' (highlighted with a red box), 'Trust relationships', 'Tags', 'Access Advisor', and 'Revoke sessions'. Below the tabs, it says 'Permissions policies (1 policy applied)'. There is a blue 'Attach policies' button and a red 'Add inline policy' button (also highlighted with a red box). The main table lists one policy:

Policy name	Policy type	Actions
oneClick_Cognito_aws_wifi_provisioning_identity_poolAuth_F	Inline policy	<span style="color: #0070C0;">X</span>

Figure 103: Add inline policy

Choose the "JSON" tab, and paste the following JSON.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Create policy

1 2

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

The screenshot shows the 'Create policy' page. It has tabs: 'Visual editor' (disabled) and 'JSON' (highlighted with a red box). To the right are 'Import managed policy' and 'Cancel' buttons. The main area contains the JSON code:

```

1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": [
7                  "iot:AttachPolicy",
8                  "iot:AttachPrincipalPolicy",
9                  "iot:Connect",
10                 "iot:Publish",
11                 "iot:Subscribe",
12                 "iot:Receive",
13                 "iot:GetThingShadow",
14                 "iot:UpdateThingShadow",
15                 "iot:DeleteThingShadow"
16             ],
17             "Resource": [
18                 "*"
19             ]
20         }
21     ]
22 }
```

Character count: 410 of 10,240.  
The current character count includes character for all inline policies in the role: Cognito\_aws\_wifi\_provisioning\_identity\_poolAuth\_Role.

[Cancel](#) [Review policy](#)

Figure 104: Update the content of JSON

Enter a name for the policy, and then choose "**Create policy**".

### Review policy

Before you create this policy, provide the required information and review this policy.

The screenshot shows the AWS IAM 'Create New Policy' wizard. In the first step, 'Add the name of policy', the 'Name\*' input field contains 'AWS\_IoT'. Below it, a note says 'Maximum 128 characters. Use alphanumeric and '+,-,\_' characters.' The second step, 'Review policy', displays a table with one service row: 'Allow (1 of 264 services) Show remaining 263'. The row for 'IoT' has 'Access level' set to 'Limited: Read, Write, Permissions management', 'Resource' set to 'All resources', and 'Request condition' set to 'None'. At the bottom, there are buttons for '\* Required', 'Cancel', 'Previous', and a red-bordered 'Create policy' button.

Figure 105: Add the name of policy

**To create and attach an IAM policy to the unauthenticated identity follow the same steps.**

#### 5.17.2.7 Prepare Configuration File for the Android Application

Prepare "aws\_wifi\_provisioning.properties" file with yours AWS credentials.

It's structure looks like this:

```
customer_specific_endpoint=<REST API ENDPOINT>
cognito_pool_id=<COGNITO POOL ID>
thing_name=<THING NAME>
region=<REGION>
policy_name=<POLICY>
```

For Example:

```
customer_specific_endpoint=a2qkq65ssjggf7-ats.iot.us-east-1.amazonaws.com
cognito_pool_id=us-east-1_NgvdTC14u
thing_name=aws_wifi_provisioning
region=us-east-1
policy_name=aws_wifi_provisioning_policy
```

#### 5.17.2.8 Run the Android Application

Follow the README.md guides to update the configuration of the Android SDK for FreeRTOS Bluetooth devices.

```
<MCUXpressoSDK>\boards\evkmimxrt1060\edgefast_bluetooth_examples\wifi_provisioning\amazon-freertos-ble-android-sdk\README.md
<MCUXpressoSDK>\boards\evkmimxrt1060\edgefast_bluetooth_examples\wifi_provisioning\amazon-freertos-ble-android-sdk\app\README.md
```

Open project "amazon-freertos-ble-android-sdk" from

<MCUXpressoSDK>\boards\evkmimxrt1060\edgefast\_bluetooth\_examples\wifi\_provisioning in Android Studio, build it and attach the Android device.

Allow USB debugging on the cell phone.

**NOTE:** If the blank screen is shown when start running the APK, it maybe the network issue.

Please view the ticket <https://github.com/aws/amazon-freertos-ble-android-sdk/issues/34> for detail information.

#### 5.17.2.9 Run the application

**NOTE:** There are some limitations for the example,

1. Only one Wi-Fi network configuration could be saved.

Please view the ticket <https://github.com/aws/amazon-freertos/issues/2678> for detail information.

2. The building warnings on IAR and ARM GCC tool-chain.

Please view the ticket <https://github.com/aws/amazon-freertos/issues/2638> for detail information.

3. Once the Wi-Fi network is configured, the Bluetooth LE will be disconnected and then disabled. So the Bluetooth LE connection could not be established after the Wi-Fi configured successfully.

4. If the log "[wm\_wlan] Connection Failed !" following the 3 times log "[wm\_wlan] Connect to AP FAIL ! Retrying ....", it means the Wi-Fi network is not established. Please check whether the entered passkey is correct, whether Wi-Fi AP is working, or whether the example is blocked by blacklist or whitelist. And then retry again.

The log below shows the output of the example in the console window. The log can be different based on your Wi-Fi network configuration and based on the actions, which you have done in the Android application.

Power reset the i.MX RT1060 EVK board, following logs will be appear on the console:

```
0 25 [main_task] [INFO ][MSD_FATFS][lu] USB Host stack successfully initialized
1 129 [main_task] Write certificate...
2 160 [iot_thread] [INFO ][DEMO][lu] -----STARTING DEMO-----

3 161 [iot_thread] [INFO ][INIT][lu] SDK successfully initialized.
MAC Address: 20:4E:F6:25:F3:19
[net] Initialized TCP/IP networking stack
[wm_wlan] WLAN_REASON_INITIALIZED

4 4367 [Bluetooth RD Ta] [INFO ][IOT_BLE_HAL_COMMON_GAP][lu] Bluetooth ON
Initialization Completed.
5 4368 [Bluetooth RD Ta] [INFO ][IOT_BLE_HAL_COMMON_GAP][lu] Stack Version -
016.004.007.
6 4378 [iot_thread] [INFO ][DEMO][lu] No networks connected for the demo.
Waiting for a network connection.
```

Run the application from Android Studio and Load the preferences "aws\_wifi\_provisioning.properties" from the cell phone application.

Create an account if not already registered. And Click on **WIFI PROVISIONING**.

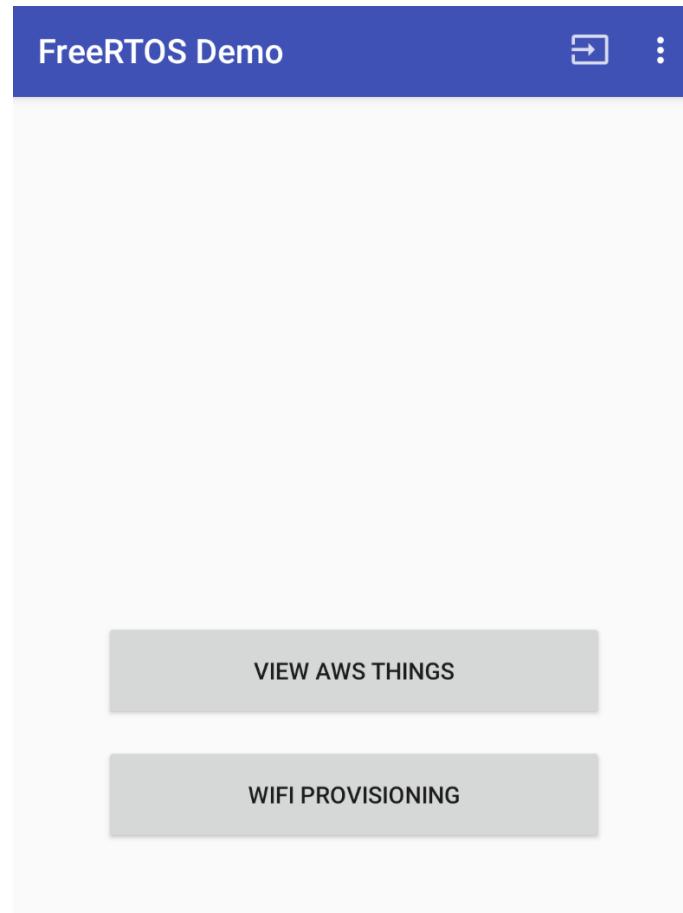


Figure 106: Android application

Start scanning for nearby BLE devices:

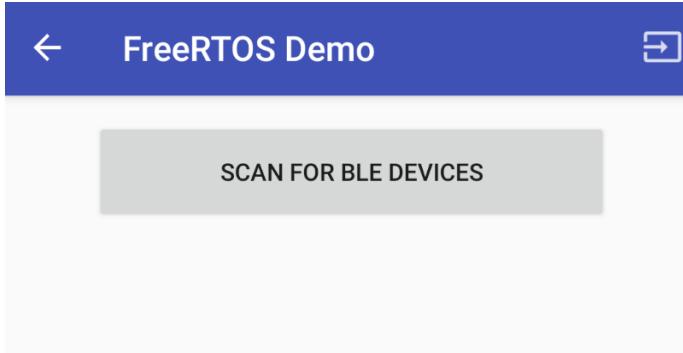


Figure 107: Scan for BLE devices

Devices will be available as shown below in the cell phone application:

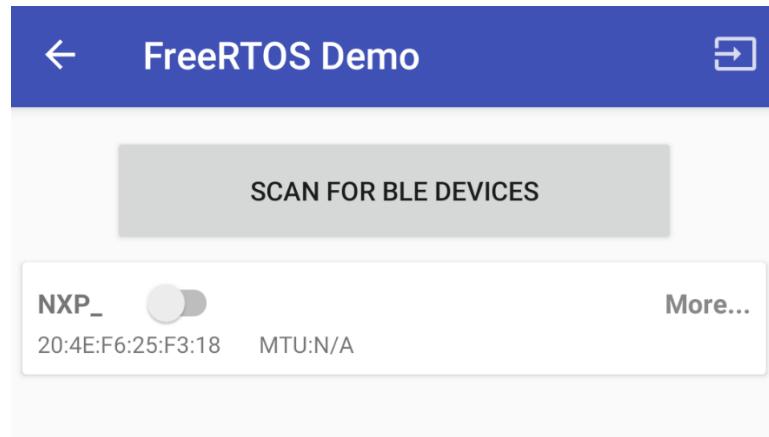


Figure 108: Available Nearby BLE devices

Toggle the button to initiate the connection with i.MX RT1060 EVK board and IW416 NXP-based wireless module:

On successful connection with IW416 wireless module, following logs will be appear on the i.MX RT1060 EVK board console:

```
7 20634 [Bluetooth RD Ta] [INFO ] [DEMO] [lu] BLE Connected to remote device,  
connId = 128
```

Now, click “More...” in cell phone application and start “Wi-Fi Provisioning” experience:

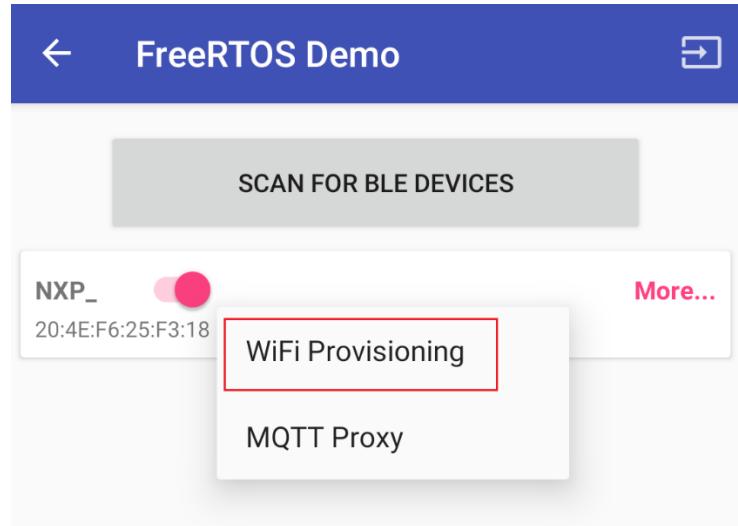


Figure 109: Enable Wi-Fi Provisioning

Available Wi-Fi networks will be visible in the cell phone application:

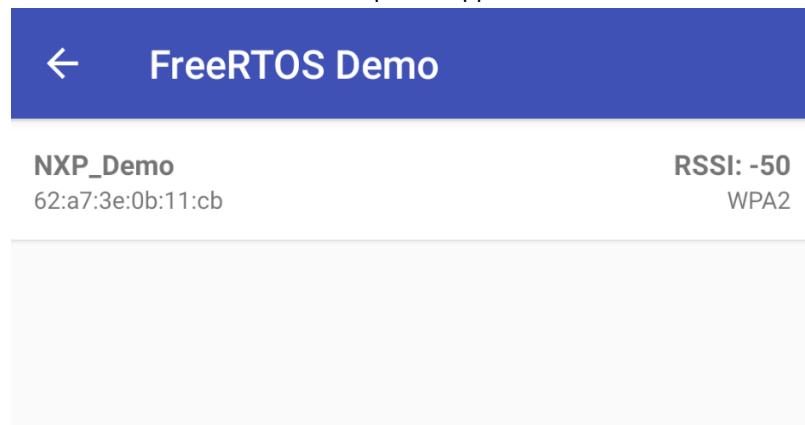


Figure 110: Available Wi-Fi devices

Click on the network to be connected and enter the password, on successful connection following logs will be appear on the i.MX RT1060 EVK board console:

```
[wm_wlan] Connecting to NXP_Demo .....

9 80380 [Bluetooth RD Ta] [INFO ] [BLE_WIFI_PROV] [lu] Connect flag not set in
request, using default value always connect = 1
[wm_wlan] Connected to with IP = [192.168.43.149]

10 105888 [iot_thread] [INFO ] [DEMO] [lu] Successfully initialized the demo.
Network type for the demo: 1
11 105888 [iot_thread] [INFO ] [MQTT] [lu] MQTT library successfully initialized.
12 105888 [iot_thread] [INFO ] [Shadow] [lu] Shadow library successfully
initialized.
13 105888 [iot_thread] [INFO ] [DEMO] [lu] Shadow Thing Name is
aws_wifi_provisioning (length 21).
14 118064 [iot_thread] [INFO ] [MQTT] [lu] Establishing new MQTT connection.
15 118064 [iot_thread] [INFO ] [MQTT] [lu] Anonymous metrics (SDK language, SDK
version) will be provided to AWS IoT. Recompile with
AWS_IOT_MQTT_ENABLE_METRICS set to 0 to disable.
16 118068 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, CONNECT
operation 0x202370c0) Waiting for operation completion.
17 118395 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, CONNECT
operation 0x202370c0) Wait complete with result SUCCESS.
18 118396 [iot_thread] [INFO ] [MQTT] [lu] New MQTT connection 0x202267d8
established.
19 118397 [iot_thread] [INFO ] [Shadow] [lu] (aws_wifi_provisioning) Modifying
Shadow DELTA callback.
20 118397 [iot_thread] [INFO ] [Shadow] [lu] (aws_wifi_provisioning) Adding new
DELTA callback.
21 118398 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
22 118400 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x20236a68) Waiting for operation completion.
23 118713 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x20236a68) Wait complete with result SUCCESS.
24 118714 [iot_thread] [INFO ] [Shadow] [lu] (aws_wifi_provisioning) Shadow DELTA
callback operation complete with result SUCCESS.
25 118715 [iot_thread] [INFO ] [Shadow] [lu] (aws_wifi_provisioning) Modifying
Shadow UPDATED callback.
26 118715 [iot_thread] [INFO ] [Shadow] [lu] (aws_wifi_provisioning) Adding new
UPDATED callback.
```

```

27 118717 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
28 118717 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x20236a68) Waiting for operation completion.
29 119034 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x20236a68) Wait complete with result SUCCESS.
30 119035 [iot_thread] [INFO ] [Shadow] [lu] (aws_wifi_provisioning) Shadow
UPDATED callback operation complete with result SUCCESS.
31 119038 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
32 119038 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Waiting for operation completion.
33 119337 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Wait complete with result SUCCESS.
34 119338 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
35 119340 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Waiting for operation completion.
36 119653 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Wait complete with result SUCCESS.
37 119656 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) MQTT
PUBLISH operation queued.
38 119976 [iot_thread] [WARN ] [Shadow] [lu] Shadow DELETE of
aws_wifi_provisioning was REJECTED.
39 119977 [iot_thread] [WARN ] [Shadow] [lu] Code 404: "No shadow exists with
name: 'aws_wifi_provisioning'".
40 119978 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0)
UNSUBSCRIBE operation scheduled.
41 119980 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Waiting for operation completion.
42 120523 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Wait complete with result SUCCESS.
43 120525 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0)
UNSUBSCRIBE operation scheduled.
44 120525 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Waiting for operation completion.
45 120824 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Wait complete with result SUCCESS.
46 120825 [iot_thread] [INFO ] [DEMO] [lu] Successfully cleared Shadow of
aws_wifi_provisioning.
47 120825 [iot_thread] [INFO ] [DEMO] [lu] Shadow update
{"state":{"reported":{"wifi_list":{"count":0,"wifi":[]}}}, "clientToken": "120825"
}
48 120827 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
49 120827 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Waiting for operation completion.
50 121151 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Wait complete with result SUCCESS.
51 121152 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
52 121152 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Waiting for operation completion.
53 121453 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Wait complete with result SUCCESS.
54 121457 [iot_thread] [INFO ] [MQTT] [lu] (MQTT connection 0x20236fa0) MQTT
PUBLISH operation queued.
55 121780 [iot_thread] [INFO ] [DEMO] [lu] Shadow was updated!
Previous: {"state":{}}
Current: {"state":{"reported":{"wifi_list":{"count":0,"wifi":[]}}}}
56 121798 [iot_thread] [INFO ] [Shadow] [lu] Shadow UPDATE of
aws_wifi_provisioning was ACCEPTED.

```

```

57 121799 [iot_thread] [INFO ][DEMO][lu] Successfully sent Shadow update.
58 122799 [iot_thread] [INFO ][DEMO][lu] Saved wifi config: ssid NXP_Demo,
index 0
59 122799 [iot_thread] [INFO ][DEMO][lu] Shadow update
{"state":{"reported":{"wifi_list":{"count":1,"wifi":[{"ssid":"NXP_Demo","bssid":"62A73E0B11CB","security":3}]}}, "clientToken": "122799"}
60 122804 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0) MQTT
PUBLISH operation queued.
61 123170 [iot_thread] [INFO ][DEMO][lu] Shadow was updated!
Previous: {"state":{"reported":{"wifi_list":{"count":0,"wifi":[]}}}}
Current:
{"state":{"reported":{"wifi_list":{"count":1,"wifi":[{"ssid":"NXP_Demo","bssid":"62A73E0B11CB","security":3}]}}, "clientToken": "123177"} 62 123177 [iot_thread] [INFO ][Shadow][lu]
Shadow UPDATE of aws_wifi_provisioning was ACCEPTED.
63 123178 [iot_thread] [INFO ][DEMO][lu] Successfully sent Shadow update.
64 123179 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
65 123179 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Waiting for operation completion.
66 123733 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Wait complete with result SUCCESS.
67 123734 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0) SUBSCRIBE
operation scheduled.
68 123734 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Waiting for operation completion.
69 124036 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0, SUBSCRIBE
operation 0x202371d8) Wait complete with result SUCCESS.
70 124039 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0) MQTT
PUBLISH operation queued.
71 124350 [iot_thread] [INFO ][Shadow][lu] Shadow DELETE of
aws_wifi_provisioning was ACCEPTED.
72 124352 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0)
UNSUBSCRIBE operation scheduled.
73 124352 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Waiting for operation completion.
74 124918 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Wait complete with result SUCCESS.
75 124919 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0)
UNSUBSCRIBE operation scheduled.
76 124919 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Waiting for operation completion.
77 125202 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0,
UNSUBSCRIBE operation 0x202371d8) Wait complete with result SUCCESS.
78 125203 [iot_thread] [INFO ][DEMO][lu] Successfully cleared Shadow of
aws_wifi_provisioning.
79 125203 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0)
Disconnecting connection.
80 125206 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0,
DISCONNECT operation 0x20236a68) Waiting for operation completion.
81 125207 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0,
DISCONNECT operation 0x20236a68) Wait complete with result SUCCESS.
82 125207 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0)
Connection disconnected.
83 125208 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0) Network
connection closed.
84 125483 [iot_thread] [INFO ][MQTT][lu] (MQTT connection 0x20236fa0) Network
connection destroyed.
85 125484 [iot_thread] [INFO ][Shadow][lu] Shadow library cleanup done.
86 125484 [iot_thread] [INFO ][MQTT][lu] MQTT library cleanup done.
87 125484 [iot_thread] [INFO ][DEMO][lu] Demo completed successfully.
[wm_wlan] Dis-connected
88 125538 [iot_thread] [INFO ][INIT][lu] SDK cleanup done.

```

```
89 125538 [iot_thread] [INFO ] [DEMO] [lu] -----DEMO FINISHED-----
```

## 6 Acronyms and abbreviations

Table 26: Acronyms and Abbreviations

Terms	Definition
ACS	Auto Channel Selection
AP	Access Point
API	Application Program Interface
CLI	Command Line Interface
CMSIS	Cortex® Microcontroller Software Interface Standard
DFP	Device Family Pack
DHCP	Dynamic Host Configuration Protocol
DHCPD	DHCP daemon
ED	Energy Detection
EU	European Union
EVK	Evaluation Kit
Ext AP	External Access Point
Ext STA	External Station
FW	Firmware
IDE	Integrated Development Environment
IP	Internet Protocol
IwIP	Lightweight IP
NAT	Network Address Translation
PS	Power Save
Rx	Receive
SD	Secure Digital
SDK	Software Development Kit
SSID	Service Set Identifier
STA	Station/client
SW	Software
TCP	Transmission Control Protocol
TRPC	Transmit Rate-based Power Control
Tx	Transmit
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
MFP	Management Frame Protection
OTP	One Time Programmable
ETSI	European Telecommunications Standards Institute
A2DP	Advanced Audio Distribution Profile
HFP	Hands-Free Profile
SPP	Serial Port Profile
BT	Bluetooth

BLE	Bluetooth Low Energy
PXR	Proximity Reporter
PXM	Proximity Monitor
HTS	Health Thermometer Service
IPSP	Internet Protocol Support Profile
HTTP	Hypertext Transfer Protocol
AWS	Amazon Web Services
HCI	Host Controller Interface
UART	Universal Asynchronous Receiver Transmitter
PCM	Pulse-code Modulation
HS	High Speed
USB	Universal Serial Bus

## 7 Contact Us

---

Please refer following links for more product details, queries and support.

- Home Page: [nxp.com](http://nxp.com)
  - Web Support: [nxp.com/support](http://nxp.com/support)
- NXP Community: <https://community.nxp.com/>

## 8 Legal Information

### 8.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

### 8.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on

any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Evaluation products** — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out of the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US\$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

**Translations** — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability.

Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products

## 8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

## Tables

Table 1: Reference Documents.....	5
Table 2: Macros for Wi-Fi Modules .....	7
Table 3: Sample Application Features .....	10
Table 4: Wi-Fi Configurations .....	10
Table 5: IPerf Configurations .....	11
Table 6: iPerf Commands for Windows Remote Host .....	12
Table 7: iPerf Commands for Linux Remote Host....	12
Table 8: iPerf Commands for Mobile Phone Remote Host.....	12
Table 9: wifi_setup Application Features .....	44
Table 10: Wi-Fi Configurations .....	44
Table 11: wifi_cli Application Features.....	46
Table 12: wifi_webconfig Sample Application Features .....	60
Table 13: wifi_webconfig Application Wi-Fi Configurations.....	60
Table 14: Data rate parameter .....	70
Table 15: Tx power command sequences for 2.4GHz .....	73
Table 16: Tx power command sequences for 5GHz	74
Table 17: wifi_cert Application Features .....	77
Table 18: ED MAC Parameters.....	87
Table 19: ED MAC 2.4 GHz Command Operations ..	87
Table 20: ED MAC 5 GHz Command Operations ....	87
Table 21: Set ED MAC API argument .....	88
Table 22: Get ED MAC API argument.....	88
Table 23: Preprocessor Macros for Bluetooth Modules .....	92
Table 24: audio_profile Application Configurations .....	116
Table 25: wifi_provisioning Application Configurations.....	143
Table 26: Acronyms and Abbreviations .....	165

## Figures

Figure 1: Hardware Setup for wifi_iperf Sample Application with Soft AP Mode .....	8
Figure 2: Hardware Setup for wifi_iperf Sample Application with Station Mode .....	8
Figure 3: wifi_iperf Sample Application Components .....	9
Figure 4: SDK Drag and Drop in MCUXpresso .....	13
Figure 5: Device/EVK Selection in MCUXpresso .....	14
Figure 6: Sample App Selection in MCUXpresso .....	14
Figure 7: Wi-Fi Module Selection in MCUXpresso .....	15
Figure 8: Application Build in MCUXpresso .....	16
Figure 9: Build Messages in MCUXpresso .....	16
Figure 10: Initiate Debug in MCUXpresso .....	17
Figure 11: Emulator Probe Selection in MCUXpresso .....	17
Figure 12: Application Debugging in MCUXpresso ..	18
Figure 13: Binary Flashing in MCUXpresso .....	19
Figure 14: Open Project in IAR .....	22
Figure 15: Wi-Fi Module Selection in IAR .....	23
Figure 16: Application Build in IAR .....	24
Figure 17: Build Message in IAR .....	24
Figure 18: Debugger Selection in IAR .....	25
Figure 19: Initiate Debug in IAR .....	25
Figure 20: Application Debugging in IAR .....	26
Figure 21: Binary Flashing in IAR .....	26
Figure 22: Install Packages using Pack Installer in Keil .....	27
Figure 23: DFP Verification in Pack Installer in Keil ..	27
Figure 24: Open Project in Keil .....	28
Figure 25: Wi-Fi Module Selection in Keil .....	28
Figure 26: Application Build in Keil .....	29
Figure 27: Build Message in Keil .....	29
Figure 28: Debugger Selection in Keil .....	30
Figure 29: Load the application .....	30
Figure 30: Initiate Debug in Keil .....	30
Figure 31: Application Debugging in Keil .....	31
Figure 32: Application Debugging Features in Keil ..	31
Figure 33: Binary Flashing in Keil .....	32
Figure 34: wifi_webconfig flow diagram .....	60
Figure 35: wifi_webconfig Website in AP Mode .....	62
Figure 36: Connection Attempt to AP using wifi_webconfig Application .....	63
Figure 37: wifi_webconfig Website in Client Mode ..	64
Figure 38: Clear Configurations saved in mflash using website (wifi_webconfig Application) .....	66

Figure 39: Clear Configuration Success Message in wifi_webconfig Application .....	66
Figure 40: TX Frame Packet Capture .....	72
Figure 41 : Selection of audio_profile application in MCUXpresso IDE .....	117
Figure 42: Selection of Policies from AWS IoT tab ..	118
Figure 43: Creating a new policy .....	118
Figure 44: Policy Name .....	119
Figure 45: Opening the Advanced mode .....	119
Figure 46: Adding the required JSON into the policy editor window .....	120
Figure 47: Showing the success of policy creation ..	121
Figure 48: Selection of Things from AWS IoT tab ..	122
Figure 49: Creating a new Thing .....	122
Figure 50: Creating a new Thing .....	123
Figure 51: Giving name to a new thing .....	123
Figure 52: Click next to proceed for creating a new thing .....	124
Figure 53: Selecting Device Certificate configuration for a new Thing .....	124
Figure 54: Attach a policy and create a Thing .....	125
Figure 55: Downloading the certificate, public and private keys .....	126
Figure 56: Selecting the policy and Register Thing ..	127
Figure 57: Selecting Interact and opening View Settings to get Endpoint .....	127
Figure 58: Copy the AWS IoT REST API endpoint ..	128
Figure 59: Selecting the certification .....	128
Figure 60: Selecting the Private key .....	129
Figure 61: Generate and save aws_clientcredential_keys.h .....	129
Figure 62: Manage Identity Pools .....	131
Figure 63: Create new identity pool .....	131
Figure 64: Identity pool name .....	132
Figure 65: Create pool .....	132
Figure 66: Allow to create a pool .....	132
Figure 67: Open services menu .....	132
Figure 68: Open IAM .....	133
Figure 69: Open available roles .....	133
Figure 70: Selecting un-authentication role .....	134
Figure 71: Open policy content .....	134
Figure 72: Edit the policy .....	134
Figure 73: Open JSON tab .....	135
Figure 74: Review policy .....	136
Figure 75: Save changes for the role selected .....	136
Figure 76: Open Trust relationships tab .....	137

Figure 77: copy Identity pool ID.....	137
Figure 78: Play music using Android application ...	141
Figure 79: Pause music using Android application	142
Figure 80: New policy named “aws_wifi_povisioning_policy” is created ....	145
Figure 81: Create new thing name “aws_wifi_provisioning” .....	145
Figure 82: Manage User Pools .....	146
Figure 83: Create a new user pool.....	146
Figure 84: Name the new user pool .....	147
Figure 85: Add an application client .....	147
Figure 86: Name the new application client and create an application client.....	148
Figure 87: Create a user pool.....	148
Figure 88: Create pool .....	149
Figure 89: User pool created successfully .....	149
Figure 90: Show Details .....	149
Figure 91: App client ID and App client secret ID ..	150
Figure 92: Manage Identity Pools.....	150
Figure 93: Manage Identity Pools.....	150
Figure 94: Add name for Identity pool.....	151
Figure 95: Authentication providers.....	151
Figure 96: Create Pool .....	152
Figure 97: Role names .....	152
Figure 98: Identity pool ID .....	153
Figure 99: Open services tab .....	153
Figure 100: Selecting IAM from Security, Identity and Compliance section .....	154
Figure 101: Open the available roles .....	154
Figure 102: Select authentication role .....	155
Figure 103: Add inline policy .....	156
Figure 104: Update the content of JSON .....	156
Figure 105: Add the name of policy.....	157
Figure 106: Android application .....	159
Figure 107: Scan for BLE devices .....	159
Figure 108: Available Nearby BLE devices .....	160
Figure 109: Enable Wi-Fi Provisioning .....	160
Figure 110: Available Wi-Fi devices .....	161

## Contents

---

1	About this Document .....	4
1.1	Purpose and Scope .....	4
1.2	Considerations.....	4
1.3	References.....	5
2	Tool Setup .....	6
2.1	Serial Console Tool Setup .....	6
2.2	Wireshark Tool Setup .....	6
3	Wi-Fi Sample Applications .....	7
3.1	wifi_iperf Sample Application .....	8
3.1.1	User Configurations .....	10
3.1.2	IPerf Remote Host Setup.....	12
3.1.3	Run a Demo with MCUXpresso IDE	13
3.1.4	Run a demo using ARM® GCC ...	20
3.1.5	Run a demo with IAR IDE .....	22
3.1.6	Run a demo using Keil MDK/μVision .....	27
3.1.7	wifi_iperf Application Execution	33
3.2	wifi_setup Sample Application .....	44
3.2.1	wifi_setup Application Execution	44
3.3	wifi_cli Sample Application .....	46
3.3.1	wifi_cli Application Execution ...	46
3.3.2	Add CLIs in wifi_cli Sample Application.....	57
3.4	wifi_webconfig Sample Application .....	58
3.4.1	User Configurations .....	60
3.4.2	wifi_webconfig Application Execution .....	61
3.5	wifi_test_mode Sample Application .....	67
3.5.1	wifi_test_mode Application Execution .....	67
3.6	wifi_cert Sample Application.....	77
3.6.1	wifi_cert Application Execution	77
4	Useful Wi-Fi APIs .....	88
4.1	Set/Get ED MAC Feature.....	88
4.1.1	wlan_set_ed_mac_mode() .....	88
4.1.2	wlan_get_ed_mac_mode() .....	88
4.1.3	Usage and Output .....	89
5	Bluetooth Classic/Low Energy Applications ....	92
5.1	a2dp_sink Sample Application .....	93
5.1.1	a2dp_sink Application Execution	93
5.2	a2dp_source Sample Application .....	94
	5.2.1 a2dp_source Application Execution .....	94
	5.3 handsfree Sample Application .....	95
	5.3.1 handsfree Application Execution	95
	5.4 handsfree_ag Sample Application.....	96
	5.4.1 handsfree_ag Application Execution .....	96
	5.5 spp Sample Application .....	97
	5.5.1 spp Application Execution.....	97
	5.6 peripheral_hps Sample Application .....	101
	5.6.1 peripheral_hps Application Execution .....	101
	5.7 central_hpc Sample Application.....	101
	5.7.1 central_hpc Application Execution	101
	5.8 peripheral_pxr Sample Application.....	102
	5.8.1 peripheral_pxr Application Execution .....	102
	5.9 central_pxm Sample Application .....	103
	5.9.1 central_pxm Application Execution .....	103
	5.10 peripheral_ht Sample Application ...	104
	5.10.1 peripheral_ht Application Execution .....	104
	5.11 central_ht Sample Application.....	104
	5.11.1 central_ht Application Execution	104
	5.12 peripheral_ipsp Sample Application	105
	5.12.1 peripheral_ipsp Application Execution .....	105
	5.13 central_ipsp Sample Application .....	105
	5.13.1 central_ipsp Application Execution .....	105
	5.14 Wireless UART Sample Application..	106
	5.14.1 wireless_uart Application Execution .....	106
	5.15 Shell Sample Application .....	108
	5.15.1 Shell Application Execution .....	108
	5.16 audio_profile Sample Application....	116
	5.16.1 User Configurations .....	116
	5.16.2 audio_profile Application Execution .....	116

5.17 wifi_provisioning Sample Application	
143	
5.17.1 User Configurations .....	143
5.17.2 wifi_provisioning Application	
Execution .....	144
6 Acronyms and abbreviations .....	165
7 Contact Us.....	167
8 Legal Information.....	168
8.1 Definitions .....	168
8.2 Disclaimers .....	168
8.3 Trademarks.....	169
Tables.....	170
Figures .....	171

---

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

---

**© NXP B.V. 2021. All rights reserved.**

For more information, please visit:

<http://www.nxp.com>

For sales office addresses, please send an  
email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

**Date of release: 9 Sept 2021**

**Document identifier: UM11442**