

Communicating with AWS IoT Services using Bluetooth LE and FreeRTOS

[Introduction](#)

[FreeRTOS Architecture](#)

[FreeRTOS Bluetooth LE Sample Apps](#)

[FreeRTOS Aws Test Sample For Bluetooth LE](#)

Communicating with AWS IoT Services using Bluetooth LE and FreeRTOS

Communicating with AWS IoT Services using Bluetooth LE and FreeRTOS

NOTE: This section replaces *AN1362: Communicating with AWS IoT Services using Bluetooth LE and FreeRTOS*. Further updates to this application note will be provided here.

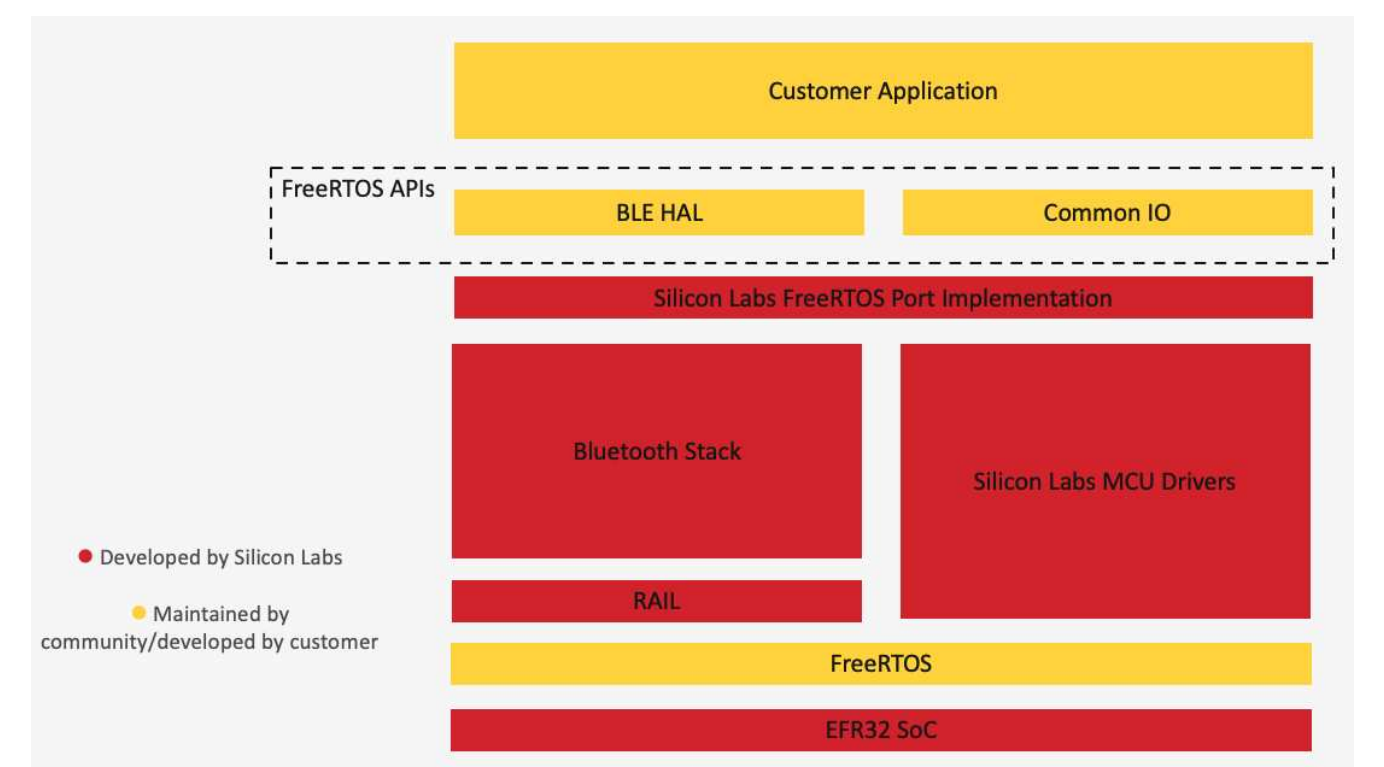
FreeRTOS is a real-time operating system supported by Amazon as an [open source project](#). It consists of a snapshot of the FreeRTOS Real-Time Operating System project, as well as a collection of standardized APIs for common MCU peripherals, such as SPI, UART, GPIO, and so on, and wireless connectivity standards, such as Bluetooth Low Energy (Bluetooth LE). Silicon Labs has integrated FreeRTOS components and sample applications into the GSDK, and the Simplicity Studio developer experience. This document summarizes these components and sample applications, and gives an example of how to use the examples to communicate with the Amazon Web Services (AWS) cloud with a smart phone app.

FreeRTOS Architecture

FreeRTOS Architecture

FreeRTOS is an open-source software project started by started by Richard Barry, now backed by Amazon, which provides a snapshot of the FreeRTOS Real-Time Operating System Project along with a standardized API for wireless connectivity (Bluetooth LE) and common MCU peripherals, such as SPI, UART, GPIO, Flash, and so on. Silicon Labs has ported FreeRTOS and these abstraction layers onto its Bluetooth LE and MCU SDKs and incorporated various configuration tools in Simplicity Studio for ease of use with the EFR32 series of parts.

The following is an architecture diagram of how FreeRTOS works with Silicon Labs software development kits (SDKs).



The following sections describe the various sub-components of FreeRTOS.

FreeRTOS

Silicon Labs uses FreeRTOS version 10.4.3 across the GSDK – both for FreeRTOS support and for other apps that use FreeRTOS.

Abstraction Layers

While there are multiple modules in FreeRTOS, the two relevant for Silicon Labs’ implementation are BLE HAL and Common IO. These are used to control the Bluetooth LE networking stack functionality and hardware peripherals, respectively, and are discussed in the following sections.

Common IO

FreeRTOS Common IO is an abstraction layer for common MCU hardware peripherals.

Instantiable peripheral. Common IO modules rarely can access different instances of a peripheral using an ID. Instead, a function to retrieve the instance descriptor must be provided. Simplicity Studio generates a standard file from a template file that provides the functionality for a specific module. A configuration .h file is generated for a specific peripheral instance with the default configuration values, including the peripheral ID to use when you call the Common IO module API. Another file is generated that appends the configurations for all instances in a table. This file also provides the function to return the instance descriptor based on the device ID. The following Common IO module are instantiable:

- adc
- flash
- flash
- gpio
- i2c
- pwm
- rtc
- spi
- timer
- uart
- Watchdog

Note: Simplicity Studio can suggest instances for your evaluation board and will provide a functional default pin configuration, but you may want to change the peripheral ID to a value more appropriate for your use case or application.

Non-instantiable peripheral. These may only require a configuration file. The following modules are not instantiable:

- power
- reset
- temp sensor
- efuse

BLE HAL

The GSDK implements the BLE HAL API as a translation layer on top of the Bluetooth Stack. The translation layer uses the Bluetooth Stack API to implement the functions of the BLE HAL API and maps events from the Bluetooth Stack to the corresponding callbacks in the BLE HAL API.

The Bluetooth Stack has its own configuration that must be set based on the requirements of the application on top of the BLE HAL API. For example, the configuration must enable enough advertisers and connections to match the application's needs.

When the BLE HAL is used, the GATT database is defined programmatically by the application. The application makes calls to the BLE HAL API to create the GATT database services, descriptors, and characteristics. The project must include the dynamic GATT database feature. This is included by default in all FreeRTOS sample applications but can also be configured in the Dynamic GATT database component. The GATT Configurator does not need to be used.

FreeRTOS Bluetooth LE Sample Apps

FreeRTOS Bluetooth LE Sample Apps

GSDK 4.1.2 includes FreeRTOS Bluetooth LE example applications:

- Amazon AWS – SoC MQTT over Bluetooth Low Energy demonstrates how to use the MQTT over the Bluetooth LE service.
- Amazon AWS – SoC Bluetooth GATT Server demonstrates how to use the FreeRTOS Bluetooth LE middleware APIs to create a simple GATT server.

The architecture of the system includes the embedded application, a smartphone, and the AWS services in the cloud. Both examples use a smartphone to ensure a communication channel with the cloud and serial communication (via Virtual COM port) to interact with the embedded application.

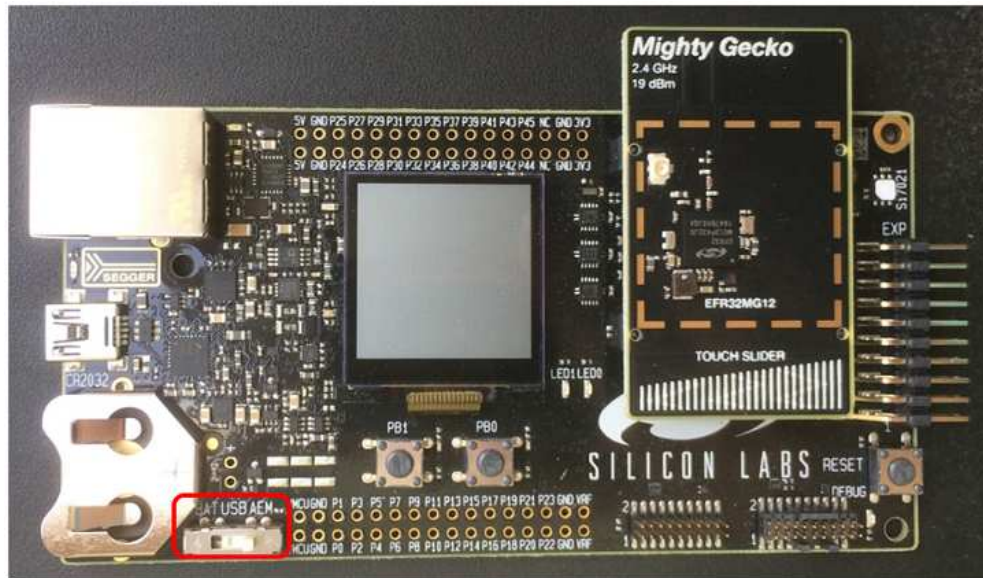
Prerequisites

To run an example system, the following items are needed:

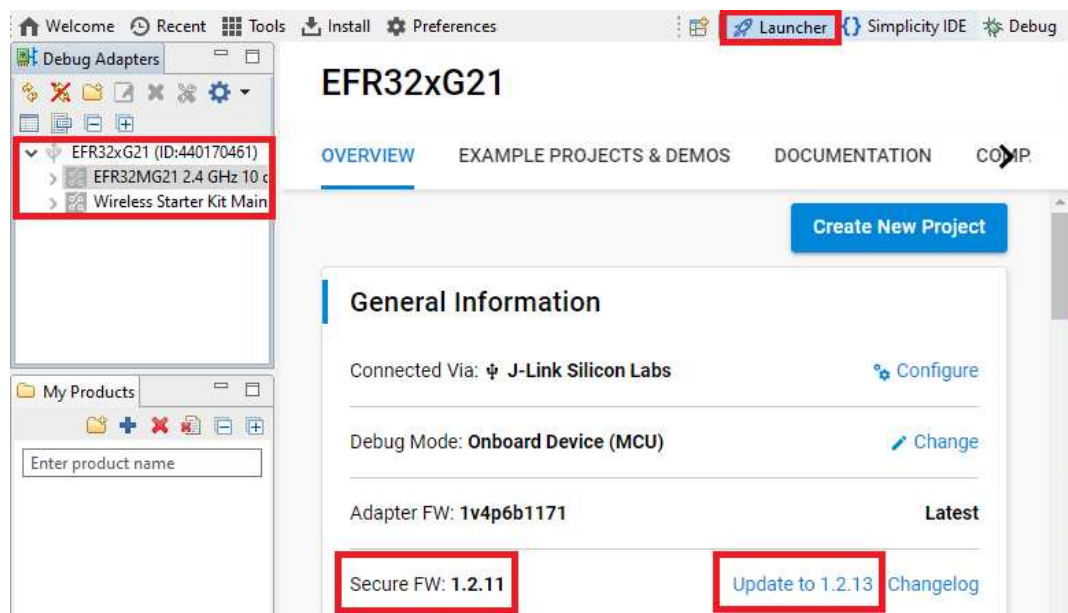
- One of the following kits:
- Bluetooth Starter Kit for EFR32BG21 (<https://www.silabs.com/development-tools/wireless/efr32xg21-bluetooth-starter-kit>) running Secure Engine running firmware version 1.2.13 or later
- Development kit for EFR32BG24 (<https://www.silabs.com/development-tools/wireless/efr32xg24-pro-kit-10-dbm?tab=overview>), running Secure Engine with firmware version 2.2.0 or later
- Smartphone with Android or iOS
- A computer running Windows, MacOS, or Linux with:
 - Simplicity Studio 5 installed (installers are available on <https://www.silabs.com/developer-tools/simplicity-software-development-kit?tab=downloads>)
 - GSDK 4.1.2 or later with the Bluetooth SDK installed through Simplicity Studio

Set Up the Silicon Labs Device and Software

1. Connect the starter kit mainboard, with radio board mounted, to your PC using the provided USB cable.
2. For best performance in Simplicity Studio, be sure that the power switch is in the Advanced Energy Monitoring or “AEM” position as shown in the following figure.



3. With your development kit connected, Install Simplicity Studio and the GSDK following the instructions in the Getting Started section of the [Simplicity Studio 5 User's Guide](#).
4. Once installation is complete, the development kit parts are displayed in the Simplicity Studio Debug Adapters view. Click the device to select it.
5. The General Information card on the Launcher Overview tab shows the device firmware and secure firmware versions, and whether updates are available. Silicon Labs recommends installing the latest version of both. For more information, see the About the Launcher > Welcome and Device Tabs section in the [Simplicity Studio 5 User's Guide](#).



Prepare the Cloud

Follow the instructions for setting up the cloud found on the Amazon Bluetooth LE demos site below.

<https://docs.aws.amazon.com/freertos/latest/userguide/ble-demo.html>

These instructions cover:

- Amazon “Thing” Setup

- Cognito Setup
- IAM Policies
- AWS IoT Policy

Amazon “Thing” Setup

Create an Amazon account (<https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/>) or log in with your existing one here: <https://aws.amazon.com/>.

Open the [AWS IoT Console](#) and create your Thing that is used to represent the device in the cloud (see the section To Set up AWS IoT at <https://docs.aws.amazon.com/freertos/latest/userguide/ble-demo.html>). Make a note of the name of the Thing.

For Bluetooth LE examples using Cognito, the thing should be created without a certificate.

Do not forget to go to AWS IoT Console > Settings menu > Device data endpoint and make a note of the device data endpoint and the AWS region that is the part of the endpoint. This endpoint describes the target of the MQTT connection in the cloud.

Cognito Setup

Cognito is required to use your mobile phone as a secure gateway to the cloud for the Bluetooth LE-enabled devices. You must create a User and an Identity pool to enable the application to connect to the cloud and manage devices.

Follow the steps in the section To create an Amazon Cognito user pool. Do not forget to make a note of the:

- User pool ID
- App client ID and app client secret (when creating Cognito User pool)
- Cognito Username and password
- Identity pool ID
- IAM roles for authenticated and unauthenticated identities to access Amazon Cognito

IAM Policies

Open the IAM console and follow the instructions in the section To create and attach an IAM policy to the authenticated identity.

AWS IoT Policy

Follow the instructions in the section FreeRTOS Bluetooth Low Energy Mobile SDK demo application for either Android or iOS SDK AWS IoT policy creation. This policy is different from the IAM policies above.

Summary of Notes

Check your notes to verify that the following information is available:

Note	Collected in	Used in	Example
Thing name	Thing creation	Embedded application	MyThing
AWS IoT REST Endpoint	Thing creation	Embedded application	<ALPHANUMERIC_STRING>-ats.iot.us-east-2.amazonaws.com
AWS Region	Thing creation	Mobile application	us-east-2
User pool ID	Cognito User pool creation	Cognito Identity pool creation	us-east-2_<ALPHANUMERIC_STRING>
		Mobile Application	
App client ID	Cognito User pool creation	Cognito Identity pool creation	<ALPHANUMERIC_STRING>
		Mobile application	
Cognito Username	Cognito User pool creation	Mobile application (runtime login)	demouser
Cognito Password	Cognito User pool creation	Mobile application (runtime login)	PASSWORD
App client secret	Cognito User pool creation	Mobile application	<ALPHANUMERIC_STRING>

Note	Collected in	Used in	Example
Identity pool ID	Cognito Identity pool creation	Mobile application	us-east-2: <ALPHANUMERIC_STRING>
IAM roles	Cognito Identity pool creation	IAM Policy setup	Cognito_DemoUnauth_Role Cognito_DemoAuth_Role

Create the Mobile Application

Android and iOS operating systems are supported by the mobile SDK from Amazon. The application can be found on GitHub at Android SDK for FreeRTOS Bluetooth Devices under [amazon-freertos-ble-android-sdk/app](https://github.com/aws/amazon-freertos-ble-android-sdk/app) and the iOS SDK for FreeRTOS Bluetooth Devices under [amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo](https://github.com/aws/amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo).

<https://github.com/aws/amazon-freertos-ble-android-sdk>

<https://github.com/aws/amazon-freertos-ble-ios-sdk>

Follow the Bluetooth LE demo site instructions to install the IDE for the selected OS and to download the SDK.

- To import the application, select the root folder for the unpacked application.
- Use your notes above to complete the configuration settings according to your account and cloud settings.

Using the IDE and the SDK, you can build the application and run it on your phone.

Create the Embedded Application

If you are not familiar with developing applications in Simplicity Studio 5, you can find more background in both the [Simplicity Studio 5 User's Guide](#) and the [Bluetooth SDK Getting Started Guide](#).

Create the Project

To create the embedded application project, you should have connected the starter kit mainboard to your PC, installed Simplicity Studio and the GSDK, and updated device firmware, as described in section [Set Up the Silicon Labs Device and Software](#). Make sure that the target device is selected in the Debug Adapters view.

Select File > New... > Silicon Labs Project Wizard... On the first page verify that your connected part is selected. If not, select it (for example, BRD4181A) and click NEXT.

New Project Wizard

Target, SDK, and Toolchain Selection

Select the target board, device, SDK, and IDE/toolchain to use for the project.

Target, SDK | Examples | Configuration

Target Boards:

Search or Select

Wireless Starter Kit Mainboard (BRD4001A Rev A01) × EFR32xG21 2.4 GHz 10 dBm Radio Board (BRD4181A) ×


Target Device:

Search or Select

EFR32MG21A010F1024IM32

SDK:

Select SDK
Gecko SDK Suite: Amazon 202012.00, Bluetooth 4.0.0, Bluetooth ESL 0.9.0.0, Bluetooth Mesh 3.0.0, EmberZNet 7.1.0.0, Find My 0.9.0

 [Manage SDKs](#)

IDE / Toolchain:

Select IDE / Toolchain
Simplicity IDE / GNU ARM v10.3.1

[CANCEL](#) [BACK](#) [NEXT](#) [FINISH](#)

On the next page, under Technology Type, select Amazon to filter examples. Select the Amazon AWS - SoC MQTT over Bluetooth Low Energy example and click NEXT.

4 resources found

Amazon Freertos Ble Hal Test

This test application is used for integration smoke testing to verify the functionality of Amazon FreeRTOS Bluetooth API translation layer

Amazon AWS - SoC MQTT over Bluetooth Low Energy

This application demonstrates how to use the MQTT over Bluetooth Low Energy service.

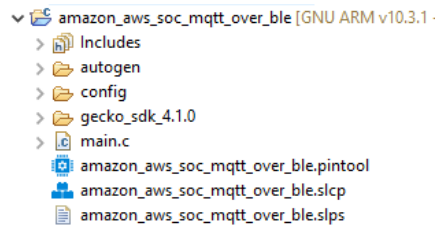
Amazon AWS - SoC Bluetooth Tests

Project to run AWS Tests including BLE tests on Silicon Labs boards.

Amazon AWS - SoC Bluetooth GATT Server

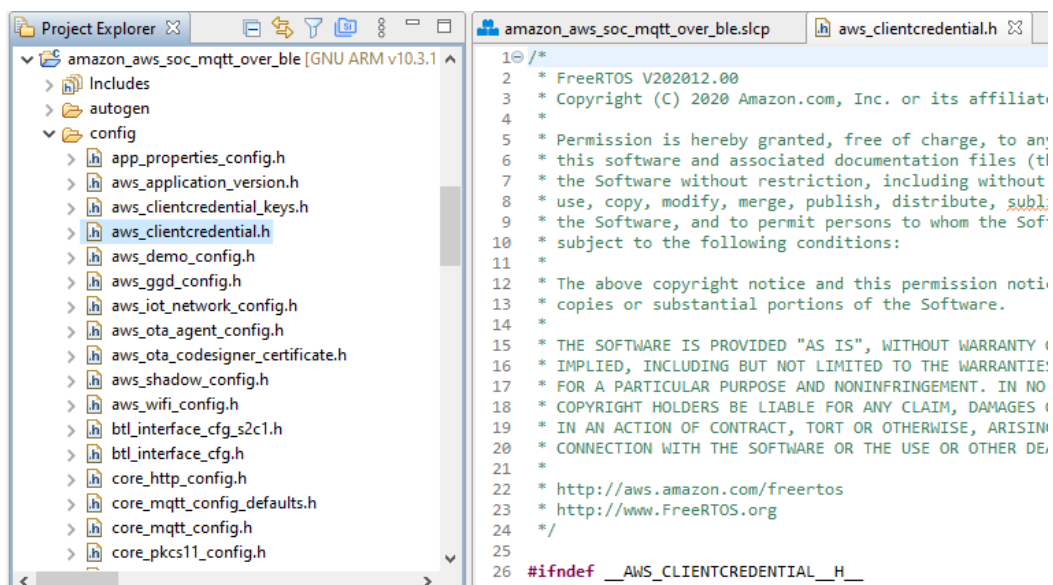
This application demonstrates how to use the FreeRTOS Bluetooth Low Energy middleware APIs to create a simple GATT server.

On the last page, optionally edit the project name and location. The linkage type of the SDK and the project sources can also be changed here. Click FINISH to open the Simplicity IDE. The project is created to the given location as shown in the Project Explorer view.



Configure the Project

To configure the application, in the Project Explorer view find and open `aws_clientcredential.h`. The file can be found in `<project_dir>/config/`:



Edit the macros:

- Copy and paste the Thing name to `clientcredentialIOT_THING_NAME`
- Copy and paste the AWS IoT REST Endpoint to `clientcredentialMQTT_BROKER_ENDPOINT`

Example:

```
#define clientcredentialMQTT_BROKER_ENDPOINT " <ALPHANUMERIC_STRING>-ats.iot.us-east-2.amazonaws.com"

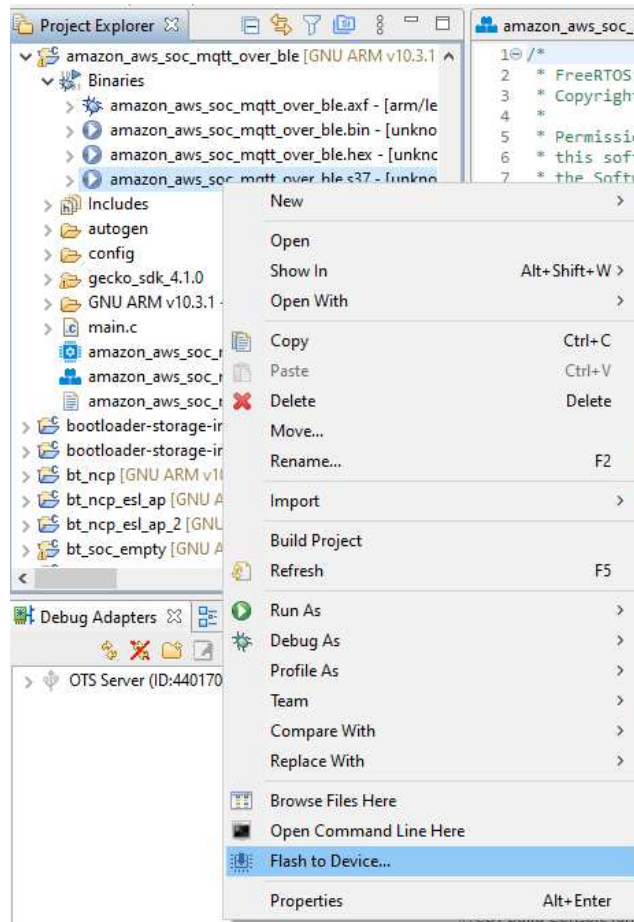
#define clientcredentialIOT_THING_NAME "MyThing"
```

Build and Flash the Application

To build the example application, click the build (hammer) icon on the toolbar. This creates a file `<project name>.s37`, which you can then flash to the device.

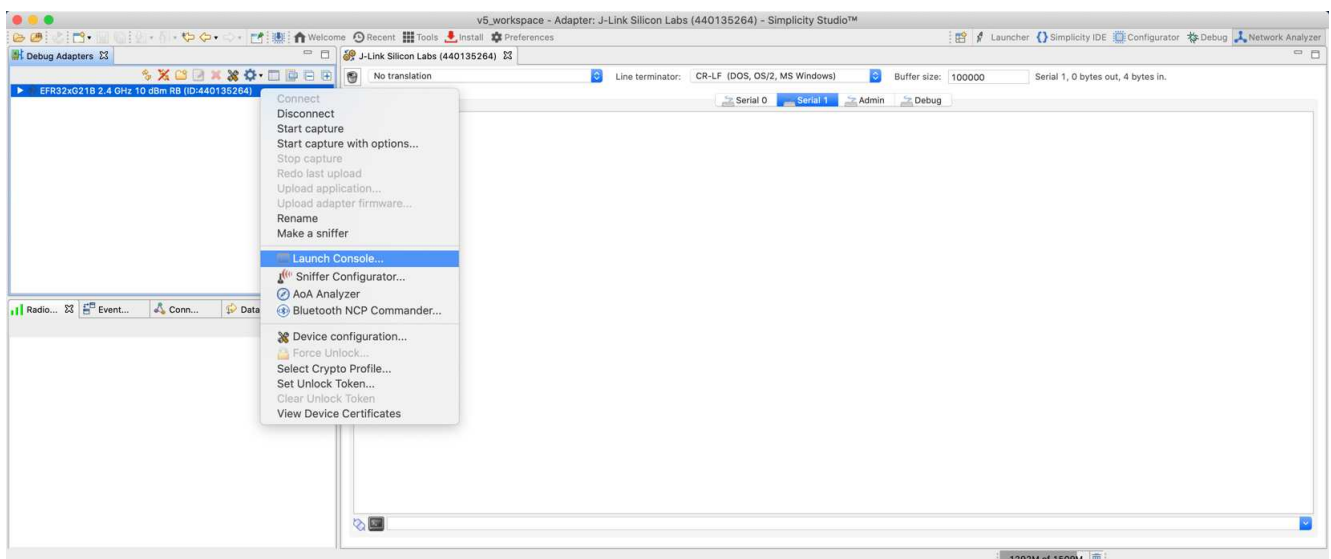
Note that, in order for this application to run, your device will also need a bootloader. An easy way to load a bootloader is to load any of the precompiled demo images, which come with the bootloader configured as part of the image. When you flash your application it overwrites the demo application, but the bootloader remains. For more information and detailed instructions, see [Using the Gecko Bootloader with Silicon Labs Bluetooth Applications](#).

With the target board connected to Simplicity Studio over direct USB connection or ethernet, you can then flash the binary you just built to the device by right-clicking the .s37 file, clicking “Flash to Device...”, and then selecting your device.

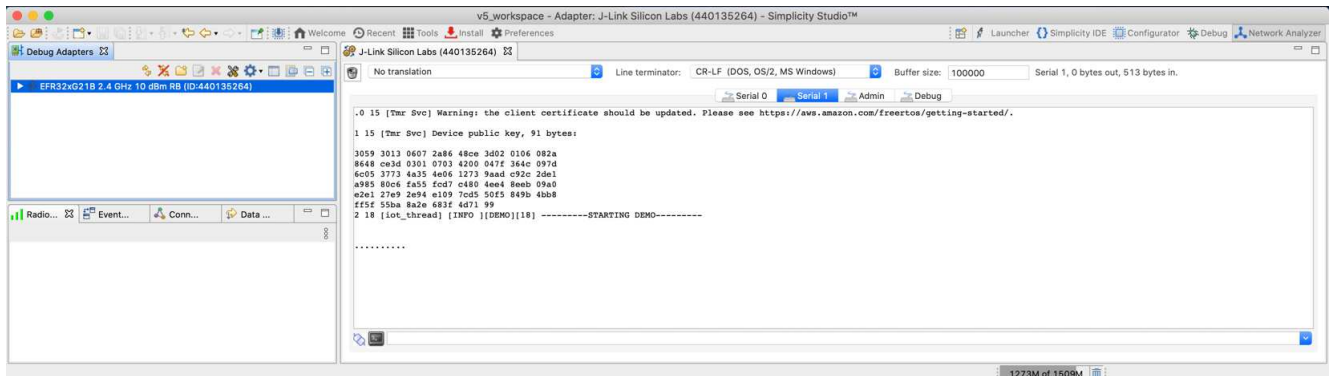


Using the Demo

Open a serial console with the device, by right-clicking the device in the Debug Adapters view and selecting Launch Console. In the console interface, select the “Serial 1” tab.



If you press the “reset” button on your board, the program will begin executing from the beginning. You should see the demo application start up with a “STARTING DEMO” message. Every few seconds, the device should print out a period (“.”) to show it is still running.

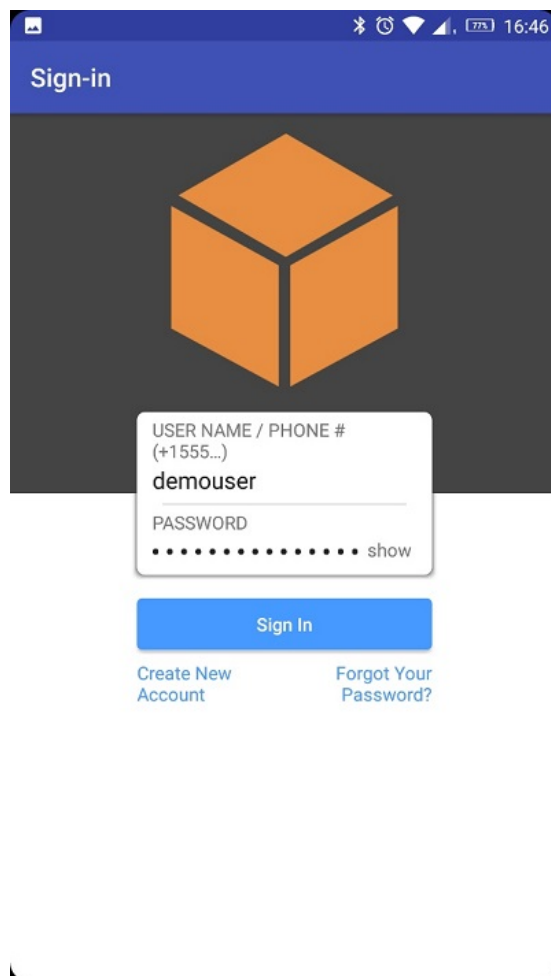


Log in to the AWS IoT Console and select the Test option to view the MQTT Test client.

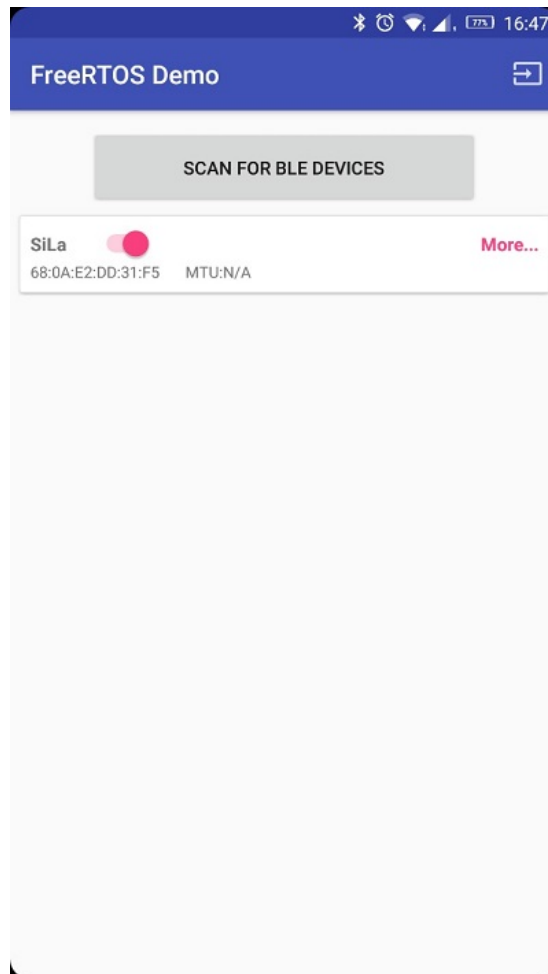
On the Subscribe panel subscribe to the following topics:

- thing-name/example/topic1
- thing-name/example/topic2

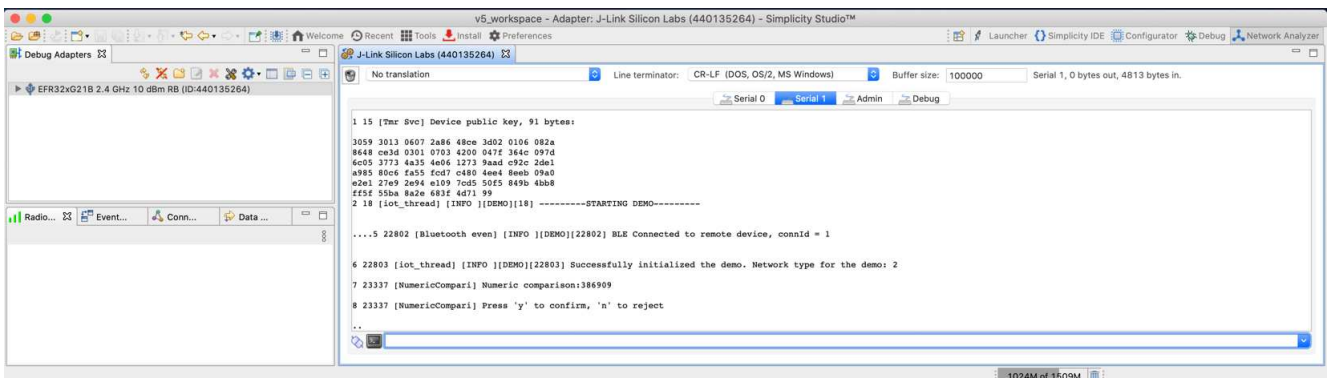
Open the mobile application and log in with your Cognito username and password.



After logging in, tap the SCAN FOR BLE DEVICES control and find the board in the list. Slide the toggle and wait for the pairing request with the Numeric Comparison method on both devices (mobile, embedded).



If the numbers are equal on both sides, accept the requests. On the embedded side, it can be done by sending an 'y' character.



After the pairing is finished and the subscription is complete, the embedded application starts to publish to the topics and this can be seen on the MQTT Test Client site.

FreeRTOS Aws Test Sample For Bluetooth LE

FreeRTOS AWS Test Sample for Bluetooth LE

The Amazon AWS - Peripheral Tests example application demonstrates that the MCU peripheral components are working as expected on your design.

The Amazon AWS – SoC Bluetooth Tests application can be used with the AWS IoT Device Tester (IDT) for FreeRTOS to verify that the operating system and the libraries are working locally on your device and can communicate with the AWS IoT Cloud. It verifies the porting layers, interfaces and board device drivers and the libraries on the top of them to ensure connectivity.

The IDT generates test reports that can be used to participate in the AWS Device Qualification Program.

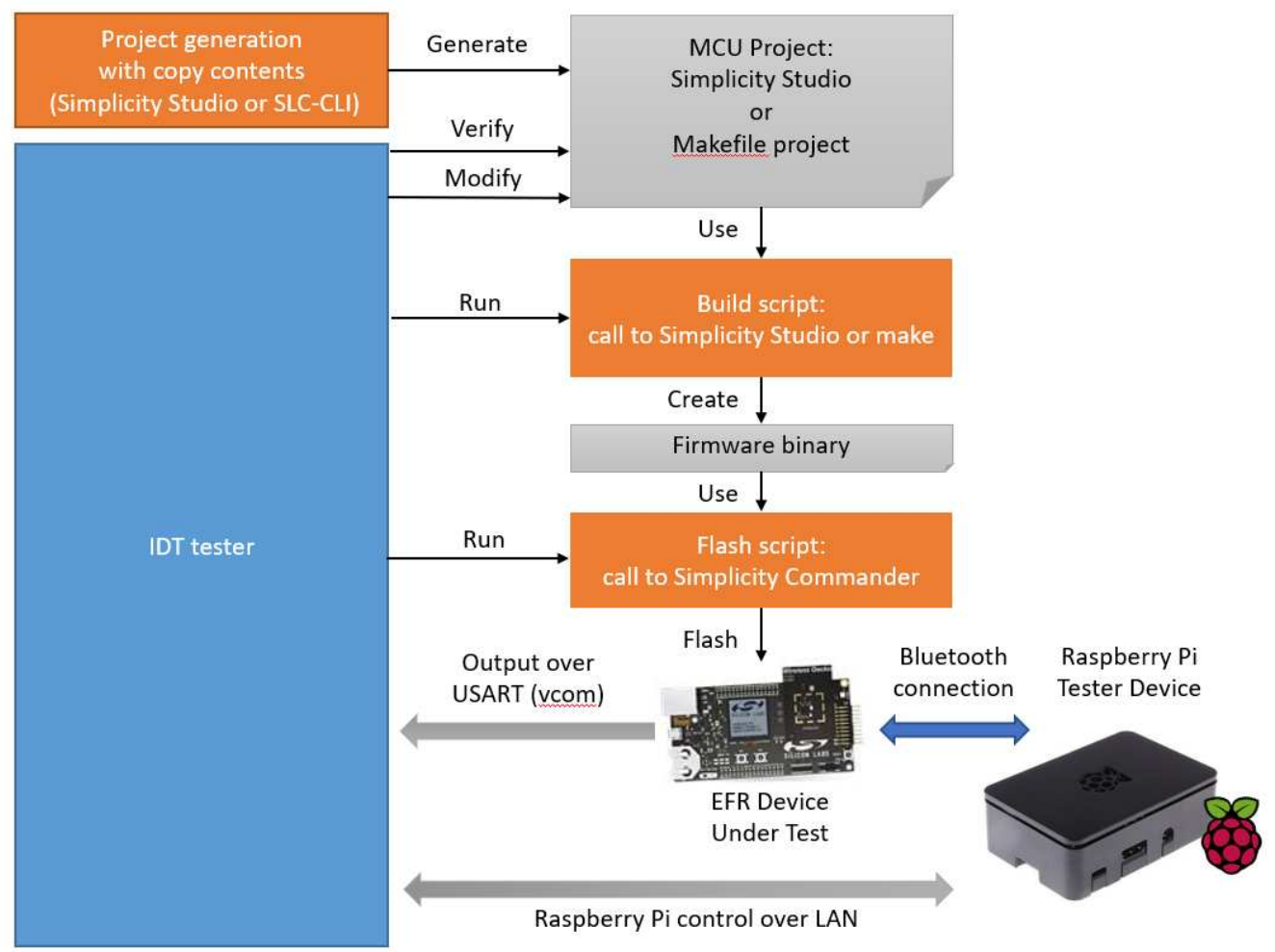
The test environment for Bluetooth LE consists of:

- A PC running the IDT application
- A Raspberry Pi running the testing software image
- The device/board under test running the Amazon AWS – SoC Bluetooth Tests example

The IDT testing workflow is illustrated in the following figure. In summary, the workflow starts with generating an MCU project to serve as the embedded application. Simplicity Studio or SLC-CLI can be used to generate the Amazon AWS – SoC Bluetooth Tests project.

When the command line IDT tester application starts, it connects to the Raspberry Pi tester device. It then verifies the project source code and modifies it by adding a generated UUID to make sure that the current setup is running on the MCU. Therefore, IDT testing requires user-defined scripts that IDT can call to build the generated MCU project and flash the firmware onto the Device Under Test.

After the flashing step, the IDT tester application starts the Bluetooth tests by reading the output of the Device Under Test coming over USART over USB VCOM and by instructing the Raspberry Pi tester device over LAN. Note that the latter function uses scripts that were previously generated (copied) into the project by SLC-CLI or Simplicity Studio.



The following sections provide detailed instructions for these steps.

Prerequisites

The following is needed to run Bluetooth LE tests:

- Development kit for EFR32xG21 (for example, BRD4181A) and a Wireless Starter Kit (WSTK)
- A Raspberry Pi 4B or 3B+ with power supply
- Wireless LAN or LAN Router and an ethernet cable
- A micro-SD card and SD card adapter for the Raspberry Pi software
- A PC with
 - Simplicity Studio 5 (installers are available on <https://www.silabs.com/developer-tools/simplicity-software-development-kit?tab=downloads>) and/or SLC-CLI (for Linux and Mac)
 - GSDK 4.1.2 or later
 - Simplicity Commander (if using SLC-CLI for project generation)

Preferred tools for project generation, building and flashing for different Operating Systems are shown in the following table.

	Windows	Linux	MacOS
Project Generation	Simplicity Studio	Simplicity Studio or SLC-CLI	Simplicity Studio or SLC-CLI
Project Type	Simplicity Studio project	Makefile project	Makefile project
Build tool	Simplicity Studio headless build	make	make
Flash Tool	Simplicity Commander	Simplicity Commander	Simplicity Commander

Creating the Embedded Application

The application can be created using SLC-CLI or Simplicity Studio.

To create the project using Simplicity Studio, follow the procedures in section [Create the Embedded Application](#) to create the embedded application. Use the Amazon AWS – SoC Bluetooth Tests example. During project creation, it is important to select Copy Contents in the Configuration step since IDT will modify project sources.

To create a project using SLC-CLI, see the [SLC-CLI User Guide](#). An example of the project generation call is:

```
slc generate -p="app/amazon/example/amazon_aws_tests/amazon_aws_soc_bt_tests.slc" -o="makefile" gcc -cp
-d="/path-to-workspace/amazon_aws_soc_bt_tests" --with="brd4181a"
```

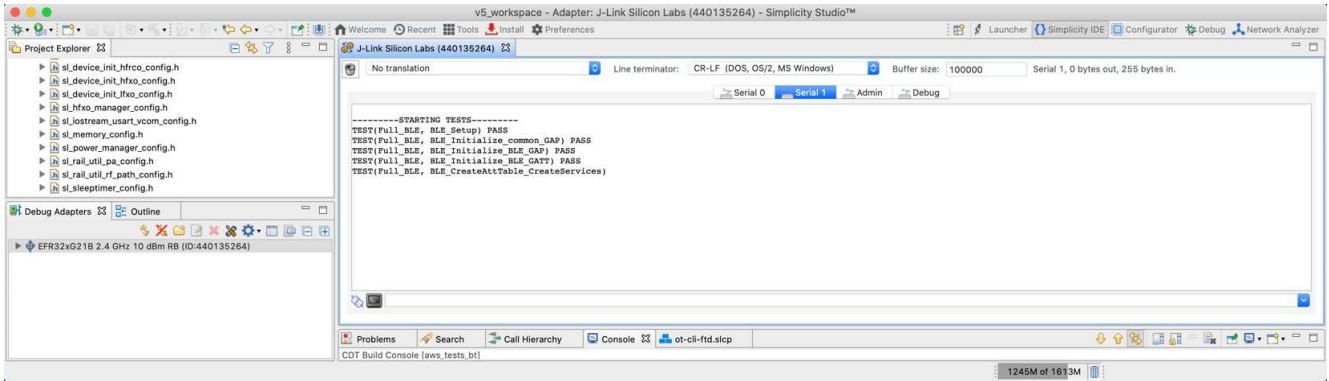
The example has been configured to run Full Bluetooth LE tests, which can be verified in the *config/aws_test_runner_config.h* config file located in the project folder.

```
/* Universal Configurator will enable the required tests during generation. */
#define testrunnerFULL_WIFI_ENABLED 0
#define testrunnerFULL_TASKPOOL_ENABLED 0
#define testrunnerFULL_WIFI_PROVISIONING_ENABLED 0
#define testrunnerFULL_TCP_ENABLED 0
#define testrunnerFULL_GGD_ENABLED 0
#define testrunnerFULL_GGD_HELPER_ENABLED 0
#define testrunnerFULL_SHADOW_ENABLED 0
#define testrunnerFULL_SHADOWV4_ENABLED 0
#define testrunnerFULL_MQTTv4_ENABLED 0
#define testrunnerFULL_MQTT_STRESS_TEST_ENABLED 0
#define testrunnerFULL_MQTT_AGENT_ENABLED 0
#define testrunnerFULL_MQTT_ALPN_ENABLED 0
#define testrunnerFULL_OTA_CBOR_ENABLED 0
#define testrunnerFULL_OTA_AGENT_ENABLED 0
#define testrunnerFULL_OTA_PAL_ENABLED 0
#define testrunnerFULL_PKCS11_ENABLED 0
#define testrunnerFULL_PKCS11_MODEL_ENABLED 0
#define testrunnerFULL_CRYPT0_ENABLED 0
#define testrunnerFULL_TLS_ENABLED 0
#define testrunnerFULL_DEFENDER_ENABLED 0
#define testrunnerFULL_POSIX_ENABLED 0
#define testrunnerUTIL_PLATFORM_CLOCK_ENABLED 0
#define testrunnerUTIL_PLATFORM_THREADS_ENABLED 0
#define testrunnerFULL_BLE_ENABLED 1
#define testrunnerFULL_BLE_STRESS_TEST_ENABLED 0
#define testrunnerFULL_BLE_KPI_TEST_ENABLED 0
#define testrunnerFULL_BLE_INTEGRATION_TEST_ENABLED 0
#define testrunnerFULL_BLE_END_TO_END_TEST_ENABLED 0
#define testrunnerFULL_FREERTOS_TCP_ENABLED 0
#define testrunnerFULL_SERIALIZER_ENABLED 0
#define testrunnerFULL_HTTPS_CLIENT_ENABLED 0
#define testrunnerFULL_COMMON_IO_ENABLED 0
#define testrunnerFULL_CORE_MQTT_ENABLED 0
#define testrunnerFULL_CORE_MQTT_AWS_IOT_ENABLED 0
#define testrunnerFULL_CORE_HTTP_ENABLED 0
#define testrunnerFULL_CORE_HTTP_AWS_IOT_ENABLED 0
#define testrunnerFULL_CLI_ENABLED 0
#define testrunnerFULL_DEVICE_SHADOW_ENABLED 0
```

Verify the Embedded Application

To verify the generated project before running IDT, build and flash the firmware to your device and open a serial terminal. Building can be done using Simplicity Studio or make, depending on the project type. Flashing can be done using Simplicity Commander. More information about Simplicity commander can be found in the [Simplicity Commander Reference Guide](#).

Verify that the application starts the tests after 5 seconds.



Disconnect the serial monitor by closing the port to let IDT run. Make sure to close Simplicity Studio before running IDT testing.

Prepare the Raspberry Pi

Follow the instructions for setting up the Raspberry Pi found on the following Amazon site.

<https://docs.aws.amazon.com/freertos/latest/userguide/afr-bridgekeeper-dt-bt.html>

During the installation, make a note of the login settings for the Raspberry Pi:

- IP address
- Username
- Password or the private key file for the Public Key authentication

These are needed during the following steps for the IDT setup.

Prepare the IDT

Download the software package containing IDT v4.0.3 and test suite version 1.5.1 for FreeRTOS 202012.00 (uses FreeRTOS 202012.00 LTS libraries) from here:

<https://docs.aws.amazon.com/freertos/latest/userguide/dev-test-versions-afr.html>

Follow the instructions here:

<https://docs.aws.amazon.com/freertos/latest/userguide/qual-steps.html>

Besides the steps above, this provides additional information and examples for the configuration.

An example for device.json file follows. Modify the serialPort to correspond to your system.

```
[
  {
    "id": "ble-test-raspberry-pi",
    "sku": "brd4181b",
    "features": [
      {
        "name": "BLE",
        "value": "Yes"
      },
      {
        "name": "Cellular",
        "value": "No"
      },
      {
        "name": "WIFI",
        "value": "No"
      },
      {
        "name": "OTA",
        "value": "No"
      },
      {
        "name": "TCP/IP",
        "value": "No"
      },
      {
        "name": "TLS",
        "value": "No"
      },
      {
        "name": "PKCS11",
        "value": "ecc"
      },
      {
```

```
"name": "KeyProvisioning", "value": "Onboard"}], "devices": [{"id": "brd4181b", "connectivity":
{"protocol": "uart", "serialPort": "COM4"}}]}
```

An example for the userdata.json file follows. Change the “/absolute-path-to/” to the locations on your system.

```
{
  "sourcePath": "/absolute-path-to/amazon_aws_soc_bt_tests/gecko_sdk_4.1.0/util/third_party/aws_iot_libs",
  "vendorPath": "/absolute-path-to/amazon_aws_soc_bt_tests",
  "buildTool": {
    "name": "build_script",
    "version": "4.1",
    "command": [
      "/absolute-path-to/buildTool"
    ]
  },
  "flashTool": {
    "name": "flash_script",
    "version": "4.1",
    "command": [
      "/absolute-path-to/flashTool"
    ]
  },
  "buildImageInfo": {
    "testsImageName": "tests.img",
    "demosImageName": "demos.img"
  }
}
```

The sourcePath is an absolute path pointing to the FreeRTOS libraries in the generated Amazon AWS – SoC Bluetooth Tests project. Project name and SDK folder inside the project may vary. The default project name is amazon_aws_soc_bt_tests.

The vendorPath is the location of the generated Amazon AWS – SoC Bluetooth Tests project.

The buildTool and flashTool are user-defined scripts that can be used by IDT to build and flash the previously generated and modified project.

The scripts can be batch files for Windows (build.bat, flash.bat) and shell scripts for Linux/MacOS (build.sh, flash.sh).

The buildTool can call either Simplicity Studio headless build (on Windows) or make (on Linux/MacOS). The flashTool can call Simplicity Commander with command line options. More information about Simplicity commander can be found in the [Simplicity Commander Reference Guide](#).

Templates for Build and Flash Tools

Example of the buildTool (build.bat) for Windows:

```
@echo off

echo Building project...

set STUDIO_PATH="<path_to_simplicity_studio>"

set PROJECT_NAME="<project_name>"

start /B /wait %STUDIO_PATH:="%\studio.exe -nosplash -application org.eclipse.cdt.managedbuilder.core.headlessbuild -
printErrorMarkers -no-indexer -cleanBuild %PROJECT_NAME%

exit 0
```

Example of the flashTool script (flash.bat) script for Windows:

```
@echo off

echo Flashing project...

set STUDIO_PATH="<path_to_simplicity_studio>"

set PROJECT_BINARY="<path_to_firmware_binary>"

start /B /wait %STUDIO_PATH:="%\developer\adapter_packs\commander\commander.exe flash %PROJECT_BINARY% --address 0x0

start /B /wait %STUDIO_PATH:="%\developer\adapter_packs\commander\commander.exe device reset
```

Example of the buildTool (build.sh) for Linux / MacOS:

```
#!/bin/bash

echo Building project...

export ARM_GCC_DIR=<path_to_arm_gcc_folder>

make -C <path_to_project_directory> -f <project_name>.Makefile
```

Example of the flashTool (flash.sh) for Linux / MacOS:

```
#!/bin/bash

echo Flashing project...

COMMANDER_PATH="<path_to_simplicity_commander>"

PROJECT_BINARY="<path_to_firmware_binary>"

$COMMANDER_PATH flash $PROJECT_BINARY --address 0x0

$COMMANDER_PATH device reset
```

Run IDT Testing for Bluetooth LE

To run the IDT test, first power up and log in to the Raspberry Pi, which was configured in [Prepare the Raspberry Pi](#). After login, connect to your Wi-Fi network (example: ifup wlan0).

On Windows, make sure that Simplicity Studio is closed before running IDT tester tool.

In the directory where the IDT tester tool is downloaded, go to the bin folder, and open a command line.

Start the test with the following command (modified as appropriate):

```
devicetester_[linux|mac|win]_x86-64 run-suite --group-id FullBLE --userdata userdata.json
```

The tests can take up to 7-10 minutes. The results are printed continuously to the “results/<latest-test-results-id>/logs/FullBLE__Full_BLE.txt” file.