Chapter 6: Amazon FreeRTOS & Reference Flow

After completing this chapter, you will understand Amazon FreeRTOS and how to build IoT projects using Cypress Wi-Fi, Bluetooth, and PSoC 6.

6.1 AMAZON FREERTOS TOUR 2
6.1.1 CONNECTIVITY OVERVIEW 3
6.1.2 FIRMWARE OVERVIEW 3
6.1.3 AFR WEBSITE 4
6.1.4 FREERTOS WEBSITE 5
6.1.5 CYPRESS WEBSITE 6
6.1.6 CYPRESS GITHUB 7
6.1.7 CYPRESS COMMUNITY 8
6.1.8 DOCUMENTATION 9
6.2 DEMO WALKTHROUGH 15
6.3 EXERCISES (PART 1) 26
6.3.1 RUN THE MQTT DEMO 26
6.3.2 MODIFY THE PROJECT TO PUBLISH TO A DIFFERENT TOPIC 26
6.4 AWS FREERTOS DIRECTORY ORGANIZATION 27
6.4.1 TOP LEVEL ORGANIZATION 27
6.4.2 DEMOS 27
6.4.3 VENDORS/CYPRESS/ CY8CPROTO_062_4343W/AWS_DEMOS 30
6.4.4 PROJECTS 32
6.5 AWS DEMO NETWORK MANAGER 33
6.5.1 INTERFACE HEADERS 33
6.5.2 CONFIGURATION MACROS 34
6.5.3 NETWORK STATES 34
6.5.4 FUNCTIONS 34
6.5.5 NETWORK MANAGER - CAUTION 36
6.6 APPLICATION & DEMO FIRMWARE FLOW (SOURCE CODE) 36
6.6.1 STARTUP 36
6.6.2 DEMO_RUNNER 36
6.6.3 MQTT DEMO 38
6.6.4 SHADOW DEMO 38
6.6.5 YOUR APPLICATION 38
6.7 FREERTOS CONFIGURATION 39
6.7.1 CONFIGUSE_DAEMON_TASK_STARTUP_HOOK – VAPPLICATIONDAEMONTASKSTARTUPHOOK 39
6.7.2 CONFIGUSE_IDLE_HOOK – VAPPLICATIONIDLEHOOK 39
6.7.3 CONFIGUSE_TICK_HOOK – VAPPLICATIONTICKHOOK 39
6.7.4 CONFIGPRINTF – VLOGGINGPRINTF 40
6.7.5 CONFIGUSE_MALLOC_FAILED_HOOK – VAPPLICATIONMALLOCFAILEDHOOK 40
6.7.6 CONFIGCHECK_FOR_STACK_OVERFLOW – VAPPLICATIONSTACKOVERFLOWHOOK 41
6.8 FREERTOS NAMING CONVENTION(S) 41
6.8.1 VARIABLES 41
6.8.2 FUNCTION NAMES 41
6.8.3 MACROS 42
6.9 AMAZON FREERTOS LIBRARIES 42
6.9.1 MQTT 42
6.9.2 BLUETOOTH 43
6.9.3 LOGGING 44
6.9.4 AWS DEVICE SHADOW 46

6.10 USING THE CYPRESS HAL & PDL 47
6.11 BUILD SYSTEM 47
6.11.1 MAKE 47
6.11.2 FILES 47
6.11.3 CMAKE 48
6.12 COMING IN FUTURE RELEASES 48
6.13 EXERCISES (PART 2) 48
6.13.1 AWS IOT SETUP 48
6.13.2 AWS COGNITO SETUP 49
6.13.3 BUILDING THE IOS APP 52
6.13.4 BUILDING THE ANDROID APP 56
6.13.5 RUNNING MQTT OVER BLE DEMO 59
6.13.6 RUNNING BLE GATT SERVER DEMO 70
6.13.7 PRINT SCAN OF ALL THE WI-FI ACCESS POINTS 75
6.13.8 DISPLAY USING TFT 77
6.13.9 MODIFY THE MQTT DEMO WITH DISPLAY 77
6.13.10 SIMPLE MQTT OVER WI-FI 78
6.13.11 SIMPLE MQTT OVER BLE 81
6.13.12 USE TFT TO DISPLAY MESSAGES RECEIVED VIA MQTT SUBSCRIBE 82
6.13.13 RUN THE SHADOW DEMO 83
6.13.14 FIX THE LOGGING SYSTEM TO PRINT OUT THE TIME INSTEAD OF [LU] 87

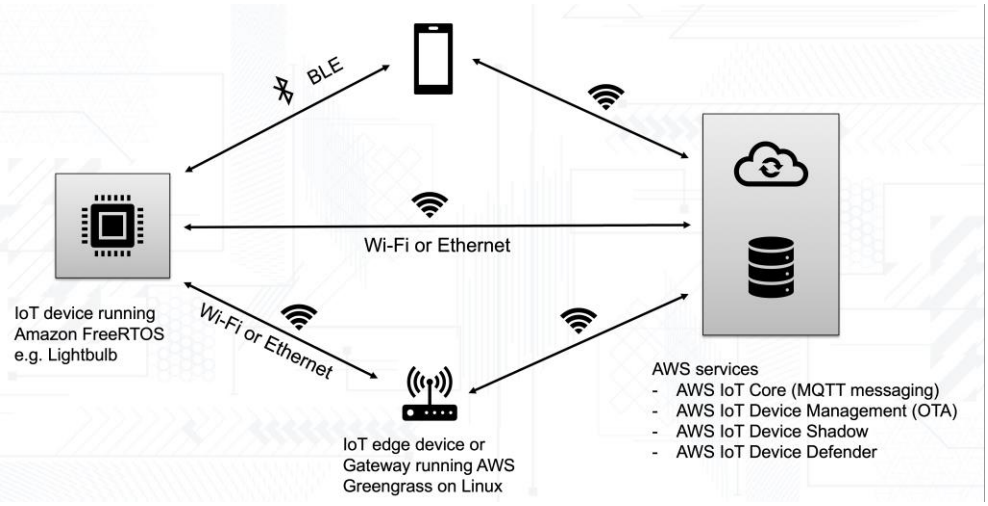
6.1 Amazon FreeRTOS Tour

Amazon FreeRTOS is a release of FreeRTOS that has been packaged (by Amazon) with a TCP/IP Stack, MBED TLS, and a complete set of IoT Middleware libraries, which can be used to connect to the Amazon Web Services IoT Core. Cypress extended this release by adding the PSoC 6 libraries (HAL, PDL) and the required drivers to use the Cypress PSoC 6 & 43xxx Wi-Fi Bluetooth combo radios.

FreeRTOS continues as a stand-alone open source product sponsored by Amazon, but distributed by FreeRTOS.org.

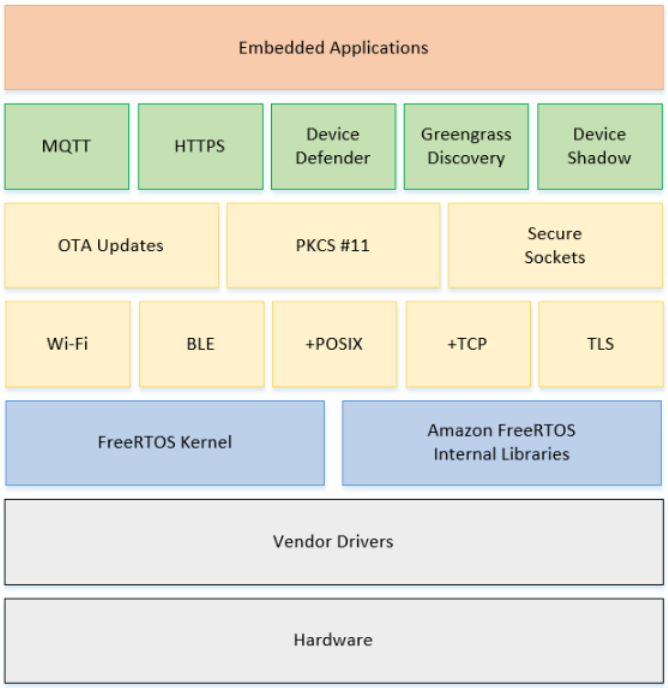
6.1.1 Connectivity Overview

AFR is built around the singular objective of implementing IoT systems that connect to the AWS cloud directly or via GreenGrass using Wi-Fi or Bluetooth.



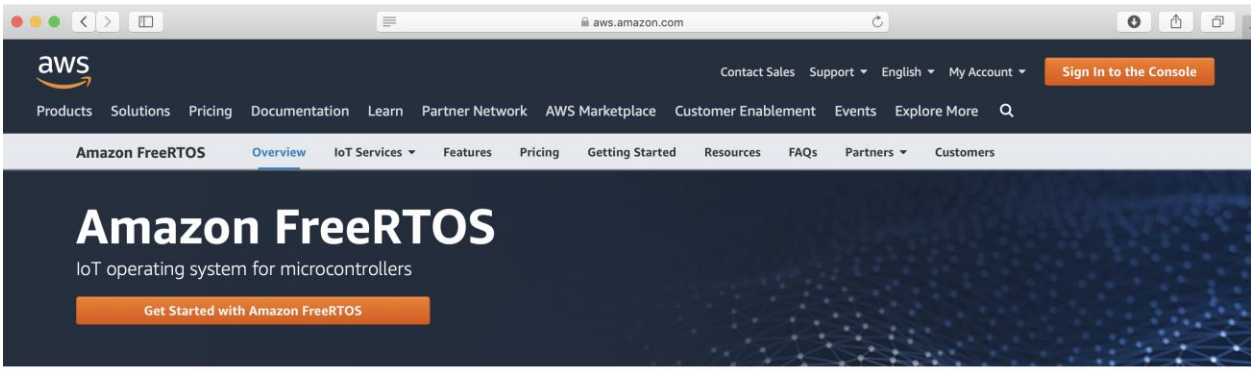
6.1.2 Firmware Overview

Amazon FreeRTOS includes a complete version of FreeRTOS, an MCU framework, and an IoT framework



6.1.3 AFR Website

https://aws.amazon.com/freertos/



Amazon FreeRTOS (a:FreeRTOS) is an open source operating system for microcontrollers that makes small, low-power edge devices easy to program, deploy, secure, connect, and manage. Amazon FreeRTOS extends the FreeRTOS kernel, a popular open source operating system for microcontrollers, with software libraries that make it easy to securely connect your small, low-power devices to AWS cloud services like AWS IoT Core or to more powerful edge devices running AWS IoT Greengrass.

A microcontroller (MCU) is a single chip containing a simple processor that can be found in many devices, including appliances, sensors, fitness trackers, industrial automation, and automobiles. Many of these small devices could benefit from connecting to the cloud or locally to other devices. For example, smart electricity meters need to connect to the cloud to report on usage, and building security systems need to communicate locally so that a door will unlock when you badge in. Microcontrollers have limited compute power and memory capacity and typically perform simple, functional tasks. Microcontrollers frequently run operating systems that do not have built-in functionality to connect to local networks or the cloud, making IoT applications a challenge. Amazon FreeRTOS helps solve this problem by providing both the core operating system (to run the edge device) as well as software libraries that make it easy to securely connect to the cloud (or other edge devices) so you can collect data from them for IoT applications and take action.

To get started, you can select a device from the AWS Partner Device Catalog. Then, you can use the Amazon FreeRTOS console to download Amazon FreeRTOS for your device or download from GitHub. Amazon FreeRTOS is open source and there is no charge to use it. Visit our getting started page to learn more about Amazon FreeRTOS.

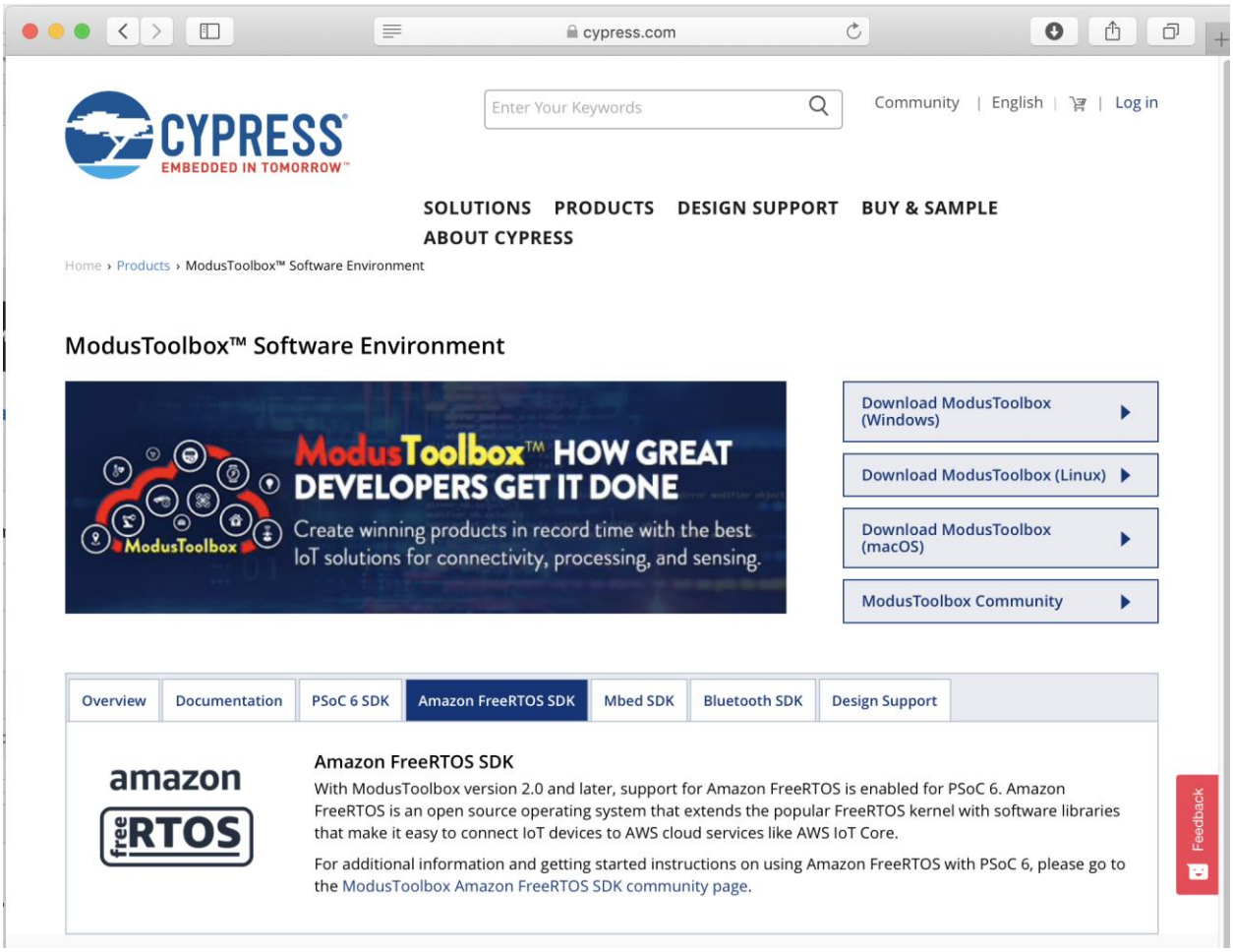
What is Amazon FreeRTOS? (1:38)

6.1.4 FreeRTOS Website

https://freertos.org

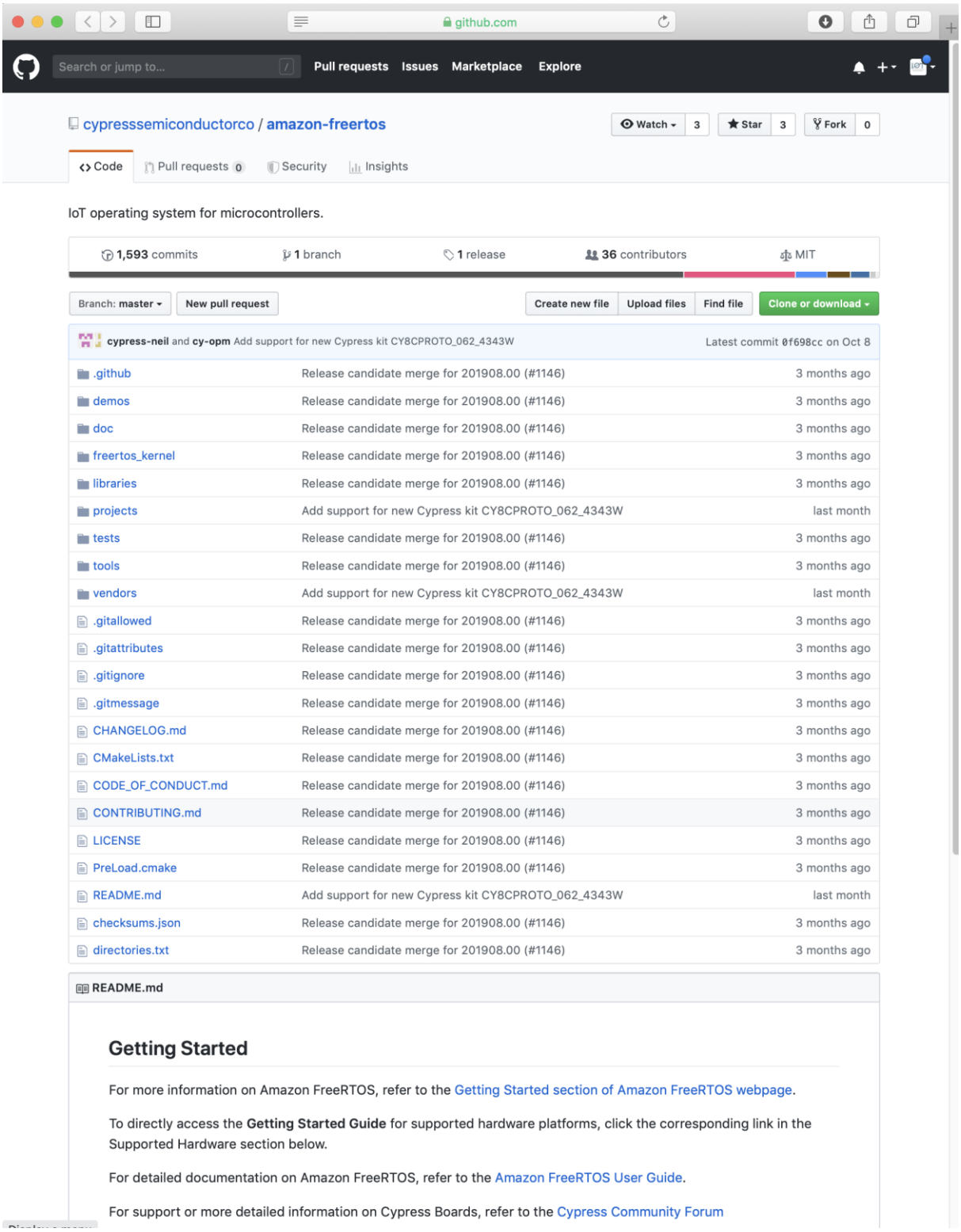
6.1.5 Cypress Website

https://www.cypress.com/products/modustoolbox-software-environment



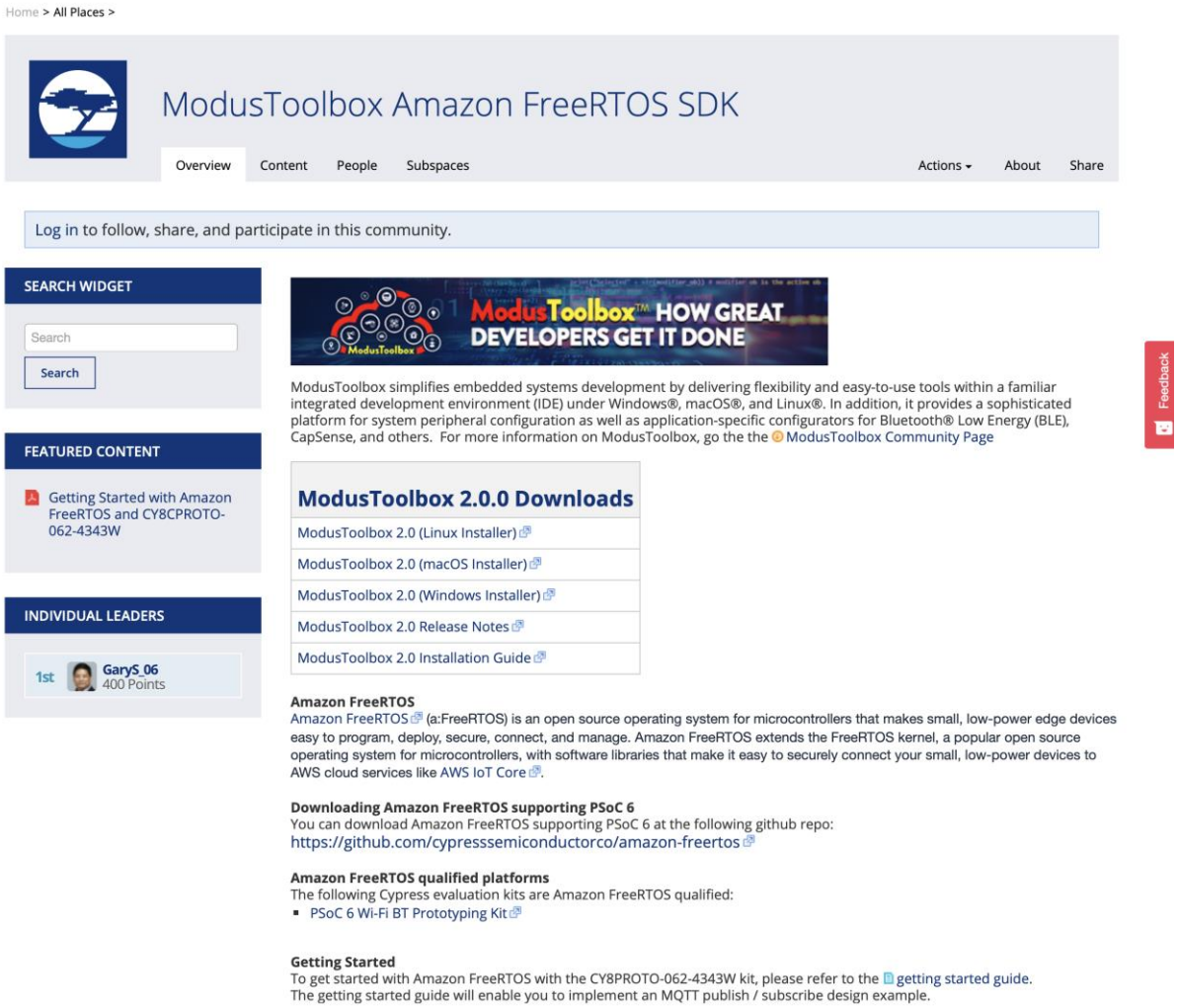
6.1.6 Cypress GitHub

https://github.com/cypresssemiconductorco/amazon-freertos



6.1.7 Cypress Community

https://community.cypress.com/community/modustoolbox-amazon-freertos-sdk



6.1.8 Documentation

Cypress Getting Started with Amazon FreeRTOS https://community.cypress.com/docs/DOC-18348

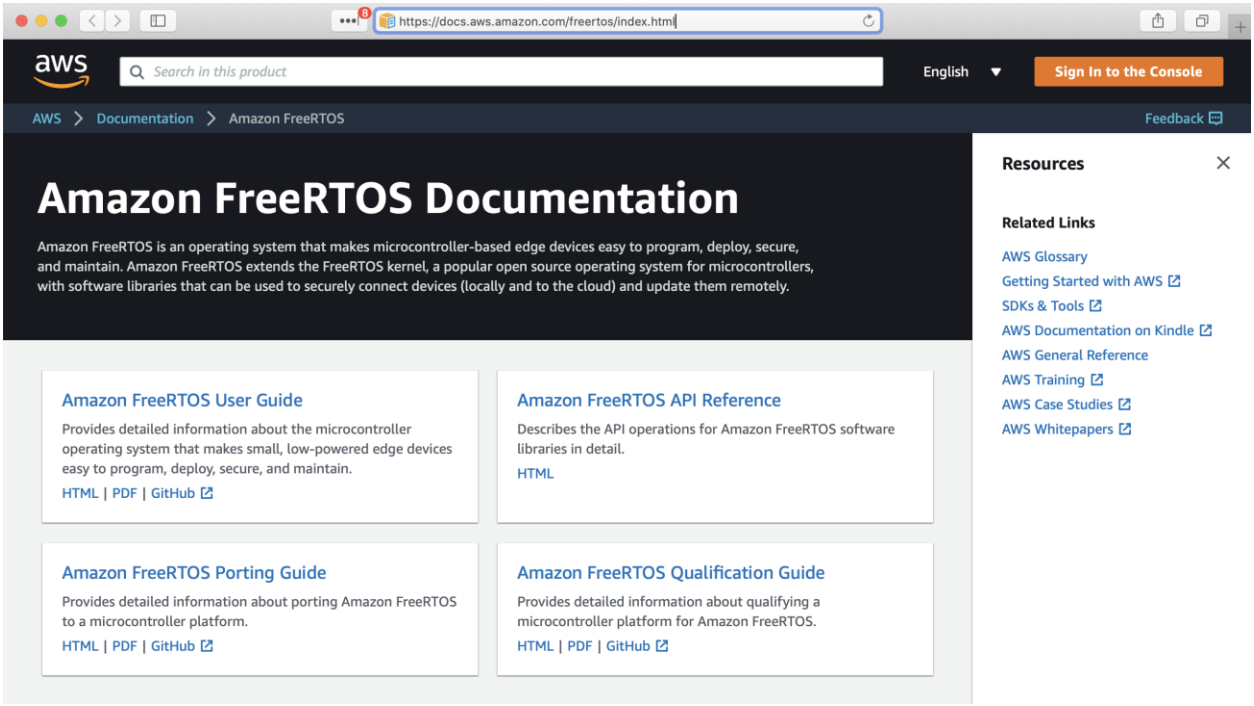
Getting Started with the CY8CPROTO-062-4343W

This document provides instructions for getting started with the CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit. If you do not have the CY8CPROTO-062-4343W Kit, visit the AWS Partner Device Catalog to purchase one from our partner.

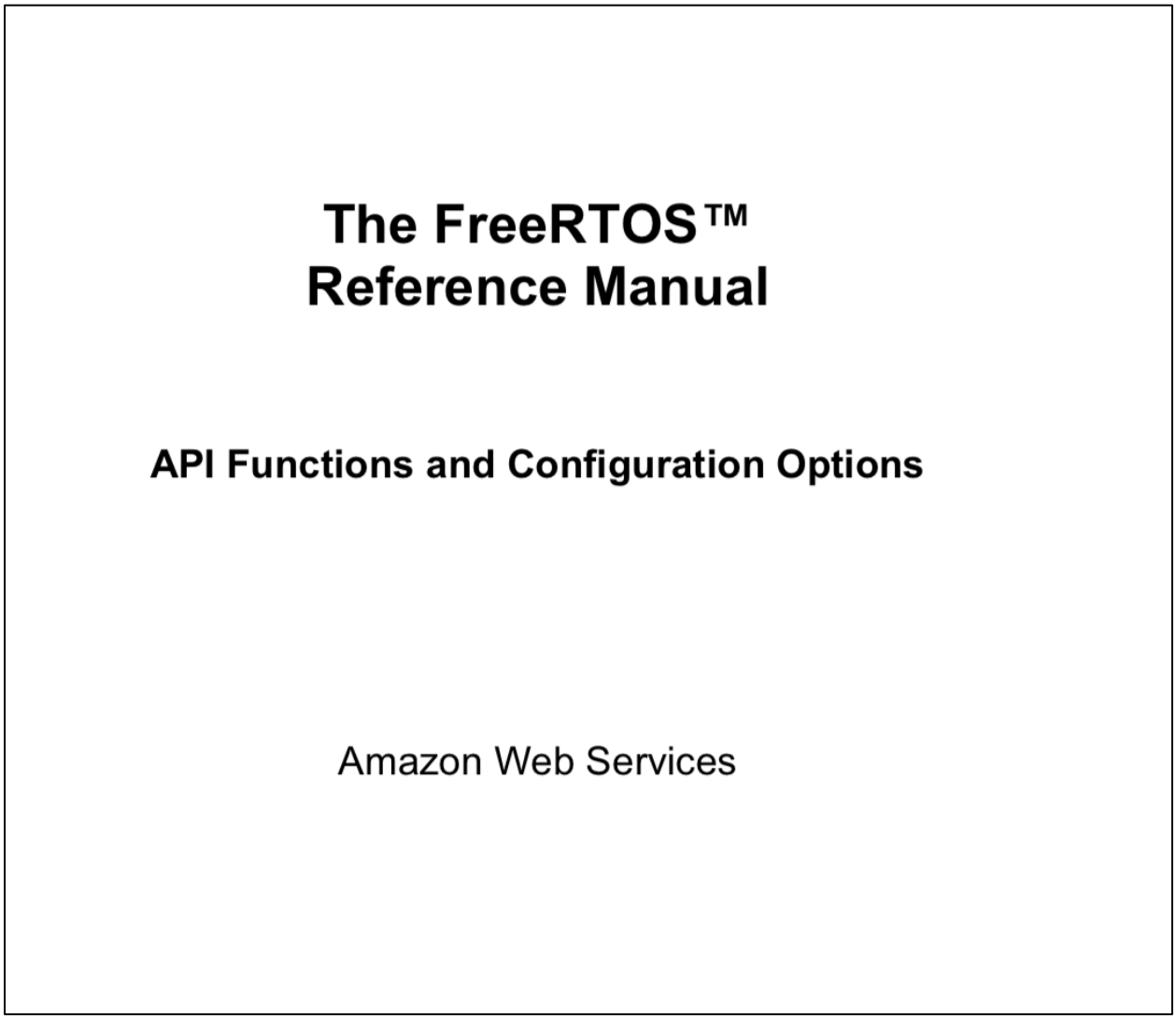
Before you begin, you must configure AWS IoT and your Amazon FreeRTOS download to connect your device to the AWS Cloud. See First Steps for instructions. In this tutorial, the path to the Amazon FreeRTOS download directory is referred to as *<amazon-freertos>*.

Note: On Windows OS, the root directory cannot have more than 98 characters. Please see Downloading Amazon FreeRTOS for details.

AWS Amazon FreeRTOS Documentation https://docs.aws.amazon.com/freertos/index.html



Freertos.org Reference Manual https://www.freertos.org/wp-content/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf



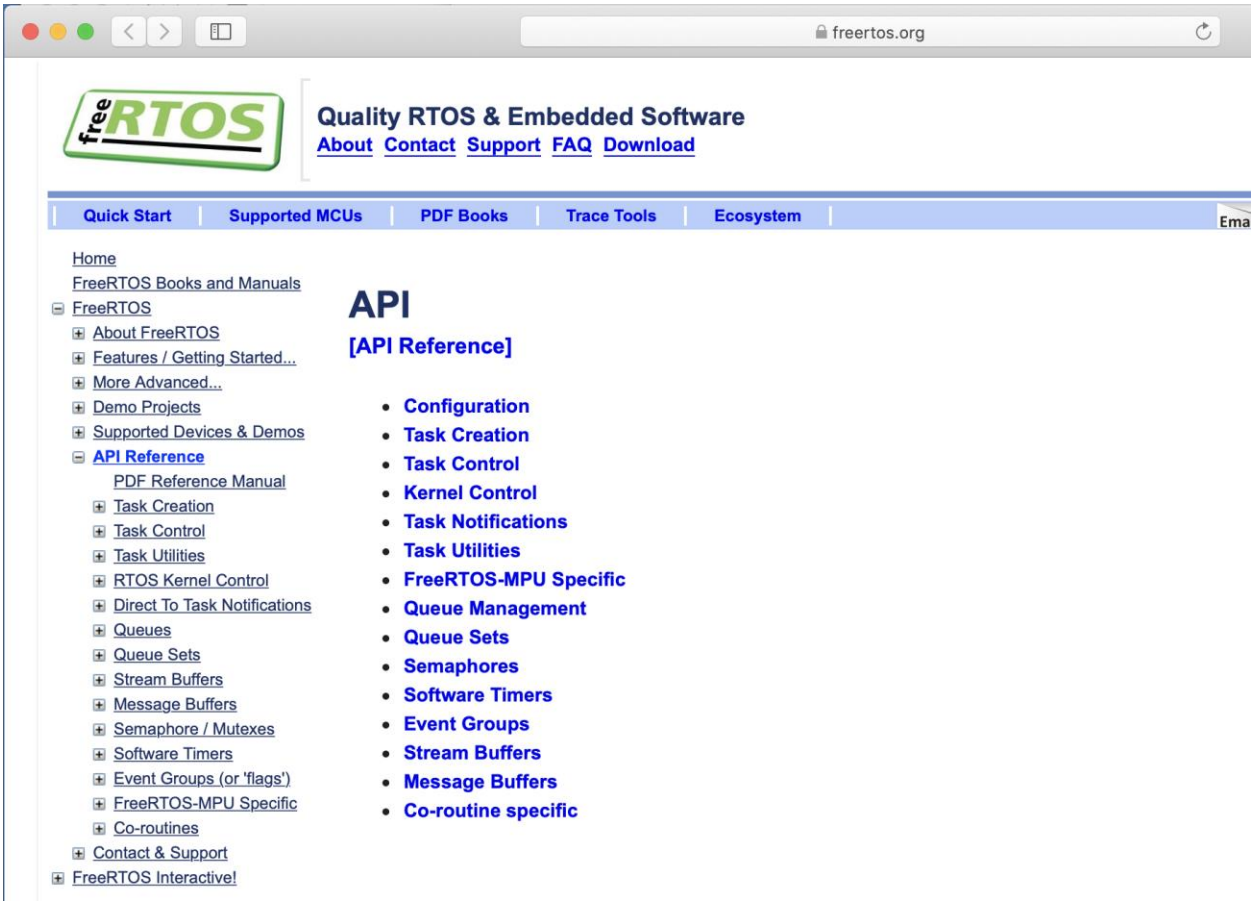
Mastering the FreeRTOS Real Time Kernel https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

Mastering the FreeRTOS™ Real Time Kernel

This is the 161204 copy which does not yet cover FreeRTOS V9.0.0, FreeRTOS V10.0.0, or low power tick-less operation. Check http://www.FreeRTOS.org regularly for additional documentation and updates to this book. See http://www.FreeRTOS.org/FreeRTOS-V9.html for information on FreeRTOS V9.x.x. See https://www.freertos.org/FreeRTOS-V10.html for information on FreeRTOS V10.x.x. Applications created using FreeRTOS V9.x.x onwards can allocate all kernel objects statically at compile time, removing the need to include a heap memory manager.

This text is being provided for free. **In return we ask that you use the business contact email link on http://www.FreeRTOS.org/contact to provide feedback, comments and corrections.** Thank you.

FreeRTOS API Reference https://www.freertos.org/a00106.html



Amazon FreeRTOS API Reference https://docs.aws.amazon.com/freertos/latest/lib-ref/index.html

Amazon FreeRTOS

Amazon FreeRTOS API Reference

Amazon FreeRTOS is a secure, cloud-native IoT operating system for microcontrollers.

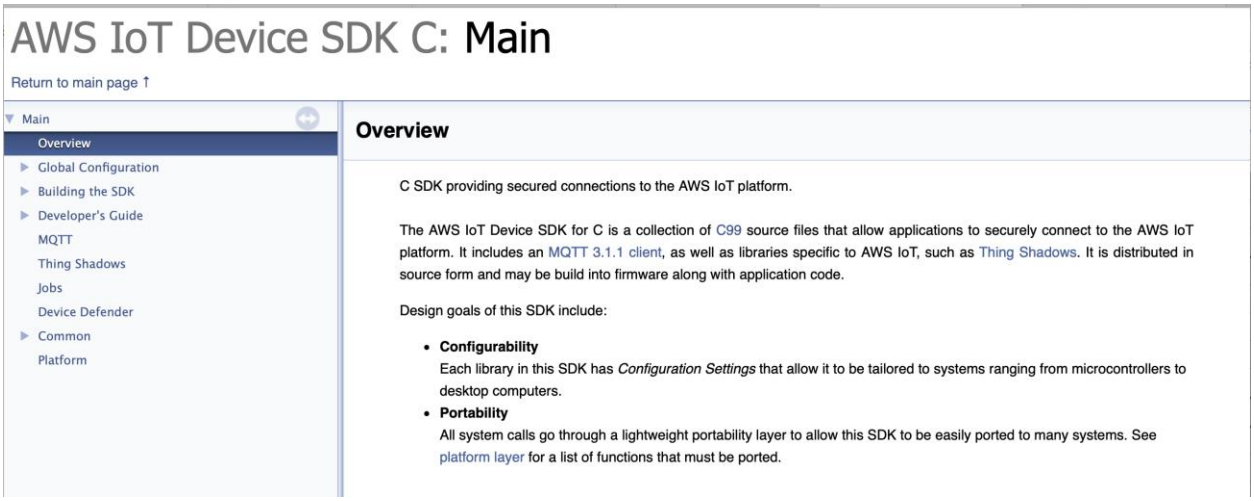
The following Amazon FreeRTOS libraries are documented in library-specific API references:

- Bluetooth Low Energy
- AWS Greengrass
- MQTT Core and MQTT Agent Interface (v1.0.0)
- AWS IoT OTA Agent
- Secure Sockets
- Wi-Fi
- HTTPS
- PKCS #11 API
    - mbedTLS-based PKCS #11 Implementation
    - PKCS #11 PAL
    - PKCS #11 Utilities
    - PKCS #11 Wrappers
- Provisioning API

The following Amazon FreeRTOS libraries share APIs with libraries in the AWS IoT Device SDK for Embedded C:

- Atomic Operations
- AWS IoT Device Defender
- Linear Containers
- Logging
- MQTT (v2.0.0)
- AWS IoT Device Shadow
- Static Memory
- Task Pool

Amazon AWS IoT Device SDK C: Main https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/main/index.html



6.2 Demo Walkthrough

Amazon FreeRTOS has built a crazy set of demo projects that are all wound together in one giant demo framework. It can be configured to run one of many tasks including MQTT, BLE, GreenGrass, etc.

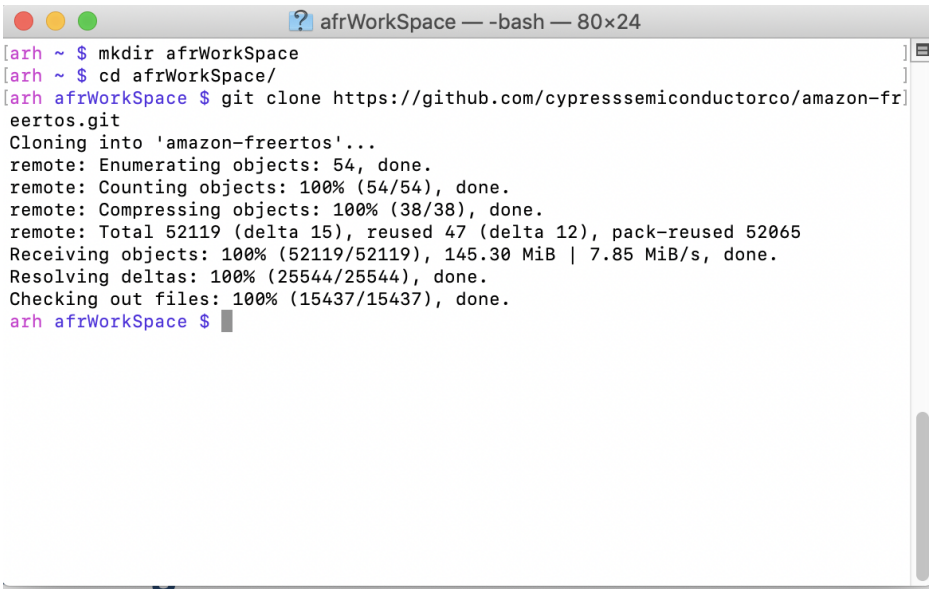
You can use Amazon FreeRTOS by cloning the Cypress Amazon-FreeRTOS GitHub site, and then importing the enclosed Eclipse project into the ModusToolbox IDE. You will be making changes directly in these directories and files (one copy of this repository per project).

Start by making a new directory (to hold the workspace and Eclipse project):

```
mkdir afrWorkSpace
cd afrWorkSpace
```

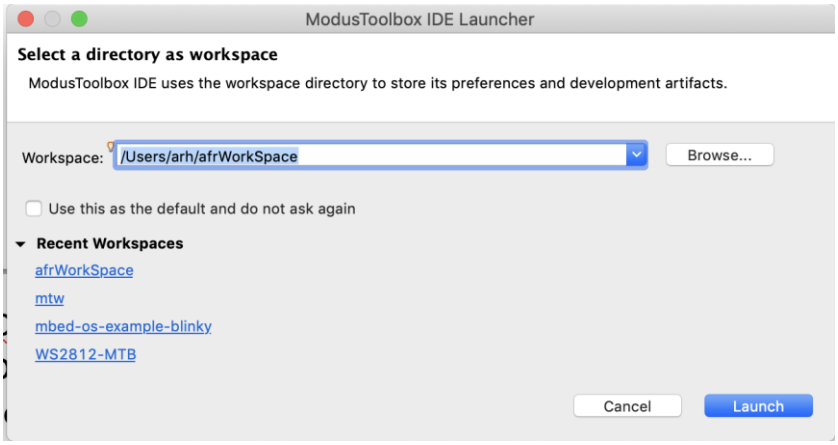
Then clone the project:

```
git clone https://github.com/cypresssemiconductorco/amazon-freertos.git
```

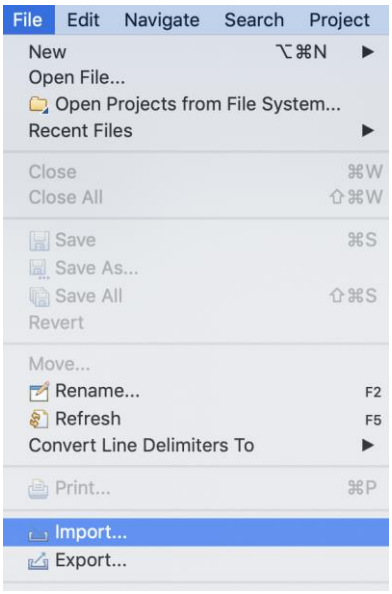


Amazon FreeRTOS is distributed with a Cypress created Eclipse project that works with the Cypress ModusToolbox IDE. To use the project, you need to import it into the ModusToolbox IDE. There can only be one Amazon FreeRTOS project in a workspace at a time (because of name collisions).

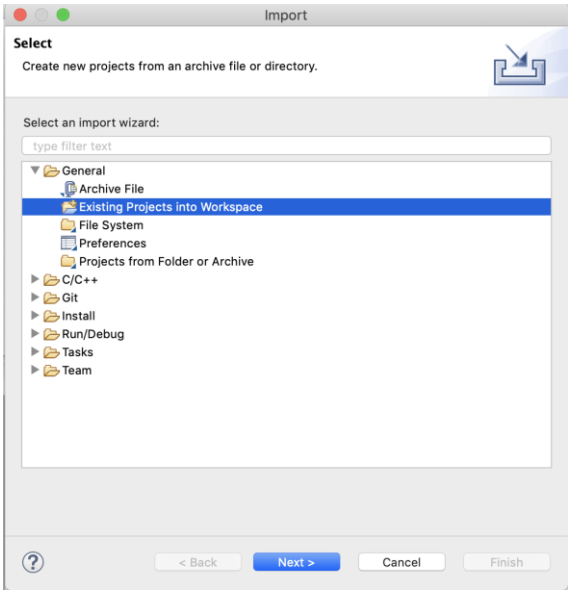
Launch the ModusToolbox IDE. Use the **Browse** button to open your new workspace (the one you created with the `mkdir` command.)



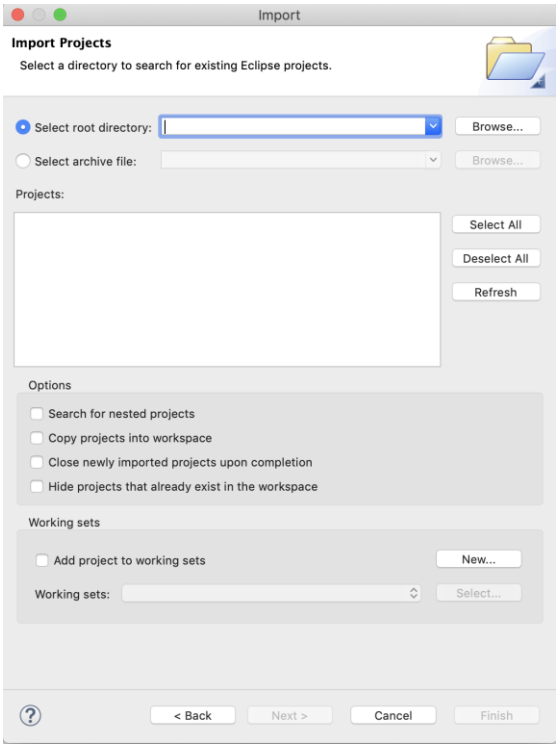
Use **File > Import** to open the Import dialog.



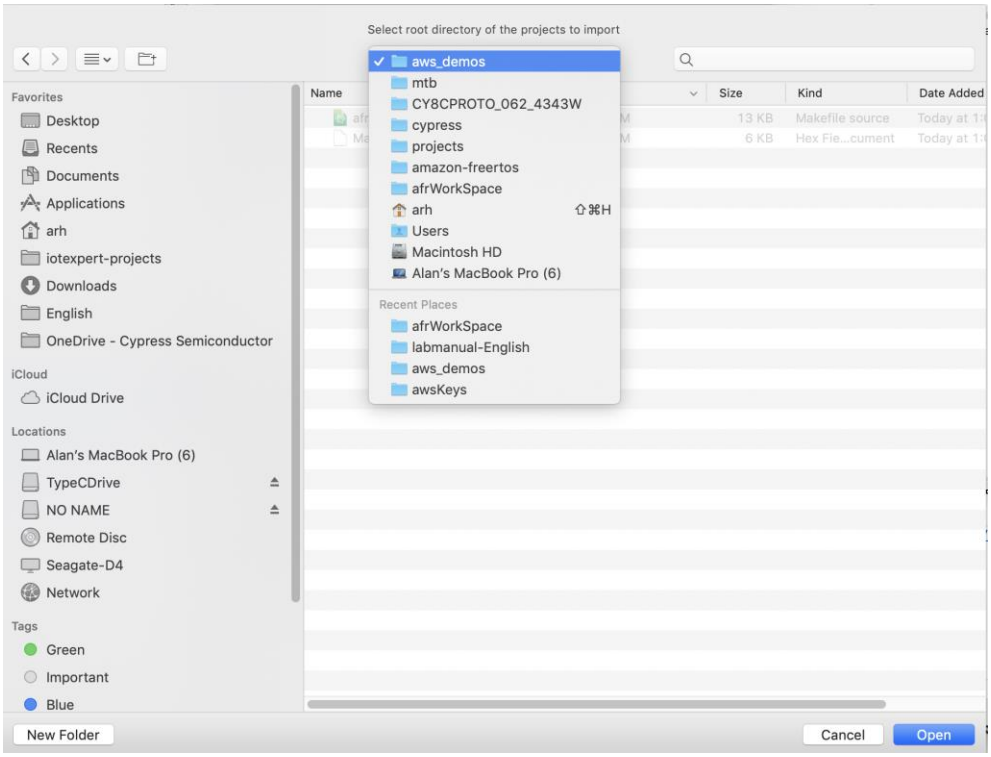
Select **General** > **Existing Projects into Workspace** and click **Next >**.



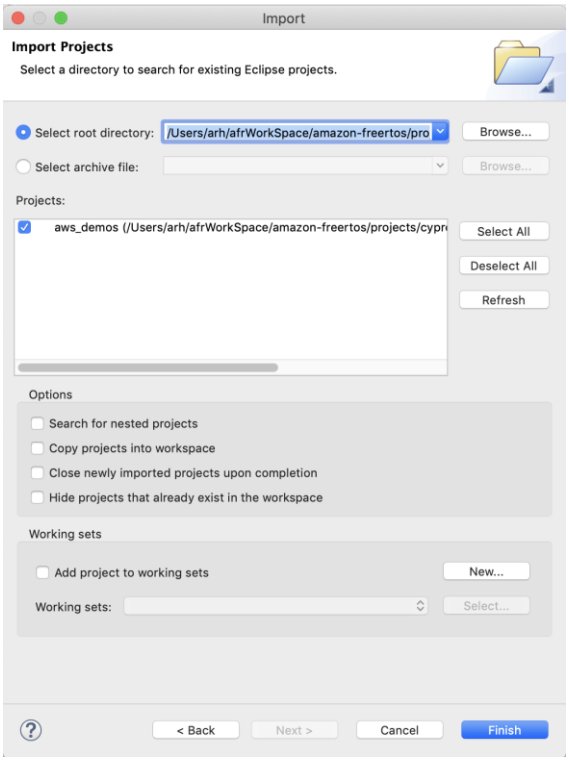
Click **Browse** and navigate to the **aws_demos** folder.



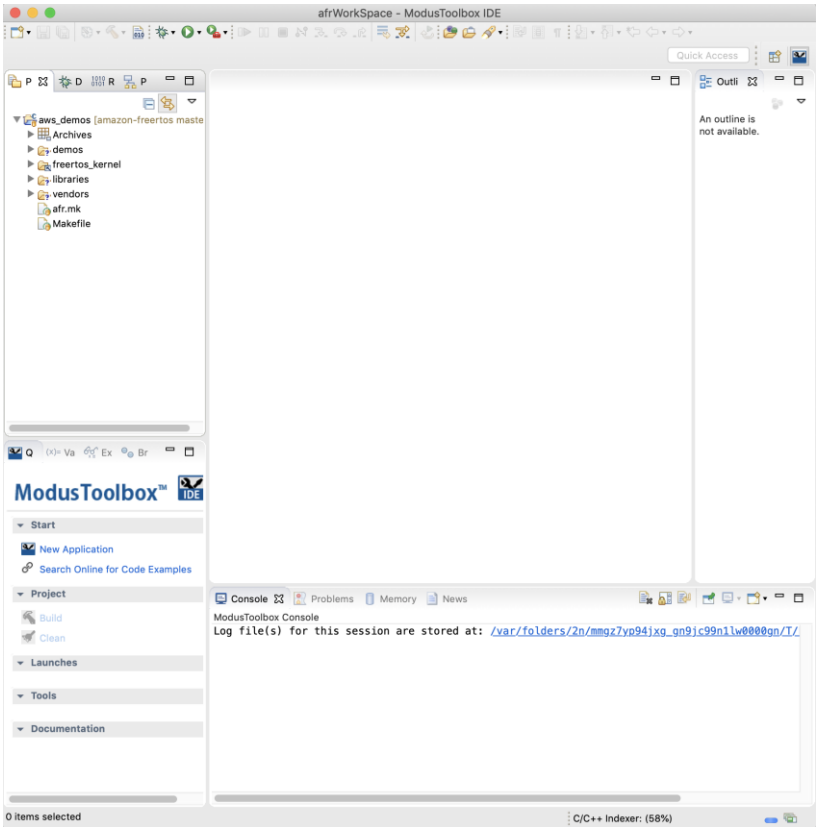
The path in the workspace is amazon-freertos/projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos.



Click **Finish** and view the project in the IDE.

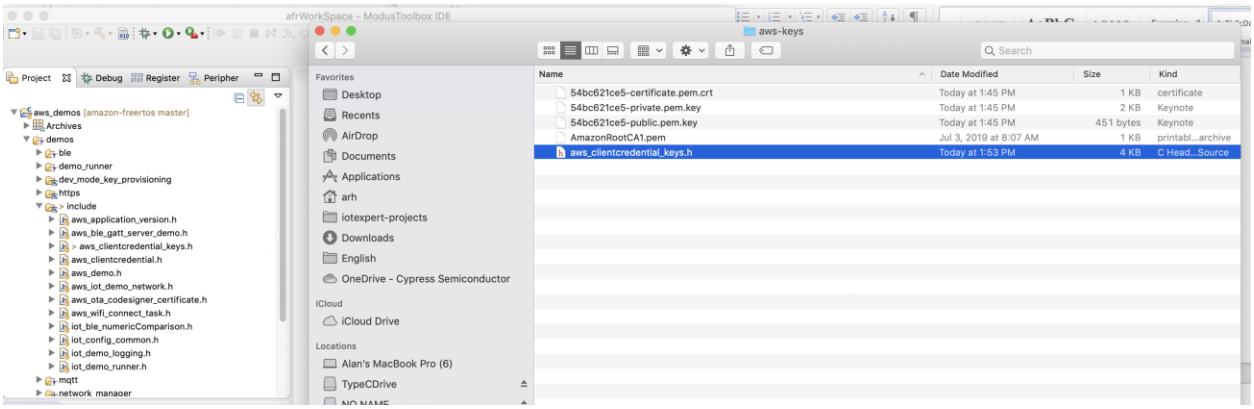


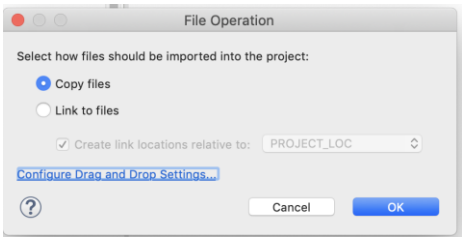
Your workspace should look something like this after you expand aws_demos:



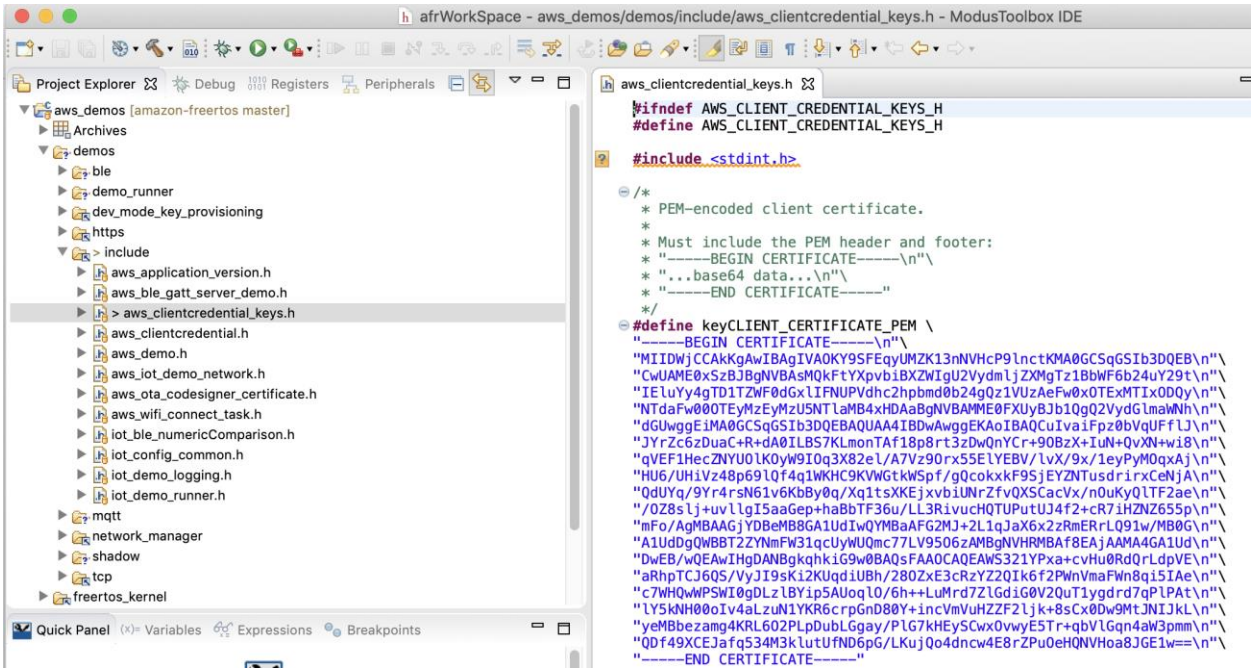
In order to make a TLS connection to the AWS IoT MQTT server, you will need to include the Certificate and Private key for your "thing". (See the AWS Chapter for instructions on making the keys.)

I have created the keys, which I store on my computer in a directory/file called aws-keys/aws_clientcredential_keys.h. To use them in a project, I copy the file from the file browser and paste it into the Eclipse project under demos/include. (Just drag and drop.)

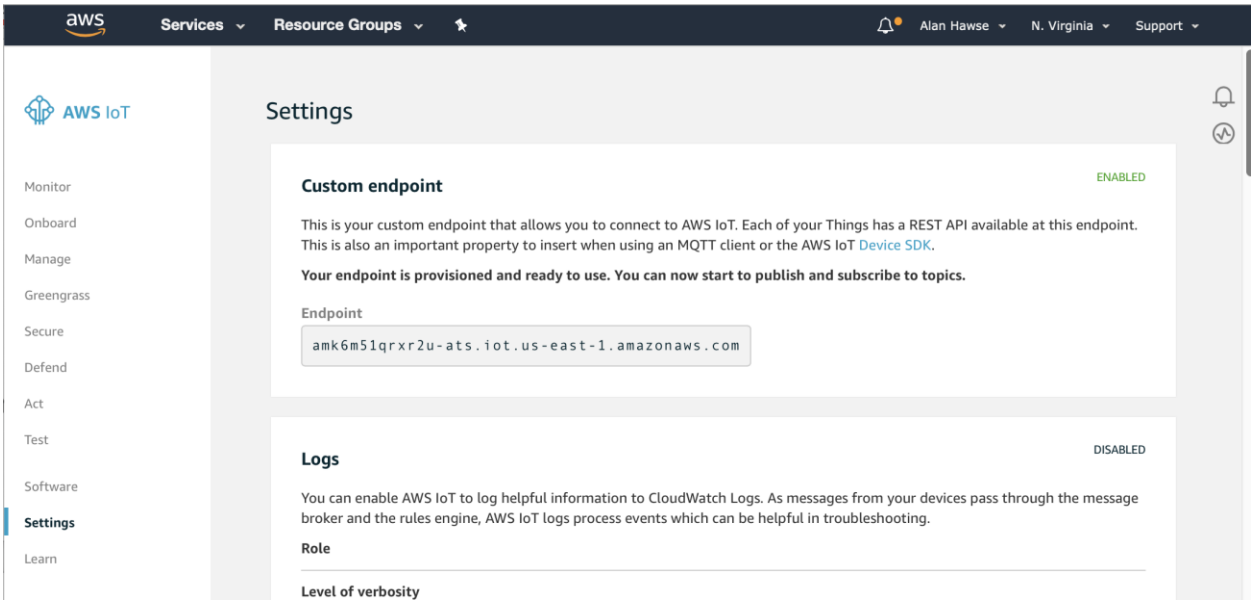




Once you have the keys in your project, open the aws_clientcredential_keys.h file and make sure that you have them correct.



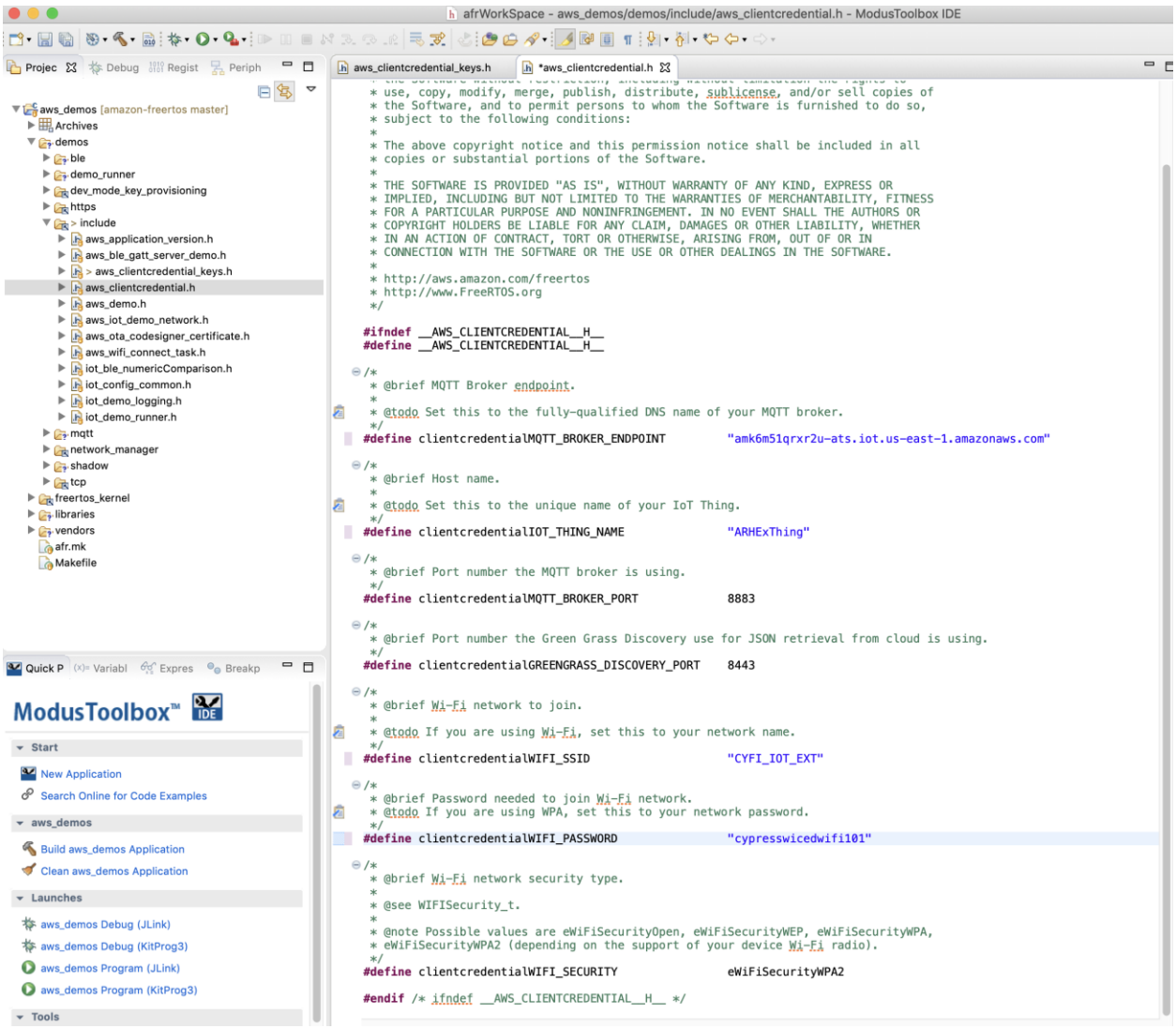
For your project to work, you need to tell it which MQTT broker to use. First, go to your Amazon Web Services IoT Core Settings screen and find the endpoint (aka the DNS name of your servers).



Next, open the file aws_clientcredential.h and paste the name into:

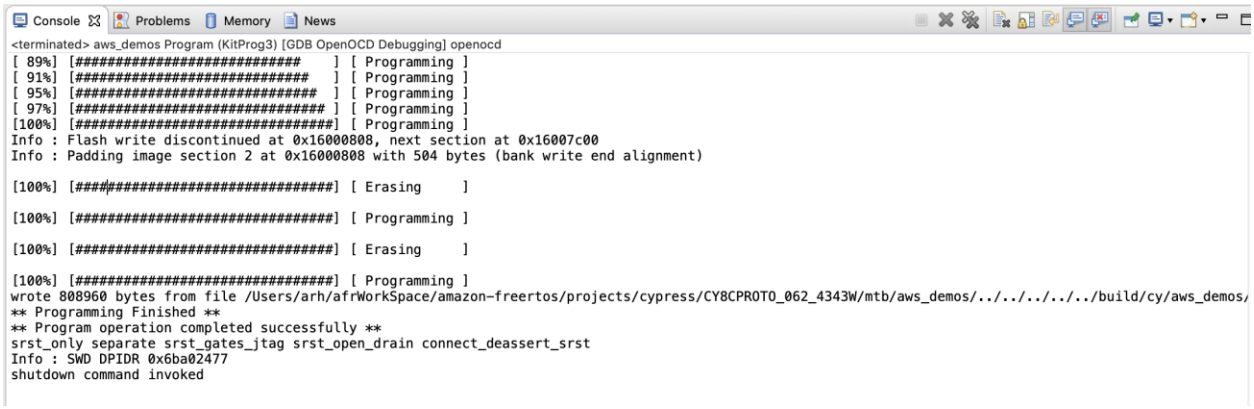
```
#define clientcredentialMQTT_BROKER_ENDPOINT
```

Then configure the name of your thing, as well as the SSID and password of your Wi-Fi AP.

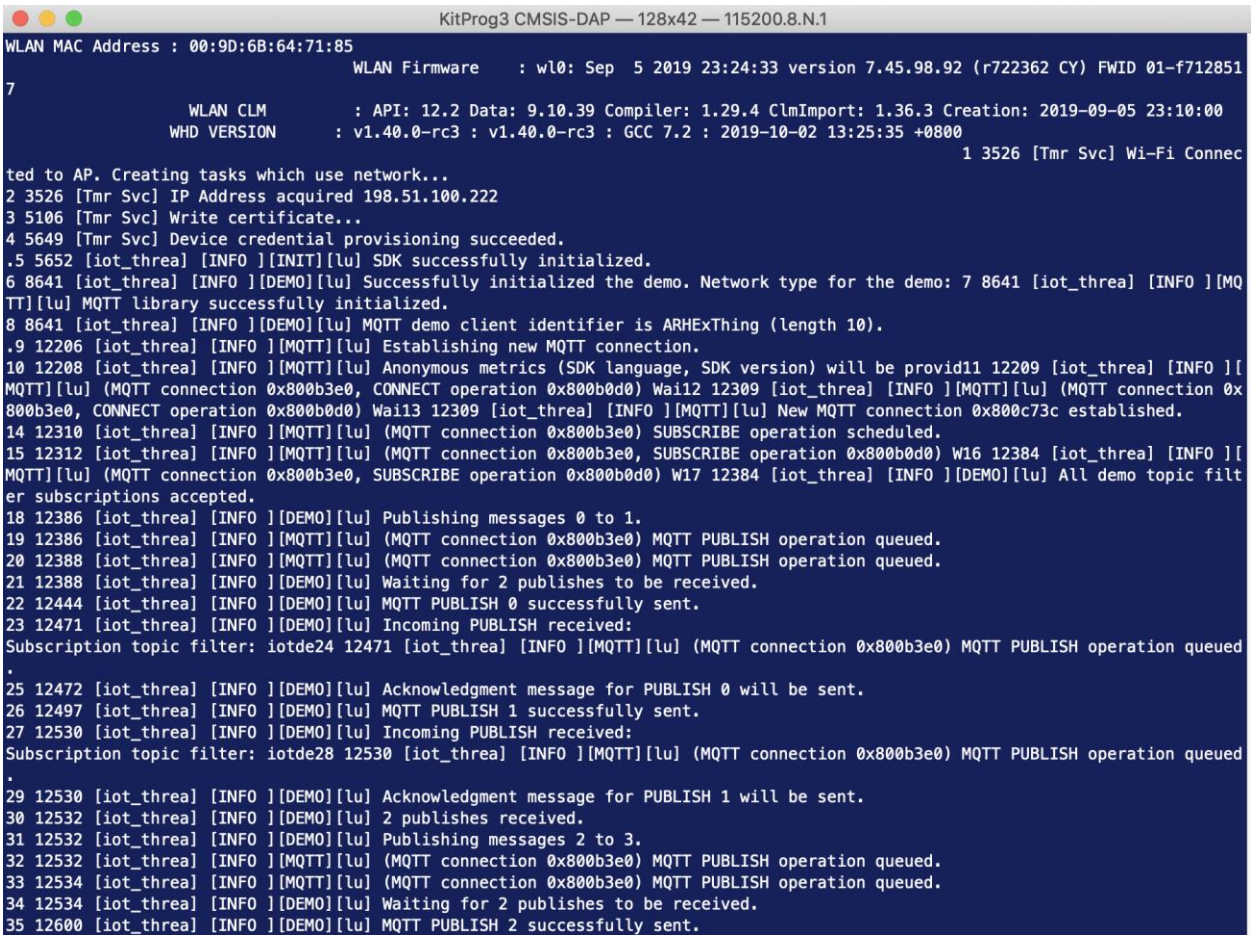


Now build/program the project by clicking “aws_demos Program (KitProg3)” in the Quick Panel.

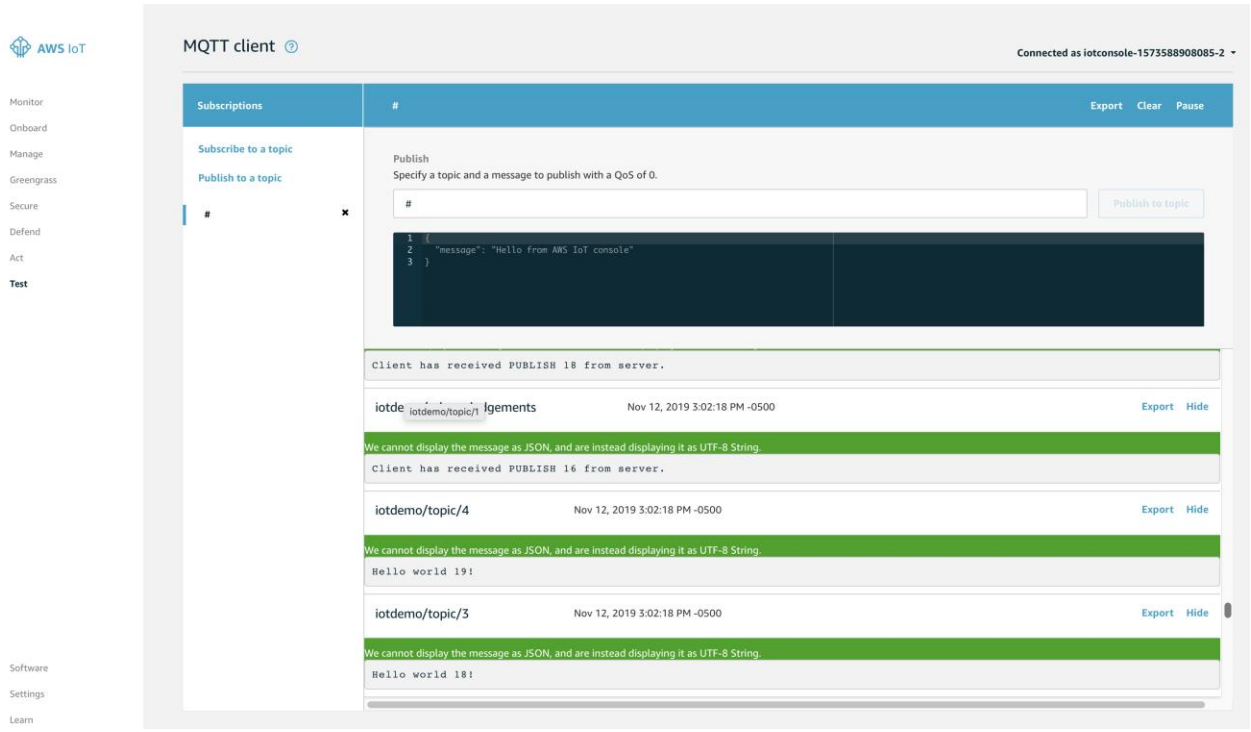
You console should look something like this:



And when you attach a serial terminal you should see something like this:



This demo does a series of publish/subscribes to an MQTT topic called “iotdemo/topic/#” You can see these messages being transmitted by subscribing via the MQTT test client on AWS IoT Core.



6.3 Exercises (Part 1)

6.3.1 Run the MQTT Demo

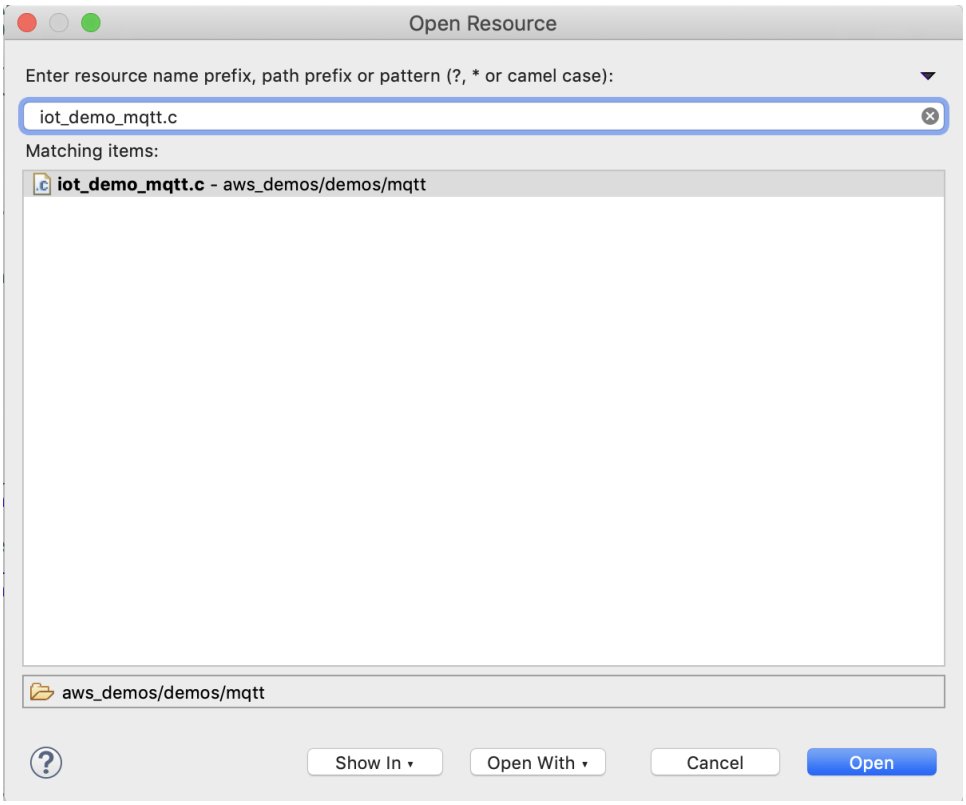
Build and run the demo following the steps in the Demo Walkthrough section.

6.3.2 Modify the project to publish to a different topic

Everyone in the class is publishing to the same topics, which will cause a significant amount of confusion. Modify your project to publish to a new topic; for example, your name.

Hint: Modify iot_demo_mqtt.c

Hint: A very useful Eclipse command is “Open Resource” which is Cmd+Shift+R (on a Mac) or Ctrl+Shift+R (on a PC).



6.4 AWS FreeRTOS Directory Organization

To use Amazon FreeRTOS you must import the entire repository from GitHub. You then make modifications to many of the source files in that repository to build your project. Each of your projects requires a complete copy of this repository.

Start with:

```
git clone https://github.com/cypresssemiconductorco/amazon-freertos.git
```

6.4.1 Top Level Organization

Directory	Comments
demos	Source code for the AWS released demos. Including MQTT, BLE, GreenGrass etc. – your project can be built with 0 files from this directory
build	Intermediate files created by the compilation process including .o's .elf etc. Can be removed
doc	Doxygen files (currently broken) – use online documentation at https://docs.aws.amazon.com/freertos/index.html
freertos_kernel	Source code for FreeRTOS
libraries	Amazon's IoT Middleware libraries including MQTT, HTTP etc.
projects	A directory containing IDE files which can be imported into Eclipse
tools	Support tools to help you build and test projects
vendors	Contains all the Cypress driver libraries including the HAL, PDL, Bluetooth, WHD etc.

6.4.2 demos

The demos directory contains all the Amazon FreeRTOS Demo projects. Each of the projects are documented on the AFR website https://docs.aws.amazon.com/freertos/latest/userguide/freertos-next-steps.html

To configure the demos you need to edit "aws_demo_config.h" and add the #define for your demo. The legal values are:

- CONFIG_MQTT_DEMO_ENABLED
- CONFIG_SHADOW_DEMO_ENABLED
- CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED (requires a GreenGrass Server)
- CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED (Requires a TCP Echo Server)
- CONFIG_DEFENDER_DEMO_ENABLED (Cypress doesn't deliver this demo)
- CONFIG_POSIX_DEMO_ENABLED (Cypress doesn't deliver this demo)
- CONFIG_OTA_UPDATE_DEMO_ENABLED (Cypress doesn't deliver this demo)
- CONFIG_BLE_GATT_SERVER_DEMO_ENABLED (requires MQTT bridge)
- CONFIG_HTTPS_SYNC_DOWNLOAD_DEMO_ENABLED (download from S3 bucket)
- CONFIG_HTTPS_ASYNC_DOWNLOAD_DEMO_ENABLED (download from S3 bucket)

These “#define”s will turn on configuration information in iot_demo_runner.h including the name of the function for the demo runner to run.

AWS Documentation » Amazon FreeRTOS » User Guide » Amazon FreeRTOS Demos

Amazon FreeRTOS Demos

Amazon FreeRTOS includes some demo applications in the `demos` folder, under the main Amazon FreeRTOS directory. All of the examples that can be executed by Amazon FreeRTOS appear in the `common` folder, under `demos`. There is also a folder for each Amazon FreeRTOS-qualified platform under the `demos` folder. If you use the Amazon FreeRTOS console, only the target platform you choose has a subdirectory under `demos`.

Before you try the demo applications, we recommend that you complete the tutorial in Getting Started with Amazon FreeRTOS. It shows you how to set up and run the Hello World MQTT demo.

Running the Amazon FreeRTOS Demos

The following topics show you how to set up and run the Amazon FreeRTOS demos:

- Bluetooth Low Energy Demo Applications
- Demo Bootloader for the Microchip Curiosity PIC32MZEF
- AWS IoT Device Defender Demo
- AWS IoT Greengrass Discovery Demo Application
- Over-the-Air Updates Demo Application
- Secure Sockets Echo Client Demo
- AWS IoT Device Shadow Demo Application

The `DEMO_RUNNER_RunDemos()` function, located in *<amazon-freertos>*`/demos/demo_runner/iot_demo_runner.c`, initializes a detached thread on which a single demo application runs. By default, `DEMO_RUNNER_RunDemos()` only calls the starts the Hello World MQTT demo. Depending on the configuration you selected when you downloaded Amazon FreeRTOS, and depending on where you downloaded Amazon FreeRTOS, the other example runner functions might start by default. To enable a demo application, open *<amazon-freertos>*`/vendors/`*<vendor>*`/boards/`*<board>*`/aws_demos/config_files/aws_demo_config.h`, and define the demo that you want to run.

> **Note**
>
> Be aware that not all combinations of examples work together. Depending on the combination, the software might fail to execute on the selected target due to memory constraints. We recommend that you run one demo at a time.

Configuring the Demos

The demos have been configured to get you started quickly. You might want to change some of the configurations for your project to create a version that runs on your platform. You can find configuration files at `vendors/`*<vendor>*`/boards/`*<board>*`/aws_demos/config_files`.

The demos directory contains the following:

- Configuration for the demo – in the include directory
- Demo source code in the demos/demo_runner directory
- 1 directory per example

demos/include	
aws_clientcredential.h	SSID Password MQTT_BROKER_ENDPOINT Thing Name
aws_clientcredential_keys.h	Contains the private key and certificate that are downloaded from AWS IoT core and converted into C strings.
aws_demo.h	Defines the generic interface for the demo function signature, e.g. the network connected/disconnected callbacks used by demos. This interface can be modified to add custom applications.
iot_demo_runner.h	Configures demo task function, stack size and priority for the selected demo in aws_demo_config.h. This file only needs to be modified when creating a custom demo.
iot_demo_logging.h	Configures default logging level used by the log library for the demo functions. These defines can be overridden in the iot_config.h file under board config files. Three logging levels are available IOT_LOG_LEVEL_DEMO, IOT_LOG_LEVEL_GLOBAL, IOT_LOG_LEVEL_NONE .
aws_application_version.h	Contains application firmware version. This is used to compare firmware version on the device during OTA operations. This can also be used by applications to track or print firmware version.
aws_ble_gatt_server_demo.h	Defines the GATT database for the aws_ble_gatt_server_demo.
aws_iot_demo_network.h	Defines the interface that allows demos to create a MQTT connection with underlying connection (WIFI/BLE) abstracted.
aws_ota_codesigner_certificate.h	Defines the certificate required for verifying digitally signed firmware. This certificate comes from the AWS manager when creating a digitally signed firmware image and must be converted to c string.
aws_wifi_connect_task.h	Defines the interface to create a task to connect to Wi-Fi. This is used by the network manager module.
iot_ble_numericComparison.h	
iot_config_common.h	Contains default configuration for several libraries and demos. Most of the default configurations in this file can be overridden by defining them in user configuration files. E.g. configuring the asserts, use of malloc vs pvPortMalloc in libraries, etc.
demos/demo_runner	
aws_demos.c	Contains support functions IFF you are using static memory allocation.
aws_demo_network_addr.c	Contains the network configuration definition for the FreeRTOS TCP stack. This file is not applicable Cypress devices, which use the LWIP stack.
aws_demo_version.c	Contains struct definition for the application firmware version.

demos/include	
iot_demo_freertos.c	Contains the implementation for the demo runner interface in aws_demo.h and default implementation for hooks (**vApplicationStackOverflowHook** /**vApplicationMallocFailedHook**).
iot_demo_runner.c	Contains implementation for demo runner interface in aws_demo.h.
Demo Directories	
ble	This directory holds three BLE example project, MQTT over BLE, WiFi Provisioning and Generic Attributes Server Demo. For these demos to work you must install the Amazon iOS/Android SDK for BLE to create a bridge to Amazon Web Services.
https	This directory holds an HTTPS demo that will download a file over a secure sockets HTTP(s) connection.
mqtt	This is the default application. It shows an MQTT connection to the AWS IoT Core with publishes and subscribes.
shadow	This demonstrates updating a device shadow via MQTT.
tcp	This project demonstrates using a secure TLS echo server. You run a "go" program which sits on the same network as your device and echo's back on a secure socket.
x-wifi_provisioning	This module creates a task for Wi-Fi provisioning that task maintains connection to the provisioned networks and gets triggered on every disconnection. This is used by the network manager.
dev_key_provisioning	This module is a reference implementation for storing and restoring keys and certificates used by the AWS connectivity library. Amazon suggests you use this as a reference to implement your own key provisioning.
network_manager	This modules is used by demo applications to handle different types of network connections and their connection/disconnection events.

6.4.3 vendors/cypress/ CY8CPROTO_062_4343W/aws_demos

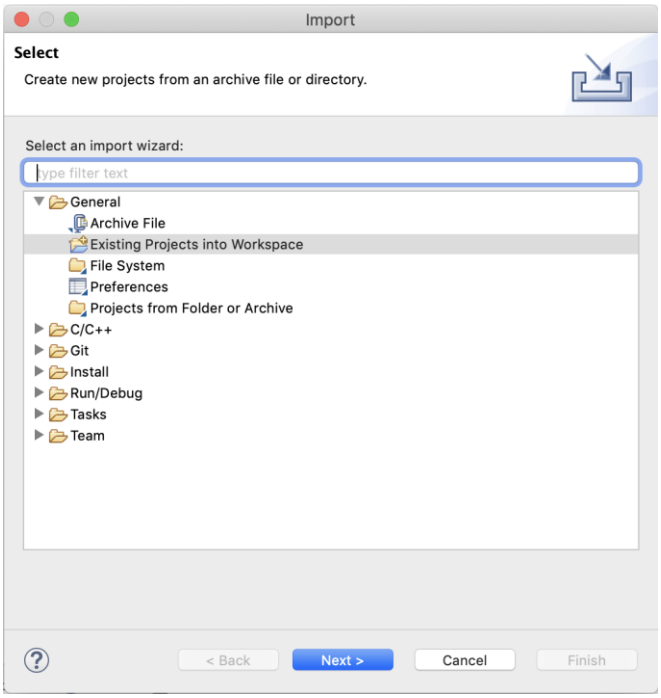
config_files	
FreeRTOSConfig.h	Configuration of FreeRTOS e.g. heap, semaphores, memory management, etc.
aws_demo_config.h	This file contains #defines that turn on the different demo projects. E.g. #define CONFIG_MQTT_DEMO_ENABLED.
aws_iot_network_config.h	Configure types of network interfaces supported/enabled by the board (Wi-Fi/BLE).
aws_ggd_config.h	Configuration for AWS GreenGrass library. Eg. Max JSON tokens, logging.
aws_mqtt_config.h	Configuration for the MQTT library. E.g. Max Subscriptions, Max Topic length, Enable debug logs etc.
aws_ota_agent_config.h	Configuration for the user configurable setting in the OTA library like ota task priority, selftest timeout etc.
aws_secure_sockets_config.h	Configuration for secure sockets. E.g. receive/send timeouts, max sockets, byte order etc.
aws_shadow_config.h	Configuration for the shadow library. E.g. Max clients, enable debug,

config_files	
aws_wifi_config.h	Configuration for the Wi-Fi port. E.g. Max network profile storage, SSID and password for AP mode.
FreeRTOSIPConfig.h	Configuration for the FreeRTOS TCP stack. Not applicable to Cypress devices.
Iot_ble_config.h	Configuration for the BLE library. E.g. device name etc.
Iot_config.h	Common demo configuration. E.g. Logging level, Shadow update count etc.
Iot_mqtt_agent_config.h	Configuration specifically for the MQTT agent in the MQTT library.
Iot_pkcs11_config.h	Configuration for the PKCS11 library. E.g. labels/key for the different keys and certificates for lookup. OTA code signing verification support.
lwipopts.h	This file configures the LWIP stack library. This is used by any application that requires WiFi connectivity. E.g. Buffer pool size, enable features like IPv6 etc.
trcConfig.h & trcSnapshotConfig.h	Percepio Tracelyzer configuration.
application_code	
main.c	Generic demo application code described in detail in the Application & Demo Firmware flow (source code) section.
cy_code	design.modus, generated source, and BSP.

6.4.4 projects

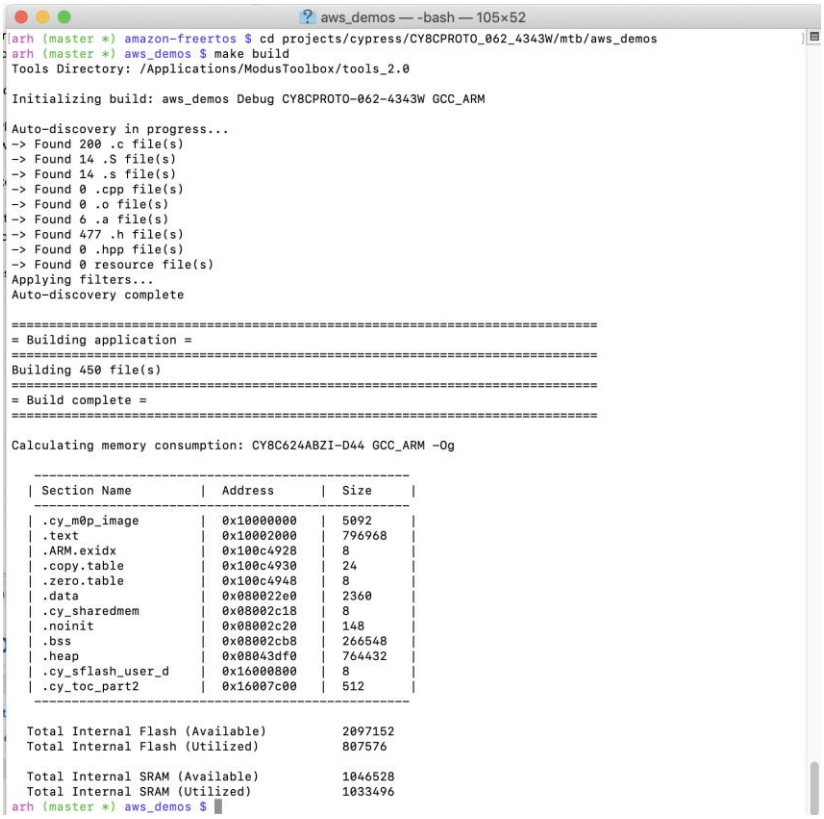
The projects directory contains the IDE configuration files, which includes the .cproject, afr.mk, and Makefile required use the demo projects in the ModusToolbox IDE and with the ModusToolbox command line.

To import the demo projects into ModusToolbox IDE, select **Import > General > Existing Projects into Workspace** and open the "projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos" directory.



To use the ModusToolbox command line:

```
cd projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos
make build
```



6.5 AWS Demo Network Manager

The AWS Demo Network Manager provides a generic API for performing common network operations e.g. connect or disconnect. These functions support WiFi, Bluetooth and Ethernet.

6.5.1 Interface Headers

These function (and parameters) are defined in:

- <amazon-freertos>/demos/network_manager/iot_network_manager_private.h
- <amazon-freertos>/demos/include/aws_iot_demo_network.h (extended for the demos)

6.5.2 Configuration Macros

The system is configured with a mix of Macros and function calls. The macros are shown in the following table, which are defined in this file:

<amazon-freertos>/vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_iot_network_config.h

Macro Name	Description
configSUPPORTED_NETWORKS	OR'ed list of networks supported by the board. The list is defined in <amazon-freertos>/libraries/c_sdk/standard/common/include/types/iot_network_types.h AWSIOT_NETWORK_TYPE_NONE AWSIOT_NETWORK_TYPE_WIFI AWSIOT_NETWORK_TYPE_BLE AWSIOT_NETWORK_TYPE_ETH AWSIOT_NETWORK_TYPE_ALL AWSIOT_NETWORK_TYPE_TCP_IP = (_WIFI | _ETH)
configENABLED_NETWORKS	OR'ed list of networks enabled for the project. See the description above for the network list.

6.5.3 Network States

The possible states of the network are defined in this file:

<amazon-freertos>/libraries/c_sdk/standard/common/include/types/iot_network_types.h

```
typedef enum AwsIotNetworkState
{
    eNetworkStateUnknown = 0, /*!< eNetworkStateUnknown State of the network is unknown */
    eNetworkStateDisabled,    /*!< eNetworkStateDisabled State of the network is disabled/disconnected */
    eNetworkStateEnabled      /*!< eNetworkStateEnabled  State of the network is enabled and connected. */
} AwsIotNetworkState_t;
```

6.5.4 Functions

The network manager provides the following functions:

Function	Description
AwsIotNetworkManager_Init()	Initializes the network manager.
AwsIotNetworkManager_GetConfiguredNetworks()	Returns the OR'ed list of configured networks. The list is fixed for a board. E.g. For CY8CPROTO-062-4343W, configured_networks = AWSIOT_NETWORK_TYPE_WIFI | AWSIOT_NETWORK_TYPE_BLE.

Function	Description
AwsIotNetworkManager_GetEnabledNetworks()	Returns the OR'ed list of enabled networks. `state != eNetworkStateUnknown` All the networks that are either enabled or disabled but initialized. See **Error! Reference source not found.** above.
AwsIotNetworkManager_GetConnectedNetworks()	Returns the OR'ed list of connected networks `state == eNetworkStateEnabled` All the networks that are both enabled and connected. See **Error! Reference source not found.** above.
AwsIotNetworkManager_EnableNetwork(uint32_t networkType)	Enables a network
AwsIotNetworkManager_DisableNetwork(uint32_t networkType)	Disables a network
AwsIotNetworkManager_GetNetworkInterface(uint32_t networkType)	Returns the interface of a network. The return type is `IotNetworkInterface` structure that has pointers to functions to perform operations such as create/delete a network connection, send/receive data over the connection.
AwsIotNetworkManager_GetCredentials(uint32_t networkType)	Returns the credentials of a network such as client certificate, private key etc. For Wi-Fi, these values are same as the macros defined in aws_clientcredential_keys.h.
AwsIotNetworkManager_GetConnectionParams(uint32_t networkType)	Returns the connection parameters such as port number, host name or MQTT end point. For Wi-Fi, these values are same as the macros defined in aws_clientcredential.h.
AwsIotNetworkManager_SubscribeForStateChange(networkType, callback, context, handle)	Subscribes for network state change – Allows multiple application-level subscribers (or tasks or threads) to subscribe for state change indication on the same network.
AwsIotNetworkManager_RemoveSubscription(handle)	Unsubscribes from the state change indication of a network

6.5.5 Network Manager - Caution

The system is parameterized with #defines rather than function calls in many places i.e. network parameters. It uses the macros defined in aws_clientcredential.h & aws_clientcredential_keys.h.

For example, the EnableNetwork() function just uses the Wi-Fi settings (SSID, password, and security type) defined in the aws_clientcredential.h file for establishing Wi-Fi connection.

Similarly, the GetCredentials() and GetConnectionParams() also return the values defined in those header files.

6.6 Application & Demo Firmware flow (source code)

All the application firmware follows the same general flow:

- Startup the system, FreeRTOS and connect to Wi-Fi
- Run the Demo_runner to start the demo
- The specific task(s) run (either your application and/or the demo task)

6.6.1 Startup

Function	Steps	Comments
main	prvMiscInitialization	Initializes BSP & retarget I/O
main	xLoggingTaskInitialize	Creates a task called “Logging” that sits on a queue and prints the messages with “configPRINT_STRING”
main	vTaskStartScheduler	Turns on FreeRTOS and runs the vApplicationDaemonTaskStartupHook
vApplicationDaemonTaskStartupHook	SYSTEM_Init	Initializes the crypto & socket metrics for the aws libraries
vApplicationDaemonTaskStartupHook	prvWifiConnect	Makes a connection to the WiFi network
vApplicationDaemonTaskStartupHook	vModeKeyProvisioning	Provision the device with AWS certificate and private key
vApplicationDaemonTaskStartupHook	DEMO_RUNNER_RunDemos	Runs the demo – by starting task from iot_demo_runner.c

6.6.2 DEMO_Runner

DEMO_RUNNER_RunDemos starts things going by calling Iot_CreateDetachedThread with a function pointer to runDemoTask and the argument being the demoContext_t structure. [iot_demo_freertos.c]

```
static demoContext_t mqttDemoContext =
{
    .networkTypes                = democonfigNETWORK_TYPES,
    .demoFunction                = DEMO_entryFUNCTION,
    .networkConnectedCallback    = DEMO_networkConnectedCallback,
    .networkDisconnectedCallback = DEMO_networkDisconnectedCallback
};
```

- DEMO_configNETWORK_TYPES is defined in aws_demo_config.h
- DEMO_entryFUNCTION is defined iot_demo_runner.h
- DEMO_networkConnectedCallback (not used)
- DEMO_networkDisconnectedCallback (not used)

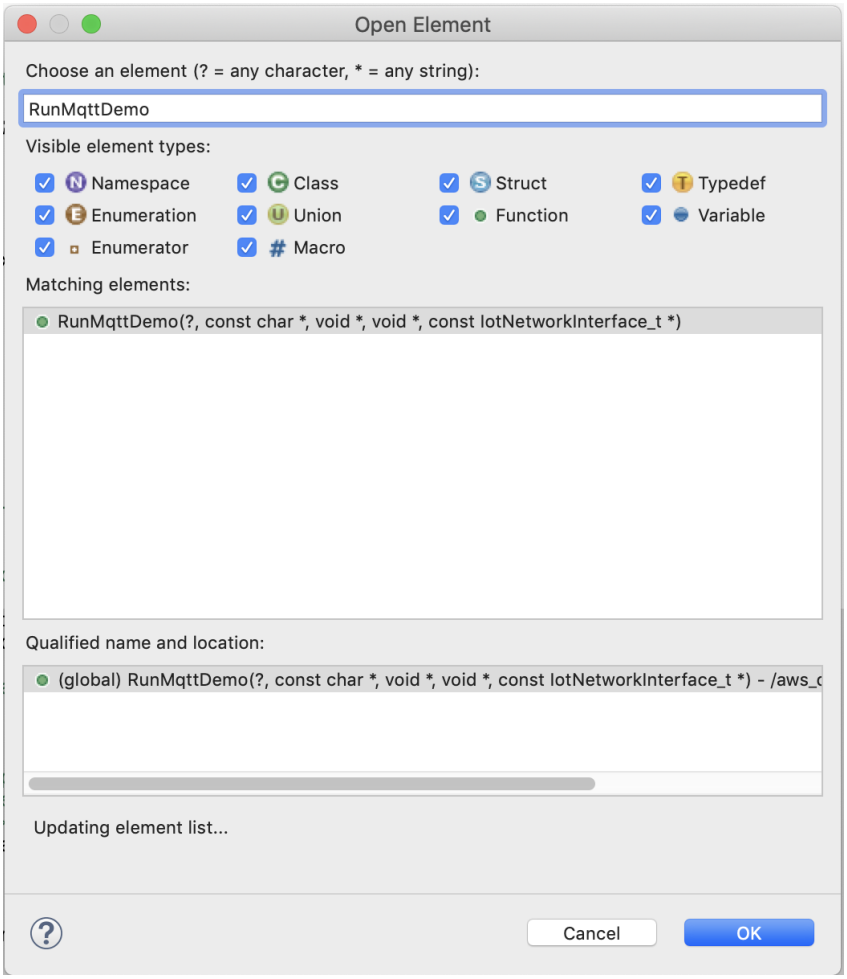
The runDemoTask calls _initialize which calls

- IotSdk_Init (initializes a task pool)
- AwsIotNetworkManager_Init
- AwsIotNetworkManager_SubscribeForStateChange
- AwsIotNetworkManager_EnableNetwork
- _getConnectedNetworkForDemo

Then it runs the demo function (via function pointer)

#define	Function	File
CONFIG_MQTT_DEMO_ENABLED	RunMqttDemo	iot_demo_mqtt.c
CONFIG_SHADOW_DEMO_ENABLED	RunShadowDemo	aws_iot_demo_shadow.c
CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED	vStartGreenGrassDiscoveryTask	aws_greengrass_discovery_demo.c
CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED	vStartTCPEchoClientTasks_SingleTasks	aws_tcp_echo_client_single_task.c
CONFIG_DEFENDER_DEMO_ENABLED	vStartDefenderDemo	aws_iot_demo_defender.c
CONFIG_POSIX_DEMO_ENABLED	vStartPOSIXDemo	
CONFIG_OTA_UPDATE_DEMO_ENABLED	vStartOTAUpdateDemoTask	aws_iot_ota_update_demo.c
CONFIG_BLE_GATT_SERVER_DEMO_ENABLED	vGattDemoSvcInit	aws_ble_gatt_server_demo.c
CONFIG_HTTPS_SYNC_DOWNLOAD_DEMO_ENABLED	RunHttpsSyncDownloadDemo	iot_demo_https_s3_download_sync.c
CONFIG_HTTPS_ASYNC_DOWNLOAD_DEMO_ENABLED	RunHttpsAsyncDownloadDemo	iot_demo_https_s3_download_async.c

It can be insane to try to figure out where functions, defines, etc. exist in the demo project. A way to find them quickly is to use Cmd+Shift+R (Mac) or Ctrl+Shift+R (PC) to run the “Open Element” dialog. In the following picture, you can see that I am looking for “RunMqttDemo”.



6.6.3 MQTT Demo

6.6.4 Shadow Demo

6.6.5 Your Application

6.7 FreeRTOS Configuration

FreeRTOS has a number of #defines that can be set in the FreeRTOSConfig file that change the behavior of FreeRTOS. By convention all start with “config” which means you define them inside of the FreeRTOSConfig.h.

Some of them you define as 0 or 1 and they turn off/on features; for example, configUSE_IDLE_HOOK. While others you link to a function; for example, configPRINTF

6.7.1 configUSE_DAEMON_TASK_STARTUP_HOOK – vApplicationDaemonTaskStartupHook

When you use `#define configUSE_DAEMON_TASK_STARTUP_HOOK` that will tell FreeRTOS to run the function vApplicationDaemonTaskStartupHook after the scheduler has started in the timer_svc thread. This means functions like vTaskDelay are functioning. This is used by the Amazon FreeRTOS example project to do things like connect to Wi-Fi before everything else gets going.

6.7.2 configUSE_IDLE_HOOK – vApplicationIdleHook

When `#define configUSE_IDLE_HOOK` is set to “1” it tells FreeRTOS to call the function “vApplicationIdleHook” which you MUST provide when the RTOS is idle. This function is provided by Cypress in main.c and just prints out a “.” every 5 seconds.

```
/**
 * @brief User defined Idle task function.
 *
 * @note Do not make any blocking operations in this function.
 */
void vApplicationIdleHook( void )
{
    /* FIX ME. If necessary, update to application idle periodic actions. */

    static TickType_t xLastPrint = 0;
    TickType_t xTimeNow;
    const TickType_t xPrintFrequency = pdMS_TO_TICKS( 5000 );

    xTimeNow = xTaskGetTickCount();

    if( ( xTimeNow - xLastPrint ) > xPrintFrequency )
    {
        configPRINT( "." );
        xLastPrint = xTimeNow;
    }
}
```

6.7.3 configUSE_TICK_HOOK – vApplicationTickHook

When `#define configUSE_TICK_HOOK` is set to “1” it tells FreeRTOS to call the function “vApplicationTickHook” which you MUST provide when the RTOS tick happens (that is, every 1 ms). This function is provided by Cypress and does a whole lot of nothing.

```
void vApplicationTickHook()
{
}
```

6.7.4 configPRINTF – vLoggingPrintf

This is a MACRO which mimics printf. FreeRTOS uses this macro to output debugging information. This macro is defined by default to call the AFR logging library print function:

```
#define configPRINTF( X )    vLoggingPrintf X
```

If you want to turn off debugging information (or prints) you could use:

```
#define configPRINTF( X )
```

6.7.5 configUSE_MALLOC_FAILED_HOOK – vApplicationMallocFailedHook

If `#define configUSE_MALLOC_FAILED_HOOK` is set to "1", when the function pvPortMalloc fails it calls this vApplicationMallocFailedHook. This function is provided in the file "demos/demo_runner/iot_demo_freertos.c". This means that YOU must define the function in your application if you remove the demo directory.

```
/**
 * @brief Warn user if pvPortMalloc fails.
 *
 * Called if a call to pvPortMalloc() fails because there is insufficient
 * free memory available in the FreeRTOS heap.  pvPortMalloc() is called
 * internally by FreeRTOS API functions that create tasks, queues, software
 * timers, and semaphores.  The size of the FreeRTOS heap is set by the
 * configTOTAL_HEAP_SIZE configuration constant in FreeRTOSConfig.h.
 *
 */
void vApplicationMallocFailedHook()
{
    configPRINTF( ( "ERROR: Malloc failed to allocate memory\r\n" ) );
    taskDISABLE_INTERRUPTS();

    /* Loop forever */
    for( ; ; )
    {
    }
}
```

6.7.6 configCHECK_FOR_STACK_OVERFLOW – vApplicationStackOverflowHook

If `#define configCHECK_FOR_STACK_OVERFLOW` is set to "1", when the stack overflows, FreeRTOS calls this vApplicationStackOverflowHook. This function is provided in the file "demos/demo_runner/iot_demo_freertos.c". This means that YOU must define the function in your application if you remove the demo directory.

```
/**
 * @brief Loop forever if stack overflow is detected.
 *
 * If configCHECK_FOR_STACK_OVERFLOW is set to 1,
 * this hook provides a location for applications to
 * define a response to a stack overflow.
 *
 * Use this hook to help identify that a stack overflow
 * has occurred.
 *
 */
void vApplicationStackOverflowHook( TaskHandle_t xTask,
                                    char * pcTaskName )
{
    configPRINTF( ( "ERROR: stack overflow with task %s\r\n", pcTaskName ) );
    portDISABLE_INTERRUPTS();

    /* Unused Parameters */
    ( void ) xTask;

    /* Loop forever */
    for( ; ; )
    {
    }
}
```

6.8 FreeRTOS Naming Convention(s)

The definitions in this section were copied from Amazon.com.

Amazon.com, Inc. or its affiliates (2017). *Reference Manual for FreeRTOS version 10.0.0 issue 1.* Retrieved from https://www.freertos.org/wp-content/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf, November 12, 2019

6.8.1 Variables

Variable Names

Variables are prefixed with their type: 'c' for char, 's' for short, 'l' for long, and 'x' for BaseType_t and any other types (structures, task handles, queue handles, etc.).

If a variable is unsigned, it is also prefixed with a 'u'. If a variable is a pointer, it is also prefixed with a 'p'. Therefore, a variable of type unsigned char will be prefixed with 'uc', and a variable of type pointer to char will be prefixed with 'pc'.

6.8.2 Function Names

Function Names

Functions are prefixed with both the type they return and the file they are defined in. For example:

- v**Task**PrioritySet() returns a void and is defined within **task**.c.
- x**Queue**Receive() returns a variable of type BaseType_t and is defined within **queue**.c.
- v**Semaphore**CreateBinary() returns a void and is defined within **semphr**.h.

File scope (private) functions are prefixed with 'prv'.

6.8.3 Macros

Table 4. Macro prefixes

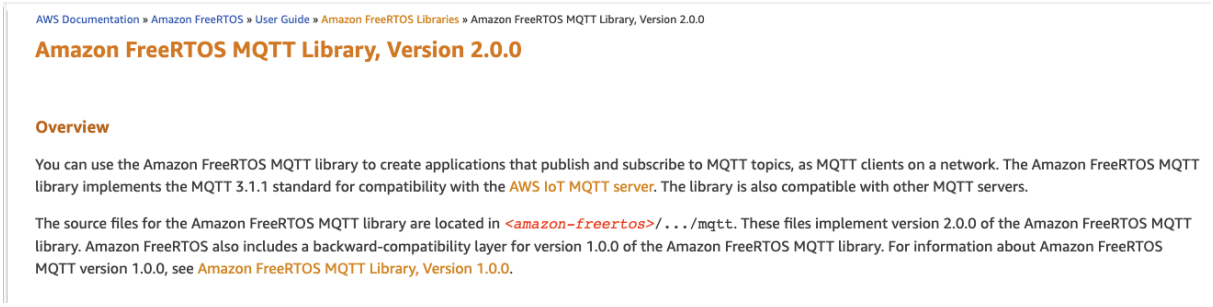
Prefix	Location of macro definition
port (for example, portMAX_DELAY)	portable.h
task (for example, taskENTER_CRITICAL())	task.h
pd (for example, pdTRUE)	projdefs.h
config (for example, configUSE_PREEMPTION)	FreeRTOSConfig.h
err (for example, errQUEUE_FULL)	projdefs.h

6.9 Amazon FreeRTOS Libraries

6.9.1 MQTT

Amazon FreeRTOS provides a MQTT library that implements MQTT protocol version 3.1.1. The library implements all the features required for compatibility with the AWS IoT MQTT broker. The library can also be used to connect to any other standard MQTT broker and to publish and subscribe MQTT topics. The library provides both a thread aware, synchronous and asynchronous APIs.

Note: Currently the MQTT library doesn't support QoS 2 MQTT messages.


AWS Documentation » Amazon FreeRTOS » User Guide » Amazon FreeRTOS Libraries » Amazon FreeRTOS MQTT Library, Version 2.0.0

Amazon FreeRTOS MQTT Library, Version 2.0.0

Overview

You can use the Amazon FreeRTOS MQTT library to create applications that publish and subscribe to MQTT topics, as MQTT clients on a network. The Amazon FreeRTOS MQTT library implements the MQTT 3.1.1 standard for compatibility with the AWS IoT MQTT server. The library is also compatible with other MQTT servers.

The source files for the Amazon FreeRTOS MQTT library are located in <amazon-freertos>/.../mqtt. These files implement version 2.0.0 of the Amazon FreeRTOS MQTT library. Amazon FreeRTOS also includes a backward-compatibility layer for version 1.0.0 of the Amazon FreeRTOS MQTT library. For information about Amazon FreeRTOS MQTT version 1.0.0, see Amazon FreeRTOS MQTT Library, Version 1.0.0.

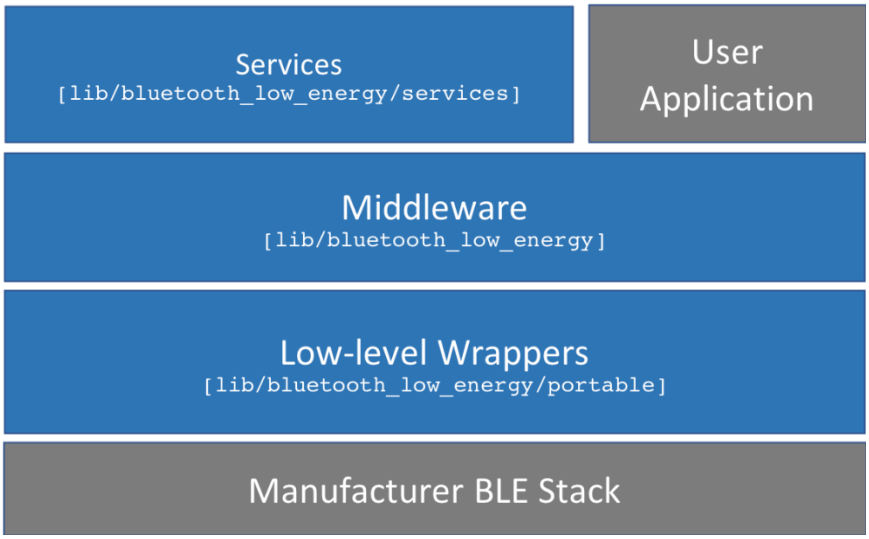
The API reference manual for this library is available at: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/mqtt_functions.html.

6.9.2 Bluetooth

Amazon FreeRTOS has defined an abstraction layer for Bluetooth. This library attempts to simplify the Bluetooth development experience, but typically requires that you make changes to the library files. In addition the Bluetooth library provides a scheme to bridge MQTT over GATT.

The API reference manual for this library is available at:
https://docs.aws.amazon.com/freertos/latest/lib-ref/ble/index.html

The library sits on top of the Cypress WICED Bluetooth Stack.



6.9.3 Logging

Amazon FreeRTOS has a built-in scheme that allows you to:

1. Create logging messages and send them to a logging queue/task.
2. The messages can be of the following levels:
    - 0/IOT_LOG_ERROR1
    - 1/IOT_LOG_WARN
    - 2/IOT_LOG_INFO
    - 3/IOT_LOG_DEBUG
    - 4/IOT_LOG_NONE
3. Messages can be suppressed on a file-by-file configurable level (only output messages < LIBRARY_LOG_LEVEL).
4. Files that don't specify LIBRARY_LOG_LEVEL will use IOT_LOG_LEVEL_GLOBAL.
5. Each file can specify which library they belong to by defining LIBRARY_LOG_NAME.
6. The log message contains:
    - A sequence number
    - System time
    - Thread
    - Level (configurable)
    - Library Name (configurable)
    - Time (configurable)
    - The message
7. The output device can be configured.
8. The system can be configured to use a static or dynamic memory scheme.

This system is nice because:

- Messages can be printed from an ISR.
- Message are printed in a standard format.
- Messages are serialized by the queue to fix the mutex problem with the logging system.
- Messages can be redirected to any output system by changing configPRINT_STRING.

The system is partially documented in the AWS Documentation https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/logging/.

To configure the system, you need to initialize in FreeRTOSConfig.h

- **extern void vLoggingPrintf**(**const char** * pcFormat, ...);
- **#define** configPRINTF(X) vLoggingPrintf X
- **extern void vLoggingPrint**(**const char** * pcMessage);
- **#define** configPRINT(X) vLoggingPrint(X)

In each file you need to:

- #define LIBRARY_LOG_NAME
- #define LIBRARY_LOG_LEVEL
- #include “iot_logging_setup.h”

The flow of the logging system code is as follows:

- main.c calls xLoggingTaskInitialize [iot_logging_task_dynamic_buffers.c] which starts a logging task running the function prvLoggingTask [iot_logging_task_dynamic_buffers.c] that waits on a queue (xQueue).
- When you use the macro functions IoTLogError, IoTLogWarn, IoTLogInfo, IoTLogDebug, they call the macro IoTLog [iot_logging_setup.h].
- IoTLog [iot_logging_setup.h] calls the function IoTLogGeneric [iot_logging.c].
- The function IoTGeneric [iot_logging.c] calls the macro IoTLogging_Puts [iot_config_common.h].
- IoTLogging_Puts [iot_config_common.h] calls the macro configPRINTF [FreeRTOSConfig.h].
- configPRINTF [FreeRTOSConfig.h] calls the function vLoggingPrintf [iot_logging_task_dynamic_buffers.c].
- vLoggingPrintf [iot_logging_task_dynamic_buffers.c] submits the message to the logging xQueue.
- When the Logging task receives messages in the xQueue it calls macro configPRINT_String [FreeRTOSConfig.h].
- configPRINT_STRING calls the function fputs(string, stdout).
- fputs is defined by the Cypress retarget-io library.

The files involved in the logging system are:

libraries/C_sdk/standard/common	
include/iot_logging_setup.h	Defines the macro’s IoTLog, IoTLogError, IoTLogWarn, IoTLogInfo, IoTLogDebug, IoTLogGeneric
include/iot_logging_task.h	Forward declaration for xLoggingTaskInitialize and vLoggingPrintf
include/private/iot_logging.h	Forward declarations of LOG_LEVEL_*, IotLogConfig_t
logging/iot_logging_task_dynamic_buffers.c	Defines the task which handles logging. Also defines vLoggingPrintf
logging/iot_logging.c	Defines the code which actually does the work of creating the message before sending it with configPRINTF

vendors/cypress/boards/cy8cproto_062_4343W/aws_demos/config_files	
FreeRTOSConfig.h	`#define configPRINTF vLoggingPrintf`

vendors/cypress/boards/cy8cproto_062_4343W/aws_demos/config_files	

iot_config.h	`#define IOT_LOG_LEVEL_GLOBAL IOT_LOG_INFO`
	`#define IOT_LOG_LEVEL_DEMO IOT_LOG_INFO`
	`#define IOT_LOG_LEVEL_PLATFORM IOT_LOG_NONE`
	`#define IOT_LOG_LEVEL_NETWORK IOT_LOG_INFO`
	`#define IOT_LOG_LEVEL_TASKPOOL IOT_LOG_NONE`
	`#define IOT_LOG_LEVEL_MQTT IOT_LOG_INFO`
	`#define AWS_IOT_LOG_LEVEL_SHADOW IOT_LOG_INFO`
	`#define AWS_IOT_LOG_LEVEL_DEFENDER IOT_LOG_INFO`

include/iot_config_common.h	`#define IotLogging_Puts( str ) configPRINTF( ( "%s\r\n", str ) )`
	`#define IotLogging_Malloc pvPortMalloc`
	`#define IotLogging_Free vPortFree`

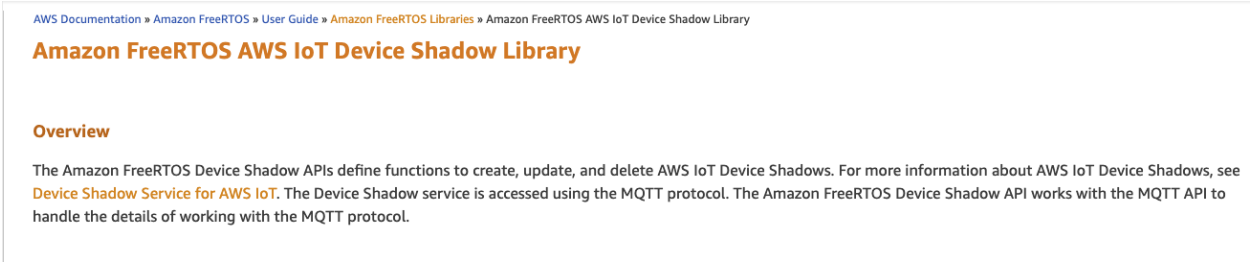
demos/include	

iot_demo_logging.h	`#define LIBRARY_LOG_LEVEL IOT_LOG_LEVEL_DEMO`
	
	`#define LIBRARY_LOG_NAME ( "DEMO" )`
	
	`#include "iot_logging_setup.h"`

6.9.4 AWS Device Shadow

The AWS IoT Core Device Shadow Service maintains a shadow for each device you connect to AWS IoT. A device's shadow is a persistent JSON document that is used to store and retrieve state information for a device, regardless of the device's connected state. Shadows support MQTT and HTTP protocols and have unique IDs derived from the "thing" name.

Amazon FreeRTOS IoT device shadow library provides APIs to interact with AWS IoT Thing shadow. Common use cases for Thing Shadows include backing up device state or sending commands to devices. The device shadow library provides synchronous and asynchronous APIs for modifying Thing Shadows and for registering notifications of a Thing Shadow change. IoT device shadow library uses MQTT library to send the messages that interact with the AWS IoT Thing Shadow service.


AWS Documentation » Amazon FreeRTOS » User Guide » Amazon FreeRTOS Libraries » Amazon FreeRTOS AWS IoT Device Shadow Library

Amazon FreeRTOS AWS IoT Device Shadow Library

Overview

The Amazon FreeRTOS Device Shadow APIs define functions to create, update, and delete AWS IoT Device Shadows. For more information about AWS IoT Device Shadows, see Device Shadow Service for AWS IoT. The Device Shadow service is accessed using the MQTT protocol. The Amazon FreeRTOS Device Shadow API works with the MQTT API to handle the details of working with the MQTT protocol.

The API reference manual for the library is available at:
https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/shadow/shadow_functions.html.

6.10 Using the Cypress HAL & PDL

The HAL and PDL in Amazon FreeRTOS are the same as for the PSoC 6 SDK chapter; refer to that chapter for more details. In order to use the HAL and PDL, include the following files in your project:

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
```

6.11 Build System

Amazon FreeRTOS is built around the CMAKE utility (which Cypress implemented). Optionally, you can use other build systems (we also implemented Make).

6.11.1 Make

The make build system is a Cypress provided system to build projects in Amazon FreeRTOS. This build system can be used either through ModusToolbox IDE or by using command-line interface (CLI).

ModusToolbox IDE

The ModusToolbox IDE project is pre-configured to build the demo projects using the make build system. When the aws_demos project is imported (per Section 6.2) and the user hits the build button it internally triggers the make build system to build the project.

Command-line interface (CLI)

A user can also build the project by running `make build` from the project directory (e.g. amazon-freertos/projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos). For other build options the user can run the `make help` command.

6.11.2 Files

There are two primary files that the user is expected to modify:

- Makefile: This is the application makefile. This contains several user configurable options such as target board, user defines, include paths, compiler flags etc.
- afr.mk: This contains the description of source files, include paths for all amazon provided libraries and demos projects. This also includes the amazon-freertos/vendors/cypress/boards. It includes all the sources required for supported demo projects by default but in order to include any additional library provided by amazon this file is expected to be modified.

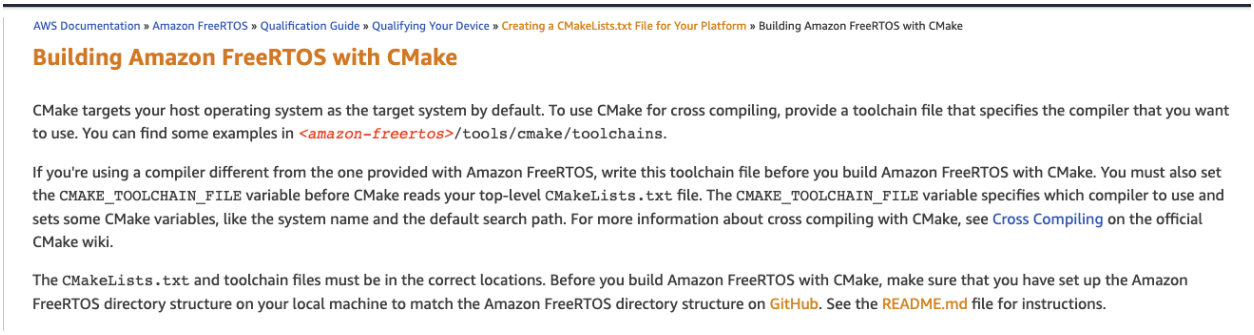
Each project for every Cypress board with have these two files.

Important Notes:

The make build system automatically searches for the sources in the project directory and all directories under the amazon-freertos/vendors/cypress folder **with the exception of the "boards" directory under that folder.** If any sources are added to this folder, they will be automatically picked up by the make build system and will not require any changes from the user.

6.11.3 CMake

Amazon also provides a CMake based build system to build projects in Amazon FreeRTOS. The instructions to using this build system are documented at: https://docs.aws.amazon.com/freertos/latest/qualificationguide/building-cmake.html.



AWS Documentation » Amazon FreeRTOS » Qualification Guide » Qualifying Your Device » Creating a CMakeLists.txt File for Your Platform » Building Amazon FreeRTOS with CMake

Building Amazon FreeRTOS with CMake

CMake targets your host operating system as the target system by default. To use CMake for cross compiling, provide a toolchain file that specifies the compiler that you want to use. You can find some examples in *<amazon-freertos>*/tools/cmake/toolchains.

If you're using a compiler different from the one provided with Amazon FreeRTOS, write this toolchain file before you build Amazon FreeRTOS with CMake. You must also set the CMAKE_TOOLCHAIN_FILE variable before CMake reads your top-level CMakeLists.txt file. The CMAKE_TOOLCHAIN_FILE variable specifies which compiler to use and sets some CMake variables, like the system name and the default search path. For more information about cross compiling with CMake, see Cross Compiling on the official CMake wiki.

The CMakeLists.txt and toolchain files must be in the correct locations. Before you build Amazon FreeRTOS with CMake, make sure that you have set up the Amazon FreeRTOS directory structure on your local machine to match the Amazon FreeRTOS directory structure on GitHub. See the README.md file for instructions.

6.12 Coming in Future Releases

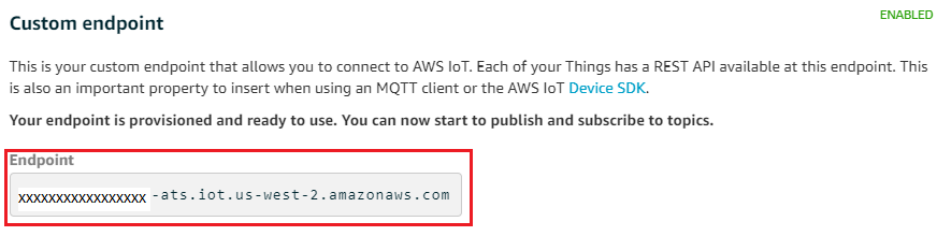
This is the first release of Cypress' Amazon FreeRTOS SDK. Cypress expects major improvements in the next few releases including but not limited to:

- OTA Support
- Easy Low Power Middleware and Configurations
- Additional Cypress Code Examples
- Middleware Management Tool (add/remove middleware to your project easily)
- Additional key middleware for your IoT designs
- Improved Eclipse workspace file organization and presentation

6.13 Exercises (Part 2)

6.13.1 AWS IoT Setup

1. Sign in to the https://console.aws.amazon.com/iot/ using your AWS account.
2. From the left navigation pane, click Settings and note down the endpoint address. In the endpoint shown in the picture below, the AWS region will be us-west-2.



Custom endpoint ENABLED

This is your custom endpoint that allows you to connect to AWS IoT. Each of your Things has a REST API available at this endpoint. This is also an important property to insert when using an MQTT client or the AWS IoT Device SDK.

Your endpoint is provisioned and ready to use. You can now start to publish and subscribe to topics.

Endpoint

xxxxxxxxxxxxxxxx -ats.iot.us-west-2.amazonaws.com

3. From the left navigation pane, choose Manage and then choose Things.
4. On top right corner, click the Create button.

5. Click Create a single thing at the bottom right corner.
6. Enter a name for the thing (e.g. cykit_app) and click Next at the bottom.
7. In the next page, click Create thing without certificate. The new thing you have created should appear under Things page.
8. From the left navigation pane, choose Secure and then Policies.
9. On top right corner, click the Create button.
10. Enter a Name for the policy (e.g. MyIotPolicy).
11. Click Advanced mode under Add Statements.
12. Delete all the existing statements, copy & paste the following, and click Create. The new policy you have created should appear under Policies page.

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

13. Make a note of the name of the policy that you have just created.

6.13.2 AWS Cognito Setup

User Pool Setup

1. In the AWS Console, click Services from the top left corner and click Cognito under Security, Identity, & Compliance.
2. Click Manage User Pools.
3. On the top right corner, click Create a user pool.
4. Enter a name under Pool name (e.g. cykit_mqtt_proxy_user_pool) and click Review defaults.
5. Under App clients, click Add app client... and click Add an app client.
6. Enter a name under App client name (e.g. mqtt_proxy_app_client), ensure that Generate client secret is selected, and click Create app client button at the bottom.
7. Click Return to pool details
8. Click Create pool at the bottom. You should see a message Your user pool was created successfully.
9. Make a note of the Pool Id. The Pool Id is in the format - <region-name>_xxxxxxxxx

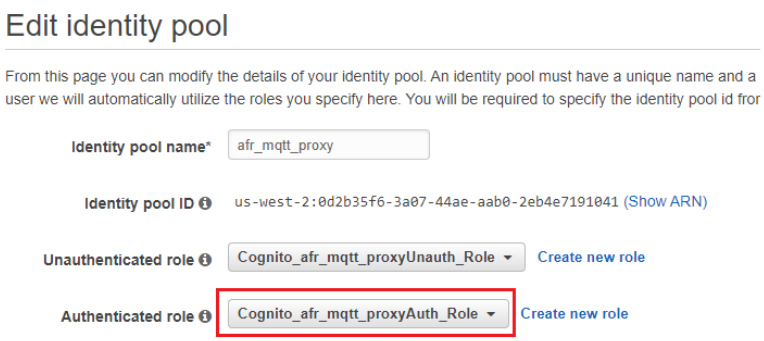
10. In the left-side navigation pane, choose App clients and click Show Details.
11. Make a note of the App client id and App client secret.

Identity Pool Setup

1. In the AWS Console, click Services from the top left corner and click Cognito under Security, Identity, & Compliance.
2. Click Manage Identity Pools
3. On the top right corner, click Create new identity pool.
4. Enter a name under Identity pool name (e.g. afr_mqtt_proxy)
5. Expand Authentication providers
6. Choose the Cognito tab
7. Enter the User Pool ID and App client id created during User Pool Setup and click Create Pool at the bottom.
8. Click Allow at the bottom of the page
9. Make a note of the Identity pool ID which is in the format of <region-name>:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. This ID is in red color in the code snippet that is under Get AWS Credentials.

Attach an IAM Policy

1. Go to Cognito, choose Manage Identity Pools, and click the identity pool (e.g. afr_mqtt_proxy) that you just created.
2. Click Edit identity pool at the top right corner
3. Make note of the IAM Role assigned to the Authenticated role (e.g. Cognito_afr_mqtt_proxyAuth_Role).



4. In the AWS Console, click Services from the top left corner and click IAM under Security, Identity, & Compliance.

5. In the left navigation pane, click Roles
6. Search for the role that you noted down in step 3 above and choose it.
7. Click Add inline policy at the right corner and choose JSON tab.
8. Copy & past the following policy into the editor

json_policy

```
{
    "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": [
                "*"
            ]
        }
    ]
}
```

9. Click Review policy button at the bottom
10. Enter a Name for the policy (e.g. mqtt_proxy_app_policy) and click Create policy button at the bottom.

6.13.3 Building the iOS App

Before starting, You need to complete both AWS IoT Setup and AWS Cognito Setup, as well as have the following ready:

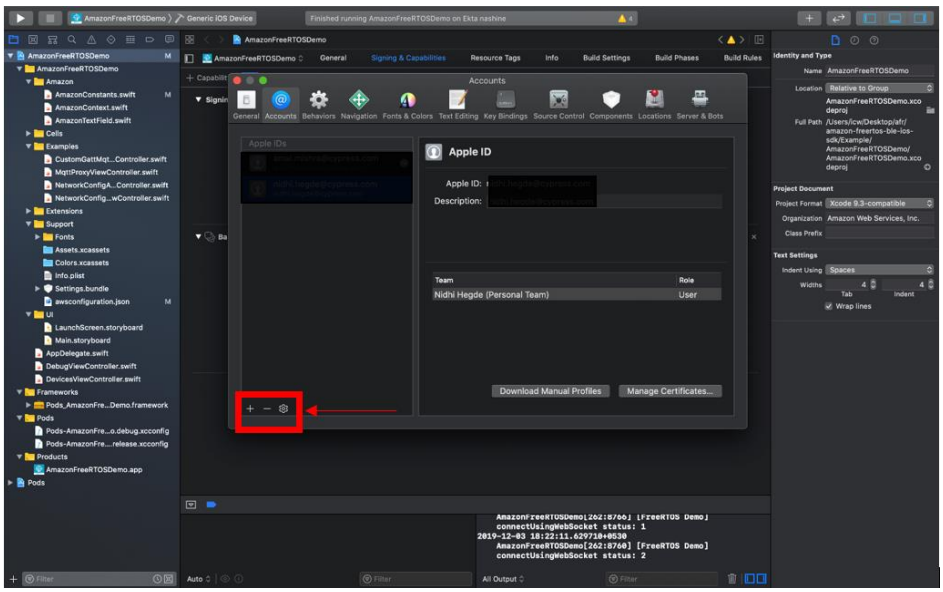
- AWS IoT policy name
- Region name
- Identity Pool ID
- User pool ID
- App client ID
- App client secret

If you are using Android, skip to Building the Android App section.

Steps

1. On the mac device, clone or download iOS SDK for AmazonFreeRTOS Bluetooth Devices.
2. To set up the SDK:
   a. Install Cocoapods. Open terminal and type:
      ```
      sudo gem install cocoapods
      ```
      You might get a prompt to enter password. Enter the login password of your mac.
      Note: If the above command does not work, try this:
      ```
      sudo gem install cocoapods -n /usr/local/bin
      ```
   b. Next type:
      ```
      pod setup
      ```
3. Go to the folder where you have cloned the iOS SDK. Navigate to amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo, open podfile. In the podfile, find the line:
   ```
   pod 'AmazonFreeRTOS', :path => ''
   ```
   Replace with:
   ```
   pod 'AmazonFreeRTOS', :git => 'https://github.com/aws/
   amazon-freertos-ble-ios-sdk.git'
   ```
4. Save the file.
5. In the terminal, navigate to amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo and type:
   ```
   pod install
   ```
6. Next, open Xcode, click file > open > (folder where sdk is cloned) > amazon-freertos-ble-ios-sdk > Example > AmazonFreeRTOSDemo > AmazonFreeRTOSDemo.xcworkspace

7. In the menu bar, go to Xcode > preferences > accounts. Click on '+' in the bottom left corner of the window, and add your apple account. If you do not have an apple account, create one at apple.com. After creating, go to Xcode and add your account as described.



8. Open amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift and change the values of the following variables:

   - region: your aws region
   - iotpolicyname: Your AWS IOT policy name
   - mqttCustomTopic: topic that you want to publish to

9. Open amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json

   Under CognitoIdentity, redefine the following variables:

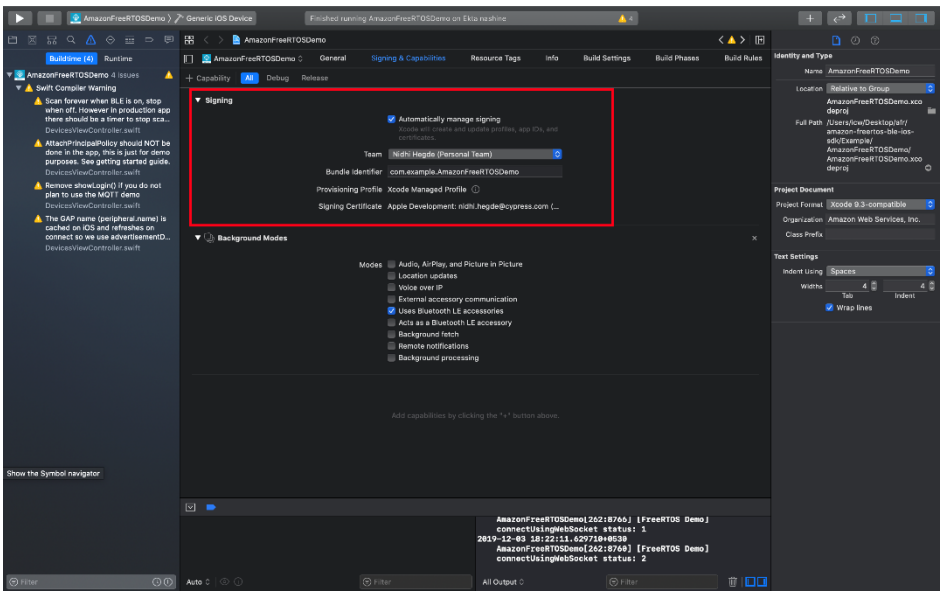
   - PoolId: Your Amazon Cognito identity pool ID.
   - Region: Your AWS Region

   Under CognitoUserPool, redefine the following variables:

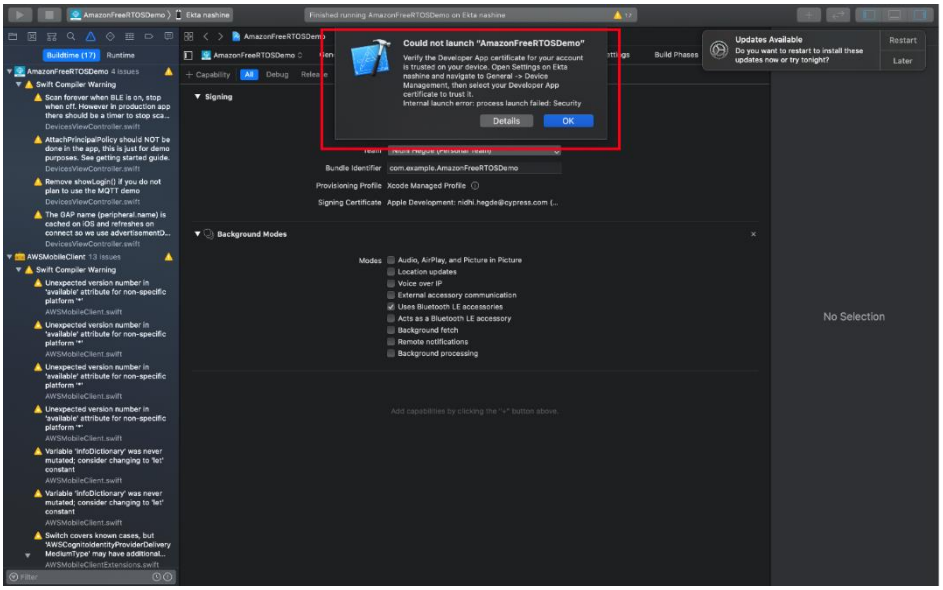
   - PoolId: Your Amazon Cognito user pool ID.
   - AppClientId: Your app client ID.
   - AppClientSecret: Your app client secret.
   - Region: Your AWS Region.

10. Go to Signing and capabilities. Select Automatically manage signing. Select the team, in this case your personal team. Add a bundle identifier.

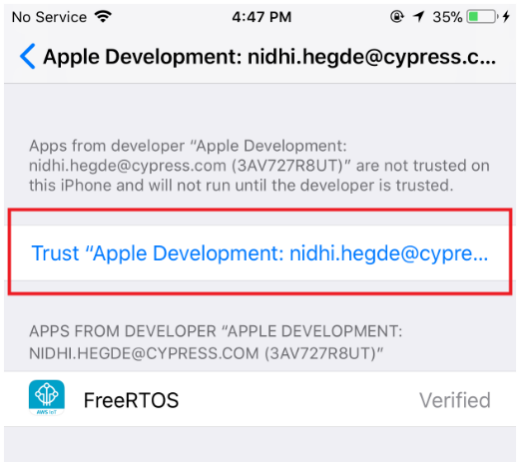
    For example: com.example.AmazonFreeRTOS



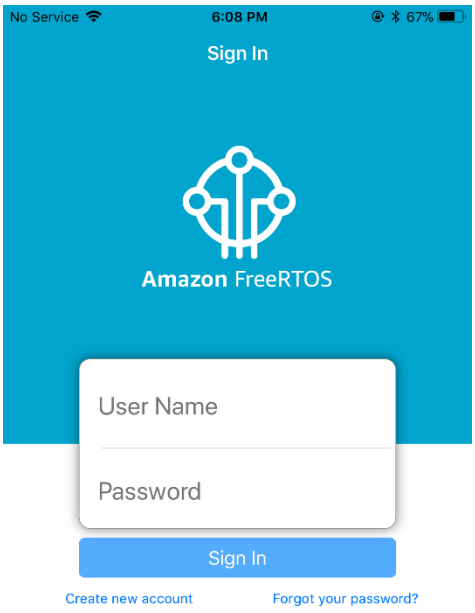
11. Build the code (menu bar > product > build). The build should succeed.
12. Next, connect your iPhone and mac with a data cable. Now run the code(menu bar > Product > Run).
13. When you try to run the app for the first time, you get a pop up saying 'could not launch AmazonFreeRTOSDemo'.



To resolve this, in the iPhone device, go to Settings > General > Device Management > Select the FreeRTOS developer app and click on 'Trust Apple Development'.



14. Run the code again from Xcode. This time the iOS FreeRTOS app will open. Create a new account and login. The app is ready to be used.



6.13.4 Building the Android App

Before starting, You need to complete both AWS IoT Setup and AWS Cognito Setup, as well as have the following ready:

- AWS IoT policy name
- Region name
- Identity Pool ID
- User pool ID
- App client ID
- App client secret

Prerequisites

- An Android phone with Android 6.0 (Marshmallow) or later
- Android Studio with Android 6.0 (Marshmallow) or later (API Level 23 or later) SDK installed

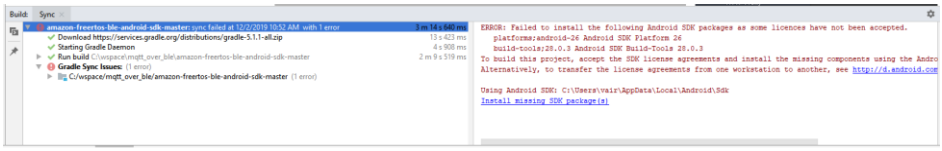
If you are using iOS, skip to Building the iOS App.

Steps

1. Download and install Android Studio from https://developer.android.com/studio
2. Before opening Android Studio, ensure you are connected to a network that allows Android Studio to download the SDK
3. Start Android Studio after the installation is complete
4. Choose Do not import settings and click OK
5. Click Don't send when Data Sharing dialog appears
6. In the Welcome page of Android Studio Setup Wizard, click Next.
7. Under Install Type, choose Standard and click Next.
8. Under Select UI Theme, select a theme (Darcula or Light) and click Next.
9. Click Finish in the next page and wait until Android Studio downloads the SDK.
10. Click Finish once the SDK installation is complete.
11. Clone or download the Amazon FreeRTOS BLE SDK for Android from https://github.com/aws/amazon-freertos-ble-android-sdk
12. In Android Studio, click Open an existing Android Studio project and select the director where Amazon FreeRTOS BLE SDK for Android is located, and click OK.
13. Once the project is opened in Android Studio, wait until the Gradle build task finishes. You can view the progress in the Build window at the bottom.

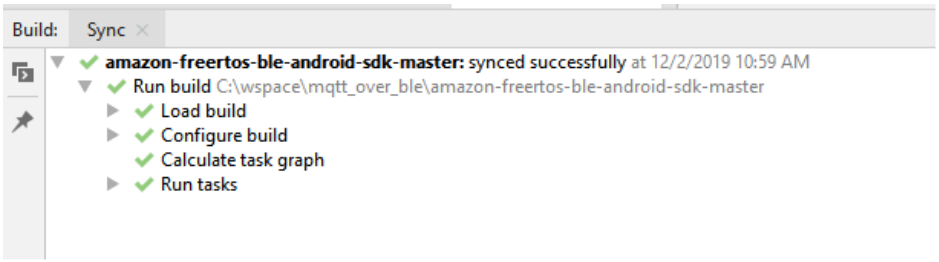
14. You may come across an error when the build finishes. Click Install missing SDK package(s) from the Build window.

    Note: This error appears because the App expects Android API Level 26 (Android 8.0 aka Oreo) to be installed but Android Studio installs only the API Level 29 (Android 10.0 aka Q) by default.

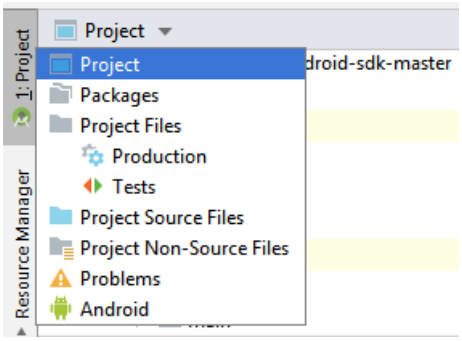


15. Accept the license and click Next. Wait until the SDK manager downloads and installs the SDK

16. Once the installation is complete, the Build task runs again, and it should succeed as shown in the image below.



17. Ensure you have selected Project view in the Project Explorer.



18. Expand app > src > main > java > software.amazon.freertos.demo and open DemoConstants.java file.

19. Update AWS_IOT_POLICY_NAME to the name of the IoT policy (e.g. MyIotPolicy) that you created under AWS IoT Setup and update AWS_IOT_REGION to the region name (e.g. us-west-2) you noted in the same section.

    Note: The policy name is the name of the IoT policy that you created under IoT Core > Secure > Policies, not the policy that you created under IAM > Roles under Attach An IAM Policy.

```
package software.amazon.freertos.demo;

public class DemoConstants {
    /*
     * Replace with your AWS IoT policy name.
     */
    final static String AWS_IOT_POLICY_NAME = "MyIotPolicy";
    /*
     * Replace with your AWS IoT region, eg: us-west-2.
     */
    final static String AWS_IOT_REGION = "us-west-2";
}
```

20. Expand app > src > main > res > raw and open awsconfiguration.json file.
21. Update
    the PoolId & Region under CognitoIdentity and PoolId, AppClientId, AppClientSecret, and Region under CognitoUserPool with the data you noted in the section AWS Cognito Setup.
22. Enable Developer Options in your Android phone. The procedure may be different for you phone. Usually, you can enable this by tapping 7 times on the Build number under Settings > About phone.
23. Now, you'll see Developer Options appearing under Settings or under Settings > System. Scroll down and enable USB debugging.
24. Connect your phone to your computer.
25. In Android Stuido, choose Build > Make Module 'app' and wait until the build finishes.The app-debug.apk is generated inside /app/build/outputs/apk/debug directory.
26. Click Run > Run 'app'
27. The app opens up in your phone after the installation finishes. You can remove the USB cable at this point.

6.13.5 Running MQTT over BLE Demo

This section provides step-by-step instructions to run the MQTT over BLE demo bundled with Amazon FreeRTOS (AFR). This demo application shows how to make an MQTT connection to AWS cloud over BLE and perform publish & subscribe operations. Amazon provides a mobile app for both Android and iOS that acts as the MQTT proxy between the device running AFR and the AWS cloud.

Before proceeding, you must familiarize yourself with Amazon FreeRTOS solution for CY8CPROTO-062-4343W. To do this, you must follow the instructions in Getting Started with Amazon FreeRTOS and CY8CPROTO-062-4343W guide and run the MQTT demo.

Prerequisites

- CY8CPROTO-062-4343W kit
- USB-A to Micro-B USB cable
- ModusToolbox IDE 2.0 b1703
- AFR 201908.00 release
- A Smart phone:
    - An iPhone, or
    - An Android phone with Android 6.0 (Marshmallow) or later

Build the iOS App

See Building the iOS App section.

Build the Android App

See Building the Android section.

Apply Patch

This step is required to fix the current issues with 201908.00 release. Extract the zip file, copy the contents of the patch directory, and paste into amazon-freertos directory.

Following is the list of files in this patch. Except the files 1 and 2, the file #3 is from this GitLab Merge Request and other files are from this GitLab Merge Request.

- demos/ble/iot_ble_numericComparison.c
- vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/application_code/main.c
- vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/iot_config.h
- vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_demo_config.h
- vendors/cypress/bluetooth/bluetooth.FreeRTOS.ARM_CM4.release.a
- vendors/cypress/bluetooth/shim.FreeRTOS.ARM_CM4.release.a

- vendors/cypress/bluetooth/psoc6/cyosal/src/cybt_osal_amzn_freertos.c
- vendors/cypress/boards/CY8CPROTO_062_4343W/ports/ble/bt_hal_internal.h
- vendors/cypress/boards/CY8CPROTO_062_4343W/ports/ble/iot_ble_hal_gatt_server.c
- vendors/cypress/boards/CY8CPROTO_062_4343W/ports/ble/iot_ble_hal_manager.c
- vendors/cypress/boards/CY8CPROTO_062_4343W/ports/ble/ iot_ble_hal_manager_adapter_ble.c

Configure Amazon FreeRTOS

1. In main.c, comment out vDevModeKeyProvisioning() inside vApplicationDaemonTaskStartupHook() function. The file is located at:

   vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/application_code/

   This is required to avoid error in parsing the client certificate and private key since we do not configure them for this demo.

2. In aws_clientcredential.h file located inside demos/include directory, update the following.

   - clientcredentialMQTT_BROKER_ENDPOINT[] to the endpoint you noted in AWS IoT Setup
   - clientcredentialIOT_THING_NAME to the name of thing you created in AWS IoT Setup

   Note: The thing name need not match the name of the thing that you setup via AWS IoT. Other settings such as the client certificate and private key are not required since the mobile app establishes MQTT connection with AWS and it handles the authentication.

3. Go to vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files directory.

4. In aws_demo_config.h file:

   - Ensure #define CONFIG_MQTT_DEMO_ENABLED is present.
   - Define only the BLE network for democonfigNETWORK_TYPES.

   ```
   #define democonfigNETWORK_TYPES     ( AWSIOT_NETWORK_TYPE_BLE )
   ```

5. In aws_iot_network_config.h file, update configENABLED_NETWORKS to contain only the BLE network:

   ```
   #define configENABLED_NETWORKS      ( AWSIOT_NETWORK_TYPE_BLE )
   ```

6. In iot_ble_config.h file, define the following above the statement #include "iot_ble_config_defaults.h".

```
/* Device name for this peripheral device. */
#define IOT_BLE_DEVICE_COMPLETE_LOCAL_NAME
"YOUR_CYPRESS_INITIALS"     //The device name is limited to 4 characters.
#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )
#define IOT_BLE_ENABLE_WIFI_PROVISIONING ( 1 )
#define IOT_BLE_ENABLE_BONDING (0)
#define IOT_BLE_ENCRYPTION_REQUIRED (0)
```

Note: Bonding and Encryption are disabled due to an issue with pairing. See Known Issues for details.

7. All the kits use same BLE address (43:43:A1:12:1F:AC) by default. You must change it to uniquely identify your kit when multiple devices are present. To change the address:
   a. Go to libraries/c_sdk/standard/ble/src/iot_ble_gap.c file
   b. Add const uint8_t bdAddr[] = {0x11, 0x22, 0x33, 0xAA, 0xBB, 0xCC}; above const BTProperty_t _deviceProperties[] =. Change the address in the array as you like.
   c. In _deviceProperties[] array, add a comma after the last element and add the following.

```
{
.xType = eBTpropertyBdaddr,
.xLen = 6,
.pvVal = (void *)bdAddr
}
```

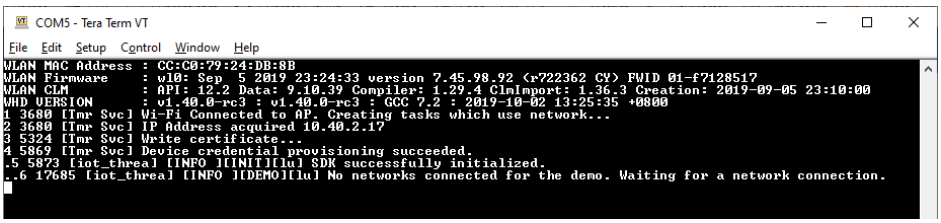
8. In FreeRTOSConfig.h, change configLOGGING_MAX_MESSAGE_LENGTH to 200

```
#define configLOGGING_MAX_MESSAGE_LENGTH            200
```

Build and Program

1. Import the project into ModusToolbox IDE from projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos directory
2. Ensure the kit is connected over USB.
3. Open a serial terminal software and open a connection to the kit's COM port with 115200, 8N1 setting.
4. Click aws_demos Program (KitProg3) from the Quick Panel to build the project and program the kit.

5. The device boots up, starts advertisement, and waits until a BLE connection is established. The console output appears as follows.

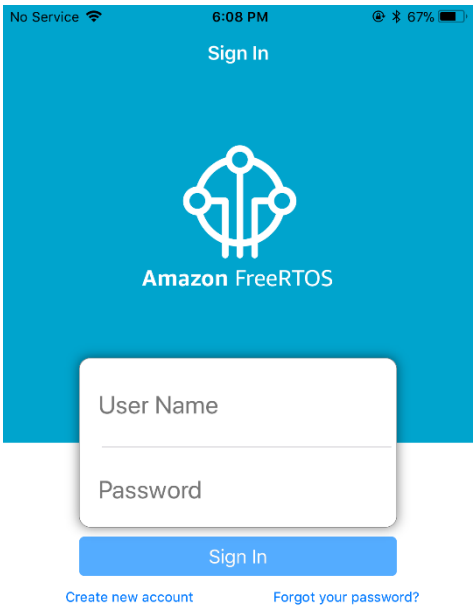


Run Using the iOS App

If you are using Android, skip to Run Using the Android App. Tested on iOS version 11.1/iPhone 6 Plus. See Building the iOS App to build and install the app in your phone.

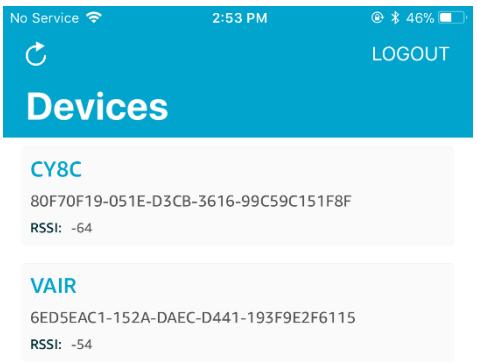
1. Open the FreeRTOS app
2. Click Create new account, enter the account information, and click Sign Up. For the Email filed, you can enter your Cypress email ID. You'll receive the confirmation code to this email ID.

   Note: This account is not your AWS account. The account you setup here will be registered under the user pool that you setup during AWS Cognito Setup.



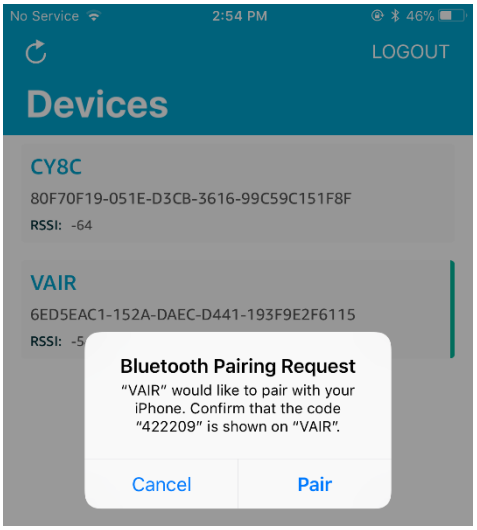
3. In the next page, enter the CONFIRMATION CODE received in your email and click Confirm. You'll see Registration Complete message and click OK.

4. Now the app signs you in, scans for the BLE devices, and lists them. You can click the refresh icon at top left corner to scan again.

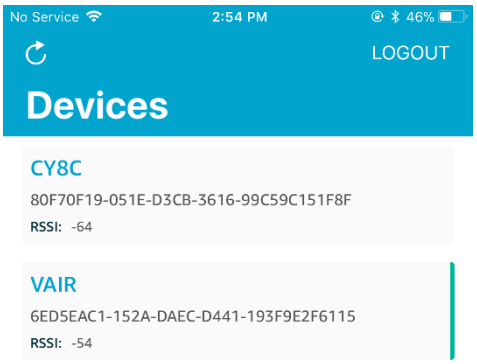


5. Touch the device in the list to connect to it.

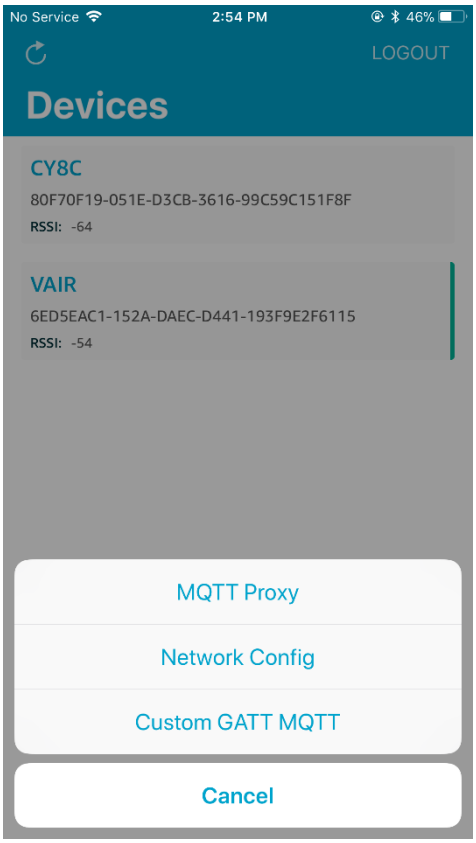
6. Device sends pairing request to your phone. Click Pair when the message appears in the app. You can now see the device listed under MY DEVICES in Settings > Bluetooth.



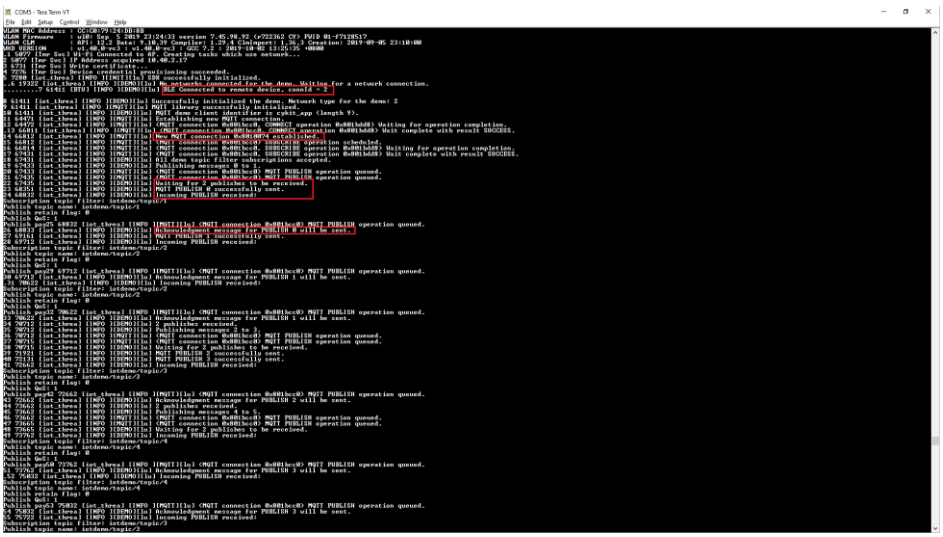
7. Once the connection is successful, the device has a vertical green line at the right corner. You can swipe left on the device to disconnect.



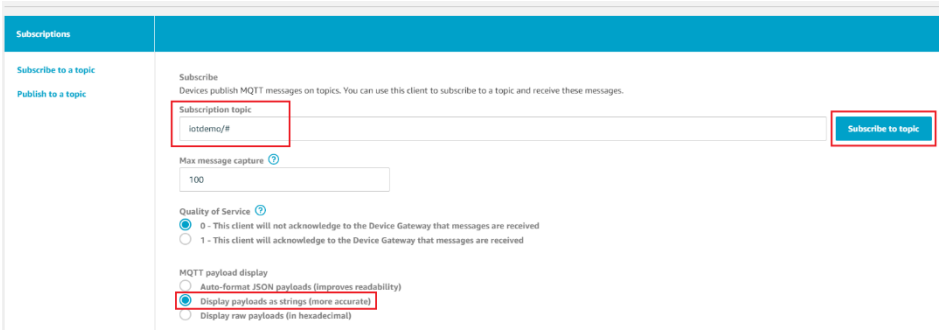
8. Tap on the device and choose MQTT Proxy.



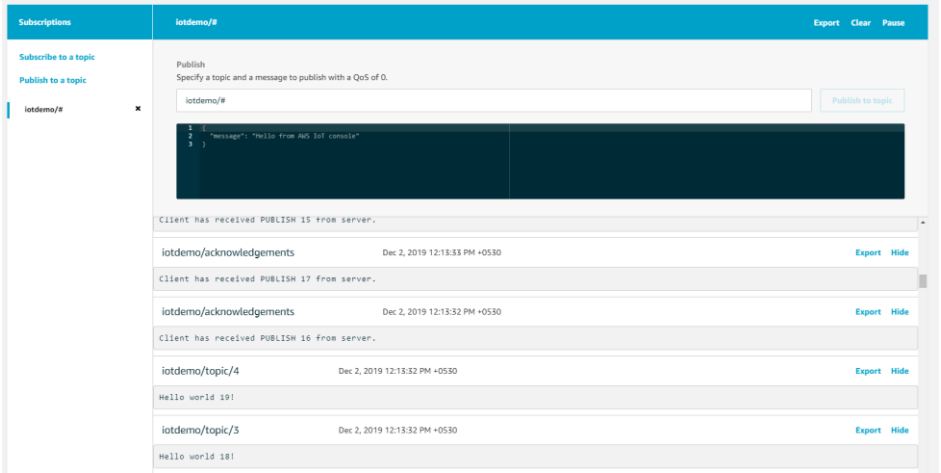
9. Now observe the UART terminal output. The device should be able to establish MQTT connection and publish the data. If you see any error that the MQTT connection failed, reset the kit and repeat the steps.



10. You can also view the published messages in the AWS IoT Test client. To view the Test client, go to AWS Services > IoT Core and click Test from the left navigation pane.
11. Enter iotdemo/# under Subscription topic, choose Display payloads as strings, and click Subscribe to topic.



12. Sample output in the Test client is shown below. The device publishes up to 20 (0 to 19) "Hello World n!" messages where n = 0 to 19.



Run Using the Android App

If you are using iOS, skip to Run Using the iOS App. Tested on iOS version 11.1/iPhone 6 Plus. See Building the Android App to build and install the app in your phone.

Notes:

- Unlike the iOS app, the Android app does not work seamlessly. You may have to close the app, clear the data/cache, and turn off/on bluetooth on the phone, reset the kit, and repeat the steps as necessary.
- Ensure that your phone is connected to a network that allows MQTT traffic. Some mobile operators seem to block it.

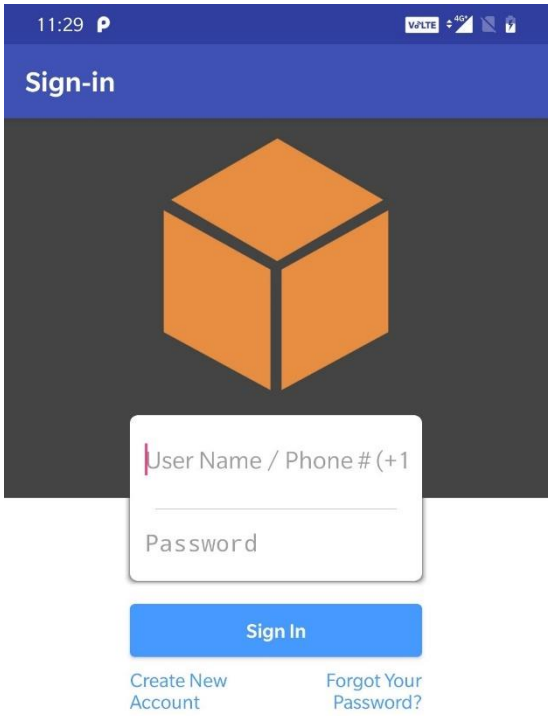
- You can keep the phone connected to your computer and open Logcat windows at the bottom of Android Studio to view log messages that provides a better picture of the progress.

Tested on One Plus 6T/Oxygen OS 9 aka Android 9 Pie.

1. Open the AmazonFreeRTOS Demo app
2. In the sign-in screen that appears, click Create New Account, enter the details, and click Sign Up and wait until the next page appears.

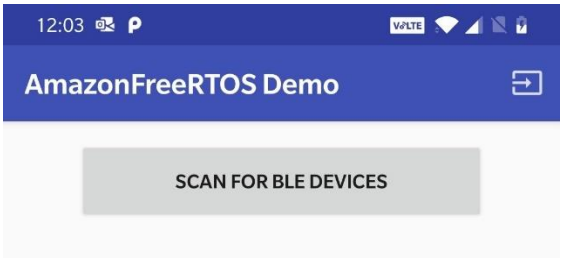
   For the Email field, you can enter your Cypress email id. You'll receive the confirmation code to this email id.

   Note: This account is not your AWS account. The account you setup here will be registered under the user pool that you setup during AWS Cognito Setup.

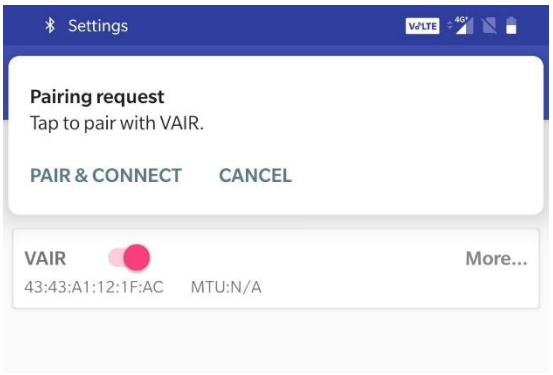


3. In the next page, enter the CONFIRMATION CODE that you received in the email and click OK.
4. Wait until the app signs you in. This might take a while. Once the sign in is successful, you'll get a screen with SCAN FOR BLE DEVICES button.
5. Allow when the app requests access to location.

6. Click the SCAN FOR BLE DEVICES button and wait until the device appears.



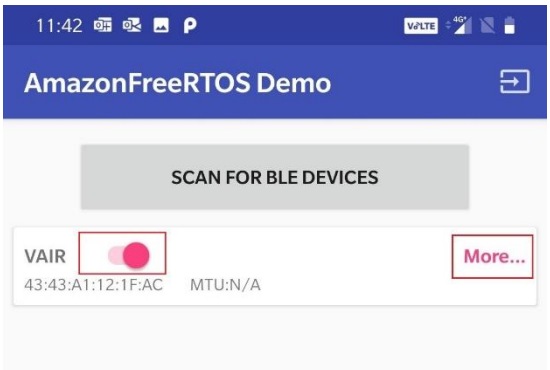
7. Wait for about 30 seconds for the scan to finish and then Toggle the switch on the device to establish connection.
8. You'll get a pairing request and click PAIR & CONNECT. The pairing request is in the notification center. Slide down from top to access the pairing request in case if you have missed it.



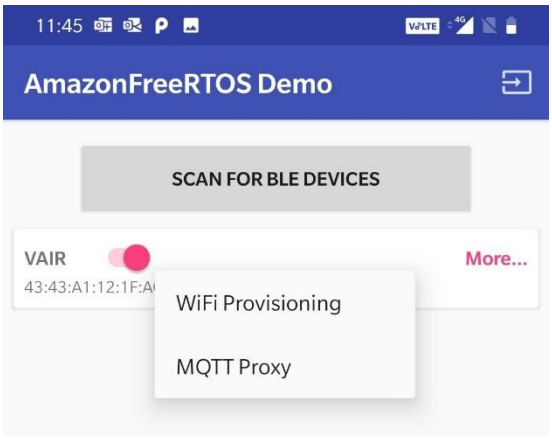
9. When the Pairing Code (passkey) appears, click Pair.

   Note: The getUserMessage() function needs to be implemented to get the user input and verify it against the incoming passkey. This function is not implemented in the 201908.00 release. Instead, the modified function userInputTask() in /demos/ble/iot_ble_numericcomparison.c file simply accepts the passkey without waiting for user input.

10. When the connection is successful, the app enables the More... menu for the device.



11. Touch More... and choose MQTT Proxy



12. Now observe the UART terminal output. The device should be able to establish MQTT connection and publish the data. If you see any error that the MQTT connection failed, reset the kit and repeat the steps.

    Note: Once the app connects to the kit successfully, it automatically establishes connection upon disconnection and the whole MQTT over BLE flow just runs through without having to repeat the steps. You need to just reset the kit and wait until the app reconnects.

13. See the last few steps (8 to 11) under Run Using the iOS App to see the sample terminal output and how to monitor the MQTT data using the AWS IoT Test client.

Known Issues

Issue #1

"Error running demo" message appears at the end of MQTT over BLE demo

Root cause: The demo code publishes acknowledgement messages under the topic "iotdemo/acknowledgements" for each received message. It initiates publishing of the last acknowledgement message which is a non-blocking function call and initiates UNSUBSCRIBE immediately without waiting for the transaction to complete. The MQTT proxy (mobile app) returns error in this case and the final status results in error. This is reason you'll see all 20 "Hello World n!" messages published in the AWS Test client but only few messages under the topic "iotdemo/acknowledgements".

Fix: This can be solved by adding a small delay at the end of the function _mqttSubscriptionCallback().

```
            else
            {
                IotLogWarn( "Acknowledgment message for PUBLISH %.*s will NOT be sent.",
                            ( int ) messageNumberLength,
                            pPayload + messageNumberIndex );
            }
        }
    }

    vTaskDelay(pdMS_TO_TICKS(1000));

    /* Increment the number of PUBLISH messages received. */
    IotSemaphore_Post( pPublishesReceived );
}
```

Issue #2

With iOS, MQTT over BLE demo works only when the device is paired for the 1st time. From the next time, device is unable to establish MQTT connection though BLE connection is successful.

With Android, pairing request is received every time device makes BLE connection with the phone but the MQTT over BLE demo works without any issues.

Root cause: TBD

Fix: TBD

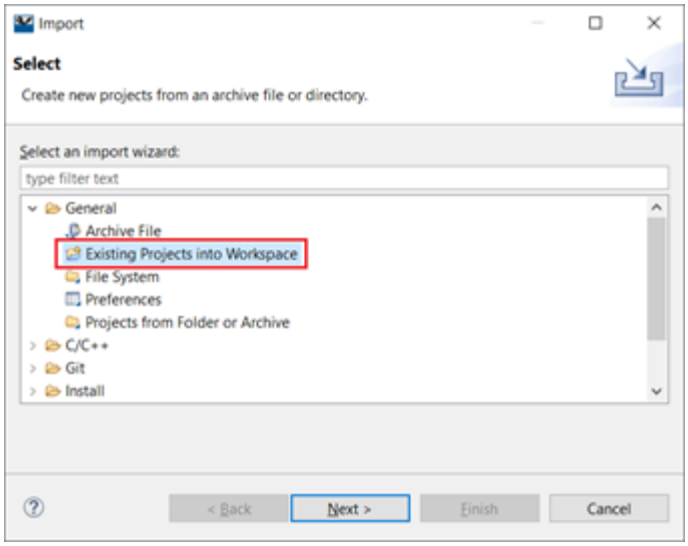
Workaround: Disable bonding and encryption by defining the following macros in vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/iot_ble_config.h file.

```
#define IOT_BLE_ENABLE_BONDING (0)
#define IOT_BLE_ENCRYPTION_REQUIRED (0)
```

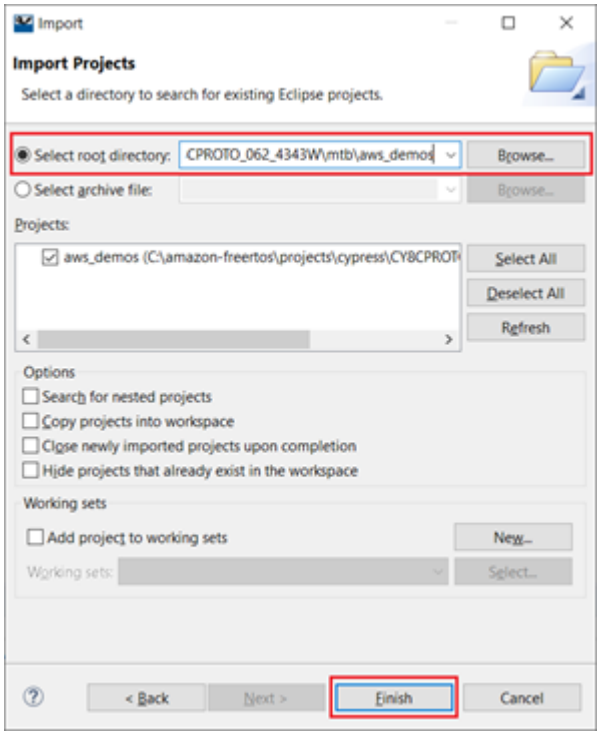
6.13.6 Running BLE GATT Server Demo

Note: If you have not done AWS IOT Setup and AWS Cognito Setup, do them now. After setting up your AWS account, to run Amazon FreeRTOS BLE Custom GATT Server demo, follow the steps below.

1. Clone the amazon-freertos project from the github link. Open git bash and type:
   git clone https://github.com/cypresssemiconductorco/amazon-freertos.git (to clone with http)
2. Add these patch files to the cloned project: ble_patch_201908.00
3. Import the project into ModusToolbox IDE.
   a. Open the ModusToolbox IDE and choose or create a workspace.
   b. Select File > Import. In the window that appears, expand General, choose Existing Projects into Workspace, and then click Next.



   c. In Select root directory, enter <amazon-freertos>/projects/cypress/ CY8CPROTO_062_4343W /mtb/aws_demos.

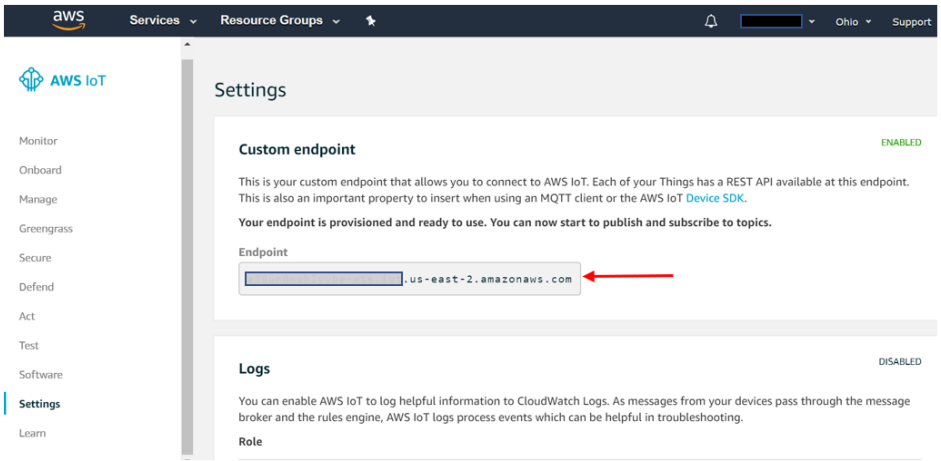


d. Click Finish to import the project into your workspace.

4. After importing, go to the file demos/include/ aws_clientcredential.h and configure MQTT broker endpoint and IOT thing name by changing values of below mentioned macros:

```
#define clientcredentialMQTT_BROKER_ENDPOINT          " "
#define clientcredentialIOT_THING_NAME                      " "
```

Note: To get the thing name, login to IOT console with your aws account. Go to Services > IOT Core. From the left menu bar, select Manage , and here you can find the "Thing" created. To get MQTT broker endpoint, from the left menu bar, go to Settings. Here you can find the endpoint:



5. Next, go to file vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/iot_ble_config.h

    a. Add the macro:

    ```
    #define IOT_BLE_ADD_CUSTOM_SERVICES                    ( 1 )
    ```

    b. The device advertising name can be changed using:

    ```
    #define IOT_BLE_DEVICE_COMPLETE_LOCAL_NAME             "NAME"
    ```

    Note: The device name can have a maximum of 4 characters.

6. Go to file vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_iot_network_config.h

    a. Change the value of macro as shown:

    ```
    #define configENABLED_NETWORKS      ( AWSIOT_NETWORK_TYPE_BLE)
    ```

    Note: This demo does not require the kit to connect to Wi-Fi.

7. Go to vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_demo_config.h

    a. Comment `#define CONFIG_MQTT_DEMO_ENABLED`

    b. Add `#define CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`

    c. Change the macro values as shown:

    ```
    #define democonfigNETWORK_TYPES     ( AWSIOT_NETWORK_TYPE_BLE )
    ```

8. Go to vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/application_code/main.c. Inside function vApplicationDaemonTaskStartupHook(), comment the function call: `vDevModeKeyProvisioning();`

9. All the kits use same BLE address (43:43:A1:12:1F:AC) by default. You must change it to uniquely identify your kit when multiple devices are present. To change the address:

    a. Go to libraries/c_sdk/standard/ble/src/iot_ble_gap.c file

    b. Add const uint8_t bdAddr[] = {0x11, 0x22, 0x33, 0xAA, 0xBB, 0xCC}; above const BTProperty_t _deviceProperties[] =. Change the address in the array as you like.

    c. In _deviceProperties[] array, add a comma after the last element and add the following:

    ```
    {
    .xType = eBTpropertyBdaddr,
    .xLen = 6,
    .pvVal = (void *)bdAddr
    }
    ```

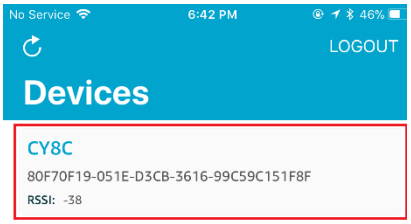
10. In FreeRTOSConfig.h, change `configLOGGING_MAX_MESSAGE_LENGTH` to 200

    ```
    #define configLOGGING_MAX_MESSAGE_LENGTH            200
    ```

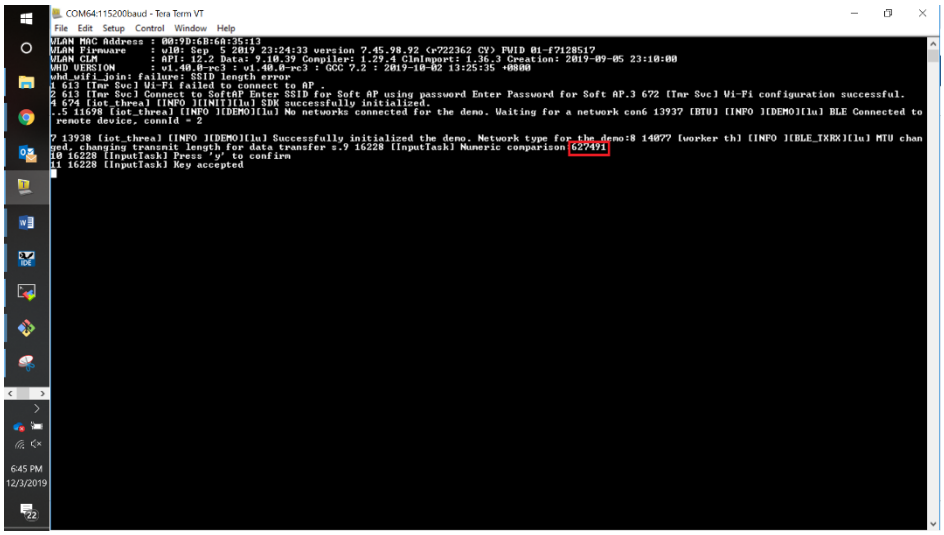
11. Build the project and program the kit.

12. Open Tera Term or any other UART terminal of your choice. Choose the COM port to to which the kit is connected with baud rate of 115200. Data is 8 bit with Stop bit as 1.

13. On the iPhone device, enable Bluetooth and open FreeRTOS app. The app starts scanning for Bluetooth devices. Select your device to connect.



14. After this, pairing request pops up in the app. Compare the numeric keys on the displayed on the terminal and iPhone device. Accept pairing request if the keys match. A green bar at the right side of device tab indicates active connection.



15. Next, select your kit in the devices list. Login to AWS IOT console and go to Services> IOT Core. Go to Tests and select Subscribe to a Topic. Enter the topic name and click Subscribe.

16. In the FreeRTOS app, select Custom GATT MQTT. Here you can start, stop or reset the counter and the values are updated to the console.

Known Issues

Issue #1

With iOS app, BLE Custom GATT server demo works for the first time. After pairing iPhone stores the BLE peripheral's (kit) information. Next time when we try to connect, pairing is skipped(as both the devices were paired for the first time and iPhone stores this info). In this case, the iOS app crashes. After connection, nothing works.

Workaround: in the iPhone, go to settings > Bluetooth > Connected Device > (our device) and click Forget this device. Do this every time to run the demo.

6.13.7 Print Scan of all the Wi-Fi access points

1. Clone the AFR repo from GitHub and import the project into a new workspace in ModusToolbox IDE.

   Use the same steps that you did previously to clone and import the AFR repo.

2. Configure the network credentials in the aws_clientcredential.h file:

   <amazon-freertos>/demos/include/aws_clientcredential.h.

   Note: This exercise does not require the kit to be connected to a Wi-Fi access point. But the demo runner flow of AFR invokes the application code after a successful connection with a Wi-Fi network. Hence you need to configure the network credentials.

   a. Enter the SSID and password of the network to be connected in the following macros:

   ```
   #define clientcredentialWIFI_SSID           "<your-ssid-here>"
   #define clientcredentialWIFI_PASSWORD       "<your-password-here>"
   ```

   b. Additionally, if your network's Wi-Fi Security is not WPA2, configure the type of Wi-Fi Security in the following macro:

   ```
   #define clientcredentialWIFI_SECURITY       eWiFiSecurityWPA2
   ```

   c. Change eWiFiSecurityWPA2 to eWiFiSecurityOpen, eWiFiSecurityWEP, or eWiFiSecurityWPA, as applicable.

3. Disable the AWS certificate provisioning in the main.c file:

   <amazon-freertos>/vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/application_code/main.c.

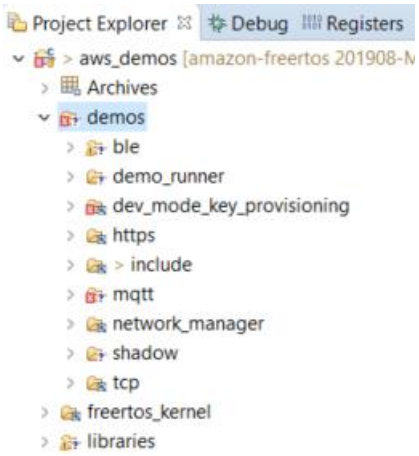
   a. Comment out the following line in the vApplicationDaemonTaskStartupHook() function:

   ```
   vDevModeKeyProvisioning();
   ```

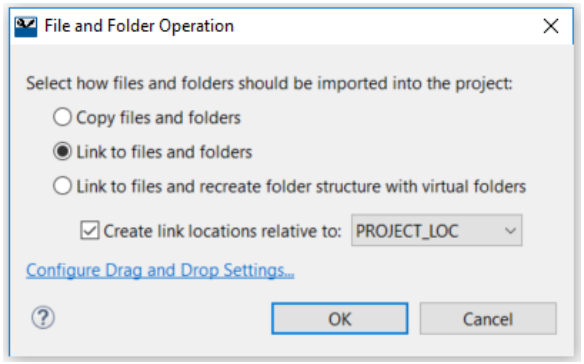
4. Copy the cy_wifi_scan folder (provided) to the <amazon-freertos>/demos directory.

   Additionally, if you want the folder to appear in the ModusToolbox IDE, perform the following steps:

a. Drag-and-drop the cy_wifi_scan folder from <amazon-freertos>/demos directory into the demos folder inside ModusToolbox IDE. Note that the demos folder inside aws_demos project in the ModusToolbox workspace as shown:



b. Choose **Link to files and folders** and ensure that **Create link locations relative to PROJECT_LOC** is checked as shown below.



5. Add the source file to the Makefile by editing the makefile:

   <amazon-freertos>/projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos/Makefile.

   Replace: `SOURCES=`

   With: `SOURCES=$(CY_AFR_ROOT)/demos/cy_wifi_scan/cy_wifi_scan.c`

6. Add the entry function macro in the <amazon-freertos>/demos/include/iot_demo_runner.h file.

```
#elif defined( CONFIG_WIFI_SCAN_DEMO_ENABLED )
#define DEMO_entryFUNCTION              RunWifiScanDemo
```

Paste the above two lines just above #else as shown:

```
#elif defined( CONFIG_HTTPS_SYNC_DOWNLOAD_DEMO_ENABLED )
    #define DEMO_entryFUNCTION              RunHttpsSyncDownloadDemo
#elif defined( CONFIG_HTTPS_ASYNC_DOWNLOAD_DEMO_ENABLED )
    #define DEMO_entryFUNCTION              RunHttpsAsyncDownloadDemo
#elif defined( CONFIG_WIFI_SCAN_DEMO_ENABLED )
    #define DEMO_entryFUNCTION              RunWifiScanDemo
#else /* if defined( CONFIG_MQTT_DEMO_ENABLED ) */
/* if no demo was defined there will be no entry point defined and we will not be able to run the demo */
    #error "No demo to run. One demo should be enabled"
#endif /* if defined( CONFIG_MQTT_DEMO_ENABLED ) */
```

7. Make the following edits to the <amazon-freertos>/vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_demo_config.h file.

    a. Comment out the line:

    ```
    #define CONFIG_MQTT_DEMO_ENABLED
    ```

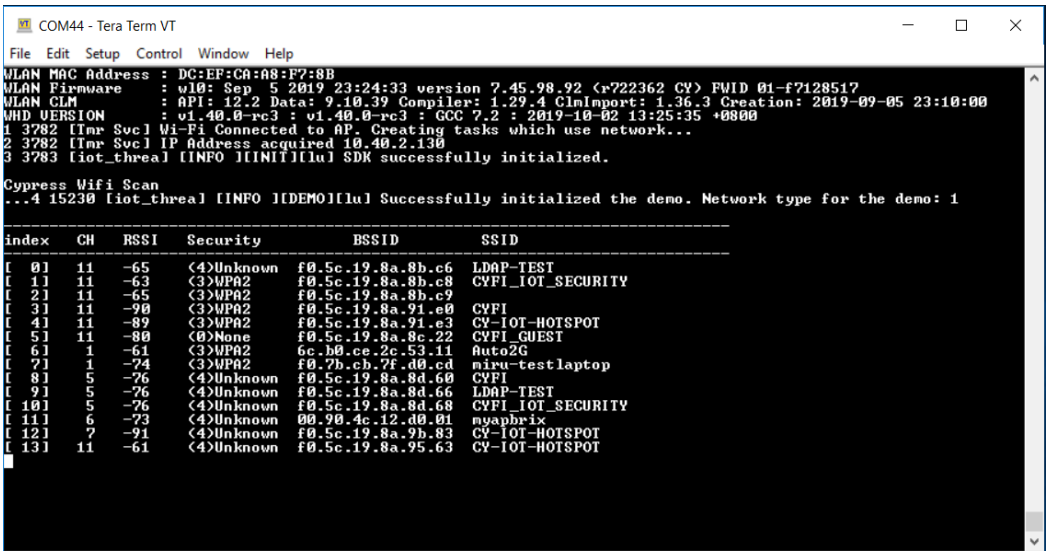
    b. Add the following line below the commented-out line.

    ```
    #define CONFIG_WIFI_SCAN_DEMO_ENABLED
    ```

8. Build the project and program the kit.

9. Observe the UART logs in a serial terminal program of your choice by connecting to the COM port of the device with a baud rate of 115200 and 8N1 configuration.

    The following shows a sample log:



6.13.8 Display using TFT

Make a program that displays on the TFT a counter that updates 1/second. Create a new task to manage the TFT display and update.

6.13.9 Modify the MQTT Demo with Display

Update the demo project to display the number of publish messages received.

6.13.10 Simple MQTT over Wi-Fi

The following are the differences between this demo and the MQTT demo provided with AFR that we looked at previously:

- This demo does not use DEMO_RUNNER (demos/demo_runner), but reuses most of the functions from it.
- The demo code does not span into multiple files. The entire demo code is in one file: demos/my_mqtt/my_mqtt_demo.c.
- It publishes only one message ("Hello world 1!") on the topic "iotdemo/topic/1".
- It does not subscribe to any topic and does not register a callback to be called when publish operations complete. That is:

```
publishComplete.function = NULL;
```

Steps to Follow

1. Clone the AFR repo from GitHub and import the project into a new workspace in ModusToolbox IDE

   Use the same steps that you did previously to clone and import the AFR repo. You must use a new workspace since the project name will be the same as the one used in previous exercises. Remember, the path is: amazon-freertos/projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos

2. Edit aws_clientcredential.h and aws_clientcredential_keys.h.

   Enter the correct Wi-Fi settings and AWS credentials.

3. Copy the folder my_mqtt into <amazon-freertos>/projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos/.

4. Right click on the project and choose Refresh if you don't see the folder inside ModusToolbox IDE after copying it.

   Alternately, you can place the my_mqtt folder anywhere inside the <amazon-freertos> folder. In that case, you must include the demo.mk file with its full path into the Makefile. For example, if my_mqtt is in the <amazon-freetos>/demos folder, you would put this in the Makefile below the line include ./afr.mk:

   include $(CY_AFR_ROOT)/demos/my_mqtt/demo.mk

   Note: You don't have to include the .mk file into Makefile when you place the source files under <amazon-freertos>/projects/cypress/ CY8CPROTO_062_4343W/mtb/aws_demos folder because this folder is included for auto-discovery by the Make build system. Other folder ssuch as <amazon-freertos>/demos are not included for auto-discovery.

5. Edit <amazon-freertos>/vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/application_code/main.c.

   In the vApplicationDaemonTaskStartupHook() function, comment out:
   ```
   DEMO_RUNNER_RunDemos();
   ```

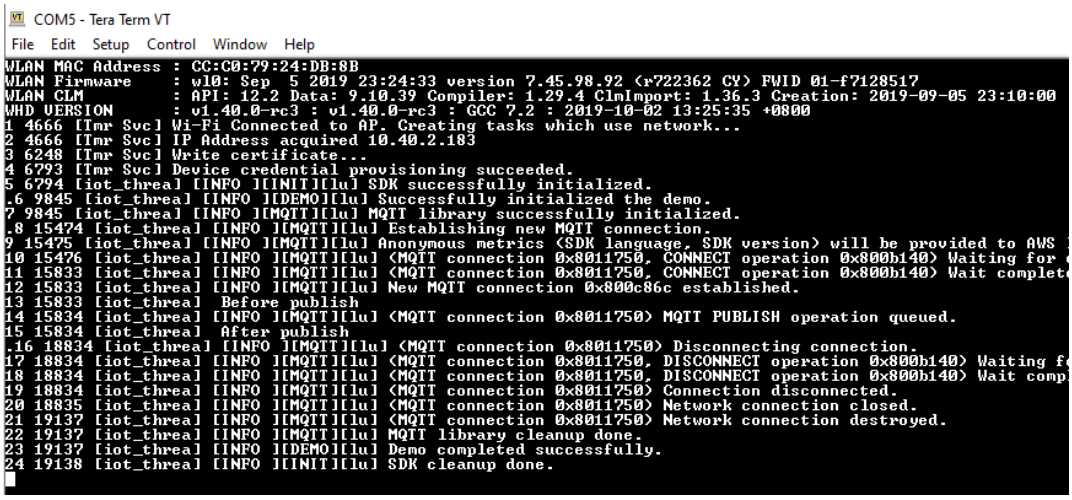
   Add the line:
   ```
   init_mqtt_demo();
   ```

6. Edit <amazon-freertos>/vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/FreeRTOSConfig.h.

   ```
   Set configLOGGING_MAX_MESSAGE_LENGTH 200
   ```

7. Connect the kit, build and program.

   Expected serial output is:



Source Code Description

In the file my_mqtt_demo.c, two public functions init_mqtt_demo(), run_mqtt_demo(), and three internal functions _initialize (), _run_mqtt_publish_subscribe(), _cleanup() implement the demo.

Public Functions

init_mqtt_demo()

This function just creates a task which will be run after the scheduler starts. The task runs run_mqtt_demo() (see below).

Is uses Iot_CreateDetachedThread() – a platform API provided by AFR, which in turn calls xTaskCreate(). Additionally, Iot_CreateDetachedThread() deletes the task after the task exits.

It could use xTaskCreate() instead of Iot_CreateDetachedThread().

run_mqtt_demo()

This function is a simpler version of the runDemoTask() function from /demos/demo_runner/iot_demo_freertos.c

It performs the following 3 steps:

_initialize() – Initializes the network manager
_run_mqtt_publish_subscribe() – Creates an MQTT connection and publishes a message
_cleanup() – Cleans up the resources used during the network manager initialization

Note: It is not possible to bypass the network manager even though the Wi-Fi connection is established as part of prvWifiConnect() in main.c because the MQTT connection set up requires various info such as network info, credentials etc. which are supplied by the network manager through simple function calls. See IotMqtt_Connect() in _run_mqtt_publish_subscribe().

Internal Functions

_initialize()

This function is an exact copy of the same function from /demos/demo_runner/iot_demo_freertos.c

It performs the following 5 steps:

IotSdk_Iit()
AwsIotNetworkManager_Init()
AwsIotNetworkManager_SubscribeForStateChange()
A simpler callback is used that just posts a semaphore when network is enabled.
AwsIotNetworkManager_EnableNetwork()
Wait for the configured network to be initialized by waiting on a semaphore
IotSemaphore_Wait(&demoNetworkSemaphore);

_cleanup()

This function is an exact copy of the same function from /demos/demo_runner/iot_demo_freertos.c.

It unregisters the network state change subscription and destroys the semaphore.

_run_mqtt_publish_subscribe()

This function performs the following 5 steps:

IotMqtt_Init()
IotMqtt_Connect()
Requires IotMqttNetworkInfo_t and IotMqttConnectInfo_t structures
IotMqtt_Publish()
Requires IotMqttPublishInfo_t and IotMqttCallbackInfo_t structures
IotMqtt_Disconnect()
IotMqtt_Cleanup()

6.13.11 Simple MQTT over BLE

1. Clone the AFR repo from GitHub and Import the project into a new workspace in ModusToolbox IDE

   Use the same steps that you did previously to clone and import the AFR repo. You must use a new workspace since the project name will be the same as the one used in previous exercises. Remember, the path is:
   amazon-freertos/projects/cypress/CY8CPROTO_062_4343W/mtb/aws_demos

2. Edit vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_iot_network_config.h.

   Change configENABLED_NETWORKS to:
   ```
   #define configENABLED_NETWORKS ( AWSIOT_NETWORK_TYPE_BLE |
   AWSIOT_NETWORK_TYPE_WIFI )
   ```

3. Edit vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/iot_ble_config.h

   Change the value for IOT_BLE_DEVICE_COMPLETE_LOCAL_NAME from "CY8C" to your initials. This is necessary so that you will be able to find your device. Note that only the first 4 characters of the name are included in the advertising packet.

4. Edit libraries/c_sdk/standard/ble/src/iot_ble_gap.c

   Change the _advParams as follows (changes shown in **bold**):
   ```
   static IotBleAdvertisementParams_t _advParams =
   {
       .includeTxPower     = true,
       .name               = { BTGattAdvNameShort,
                               IOT_BLE_DEVICE_SHORT_LOCAL_NAME_SIZE},
       .setScanRsp         = false,
       .appearance         = IOT_BLE_ADVERTISING_APPEARANCE,
       .minInterval        = IOT_BLE_ADVERTISING_CONN_INTERVAL_MIN,
       .maxInterval        = IOT_BLE_ADVERTISING_CONN_INTERVAL_MAX,
       .serviceDataLen     = 0,
       .pServiceData       = NULL,
       .manufacturerLen    = 0,
       .pManufacturerData  = NULL,
       .pUUID1             = ( BTUuid_t * ) &_advUUID,
       .pUUID2             = NULL
   };
   ```

5. Edit libraries/c_sdk/standard/ble/include/iot_ble_config_defaults.h.

   Update the following macros:
   ```
   #define IOT_BLE_PREFERRED_MTU_SIZE            ( 185 )
   #define IOT_BLE_ENCRYPTION_REQUIRED           ( 0 )
   #define IOT_BLE_ADD_CUSTOM_SERVICES           ( 1 )
   #define IOT_BLE_ENABLE_FREERTOS_GATT_SERVICES ( 0 )  // Optional
   ```

6. Edit demos/ble/aws_ble_gatt_server_demo.c.

   Comment out the definition of the IotBle_AddCustomServicesCb() function to avoid mulitple definitions.

7. Edit vendors/cypress/boards/CY8CPROTO_062_4343W/ aws_demos/application_code/main.c.
    a. Add the line `#include "ble_gatt_server.h".`
    b. To run this demo, we will disable the Wi-Fi functionality. It is optional but allows us to get rid of Wi-Fi related configurations.
    c. Comment out the call to prvWifiConnect(); in the vApplicationDaemonTaskStartupHook function.
    d. Comment out the call to vDevModeKeyProvisioning(); in the vApplicationDaemonTaskStartupHook function.
    e. Disable the existing demo by commenting out DEMO_RUNNER_RunDemos(); in the vApplicationDaemonTaskStartupHook function.
    f. Add the following line to the vApplicationDaemonTaskStartupHook function: `ble_gatt_demo_runner();`

    Note: The main .c file is in the ble_gatt_server_demo folder. You can replace the main.c with the modified main.c file.

8. Copy ble_gatt_sever.c and ble_gatt_server.h from ble_gatt_server_demo/ to project/cypress/CY8CPROTO_062_4343W/mtb/aws_demos/
9. Build and Program.
10. Connect the CySmart Dongle to the PC and open the CySmart Application.
    a. Click on **Configure Mater Settings > Scan Parameters**.
    b. Change Scan Interval to 1000ms and Scan Window to 100ms and click **OK**.
    c. This is done to prevent the application from stalling when there are many devices advertising at once. Start Scanning.
11. The App lists the BLE peripheral with the name your provided. Click on **Connect** and then click on **Discover All Attributes** to see all the services in the BLE peripheral.
12. Select the Alert Level Characteristic.
    a. Enter ‘1’ for the Value field and click on **Write Value Without Response**.
    b. Observe the LED turning ON.
    c. Enter ‘0’ for the Value field and click on **Write Value Without Response**.
    d. Observe the LED turning OFF.

6.13.12 Use TFT to display messages received via MQTT Subscribe

- Open mqtt connection
- Subscribe to the “topic”
- Publish to the topic via the test client
- Print to the screen

6.13.13 Run the shadow demo

1. Clone the AFR repo from GitHub and import the project into a new workspace in ModusToolbox IDE.

   Use the same steps that you did previously to clone and import the AFR repo.

2. Configure the network credentials in <amazon-freertos>/demos/include/aws_clientcredential.h.

   Use the same steps that you did previously for 6.13.7 WiFi Scan Demo

3. Configure AWS credentials.

   Enter the MQTT endpoint address and the AWS registered thing name in <amazon-freertos>/demos/include/aws_clientcredential.h.

   Input the AWS client certificate and private key in <amazon-freertos>/demos/include/aws_clientcredential_keys.h file.

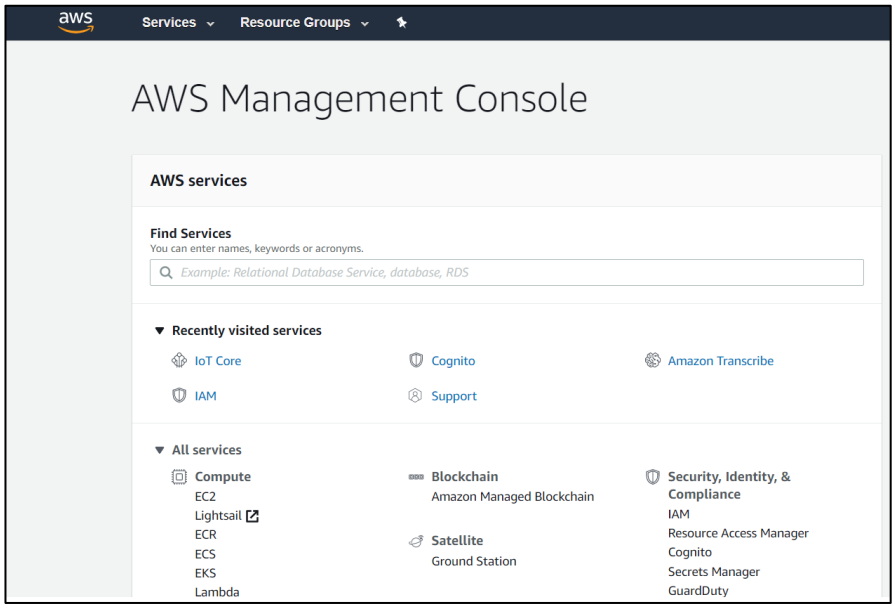
   Follow the same instructions that you used for MQTT demo.

4. Enable the MQTT Shadow demo in the <amazon-freertos>/vendors/cypress/boards/CY8CPROTO_062_4343W/aws_demos/config_files/aws_demo_config.h file by adding the following line. Make sure that only one demo is enabled.

   ```
   #define CONFIG_SHADOW_DEMO_ENABLED
   ```

5. Opening the AWS IoT Console

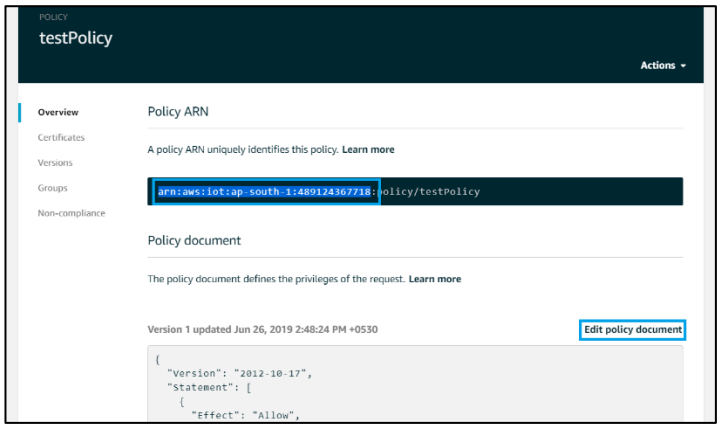
   Go to the AWS Console webpage. Once signed in using your AWS account credentials, you should be able to see a list of AWS services as shown below.



6. Click on IoT Core or type ‘IoT Core’ in the Find Services search bar.
7. Editing the policy
8. Go back to the home page of AWS IoT Console.

   From the navigation pane on the left, click on Secure > Policies. Click on the policy that is associated with the configured thing.

9. Click on **Edit policy document** as shown:



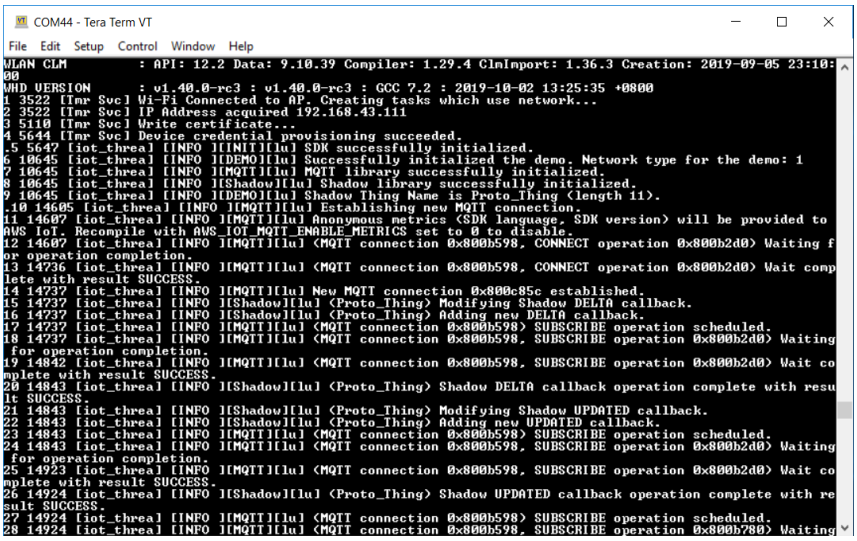
10. Replace the existing text with the following policy.

    Note: Replace the bolded text with the selected text (also shown in blue box) in the above image. Also replace <your-thing-name> with the name of the thing that you configured in aws_clientcredential.h.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:ap-south-1:489124367718:client/<your-thing-
name>"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:ap-south-
1:489124367718:topicfilter/$aws/things/Proto_Thing/shadow/*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:ap-south-
1:489124367718:topic/$aws/things/Proto_Thing/shadow/*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:ap-south-
1:489124367718:topic/$aws/things/Proto_Thing/shadow/*"
    }
  ]
}
```

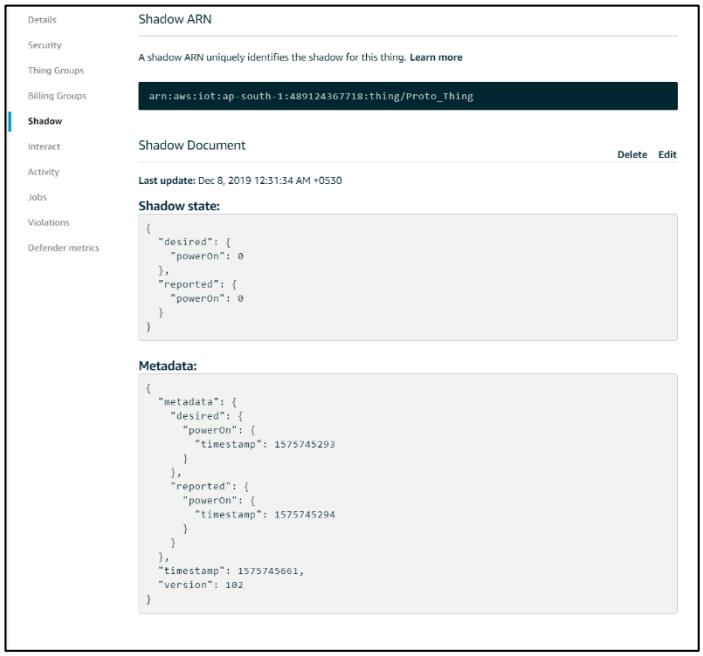
11. Build and program the project.

12. Observe the UART logs in a serial terminal program with a configuration of 115200 baud rate and 8N1. The following shows a sample log:



13. View the device's shadow in AWS IoT Console.

    From the AWS IoT Core's home page, click on **Manage > Things**. On the right, click on the thing that you configured in aws_clientcredential.h. Click on **Shadow** in the left pane. You should be able to see the shadow state and metadata if the demo ran successfully. The following shows a sample shadow state:

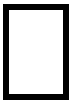


6.13.14 Fix the logging system to print out the time instead of [lu]

Issue

The reason you see [lu] at the end instead of the time is that the function IotClock_GetTimestring() in <amazon-freertos>/libraries/abstractions/platform/freertos /iot_clock_freertos.c tries to format a uint64_t variable into a char array using the “%llu” format specifier. The issue could be with the compiler not supporting 64-bit format specifiers. The fix is to print a 32-bit truncated value of the time

Fix



1. Edit the <amazon-freertos>/libraries/abstractions/platform/freertos /iot_clock_freertos.c file. Inside the IotClock_GetTimestring() function:

   Replace: `timestringLength = snprintf( pBuffer, bufferSize, "%llu", milliSeconds );`

   With: `timestringLength = snprintf( pBuffer, bufferSize, "%lu", (unsigned long) milliSeconds );`